Ιωάννης Μανωλόπουλος
Σπυρίδων Σιούτας
Αθανάσιος Τσακαλίδης
Κωνσταντίνος Τσίχλας

Ανάλυση Αλγορίθμων

Στον Ηρακλή
Στον Κερκυραϊκό
Στην Παναχαϊκή
Στον Πιερικό

Εμείς βέβαια που καταλαβαίνουμε τη ζωή, τους αριθμούς τους περιφρονούμε! Θα μ'άρεσε να αρχίσω αυτήν την ιστορία με τον τρόπο που αρχίζουν τα παραμύθια. Θα μ'άρεσε να πω.

"Μια φορά κι έναν καιρό ήταν ένας μικρός πρίγκιπας που κατοικούσε σ'έναν τοσοδούλη πλανήτη, λίγο πιο μεγάλο από το μπόι του, κι είχε ανάγκη από ένα φίλο ...". Για όσους καταλαβαίνουν τη ζωή, αυτό θα φάνταζε πιο αληθινό.

Γιατί δεν θέλω να διαβάσουν επιπόλαια το βιβλίο μου. Νιώθω τόση θλίψη, όταν αφηγούμαι αυτές τις αναμνήσεις μου. Πάνες κιόλας έξι χρόνια που έφυγε ο φίλος μου μαζί με το αρνί του. Κι αν προσπαθώ εδώ να τον περιγράψω, είναι για να μην τον ξεχάσω. Είναι λυπηρό να ξεχνάς έναν φίλο. Δεν έχουν φίλο όλοι οι άνθρωποι. Και μπορώ εγώ να καταντήσω σαν τους μεγάλους, που τους νοιάζουνε μόνο οι αριθμοί.

Αντουάν Σαιντ Εξυπερύ
Ο Μικρός Πρίγκιπας

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

0

Πρόλογος

Το αντικείμενο της Σχεδίασης και Ανάλυσης Αλγορίθμων είναι εξαιρετικά πλούσιο και έχουν γραφεί πολλά σχετικά βιβλία τόσο στη διεθνή όσο και στην ελληνική βιβλιογραφία. Το παρόν βιβλίο είναι μία προσπάθεια συλλογικής αποτύπωσης ενός υλικού και μίας εμπειρίας που συγκεντρώθηκαν μετά από πολλά χρόνια έρευνας και διδασκαλίας αντικειμένων σχετικών με Δομές Δεδομένων, Αλγορίθμους και Θεωρία Γράφων.

Σκοπός της συνθετικής αυτής προσπάθειας είναι η ομογενοποιημένη καταγραφή μίας διδακτικής προσέγγισης που θέλει να εστιάσει ιδιαιτέρως στο σκέλος της Ανάλυση Αλγορίθμων, όπως αυτή παρουσιάζεται στα αντίστοιχα κλασικά βιβλία και άρθρα, αλλά και σε συνδυασμό με την ανάλυση κλασικών Δομών Δεδομένων (κάτι που δεν είναι χρονικά εφικτό μέσα από ένα εξαμηνιαίο μάθημα Δομών Δεδομένων). Κοινός δε παρονομαστής των εξεταζόμενων αντικειμένων είναι μία εν τέλει engineering μεθοδολογία με σκοπό την επίλυση κάθε συγκεκριμένου προβλήματος μέσω πολλών εναλλακτικών τεχνικών, που συγκρίνονται και αξιολογούνται.

Ελπίζουμε ότι το συγγραφικό αυτό πόνημα θα αποτελέσει σημαντικό βοήθημα σε κάθε φοιτητή και επιστήμονα που ενδιαφέρεται για την πεμπτουσία του δομημένου και μη τετριμμένου προγραμματισμού. Σκοπίμως η βιβλιογραφία είναι εκτενής ώστε ο αναγνώστης (ο φοιτητής) να παροτρυνθεί τώρα στη μελέτη της παραχθείσας γνώσης, και αργότερα στην παραγωγή νέας γνώσης από τον ίδιο. Με τη μελέτη του βιβλίου αυτού (καθώς και άλλων αντίστοιχων), όλο και περισσότερο θα

απλώνεται η πεποίθηση στον τόπο μας, ότι η ανάπτυξη ποιοτικού λογισμικού, απαιτεί την πλήρη κατανόηση της θεωρίας των Δομών Δεδομένων και των Αλγορίθμων. Με τη βοήθεια του αναγνώστη και την κριτική του αντίληψη για το περιεχόμενο του βιβλίου αυτού και μέσα από τη γόνιμη συζήτηση κατά τη διδασκαλία του, ελπίζουμε ότι μελλοντικά το βιβλίο αυτό θα αποκτήσει μεγαλύτερη ποιοτική εμβέλεια σε σχέση με τις επιδιώξεις των υπογραφόντων.

Ιωάννης Μανωλόπουλος, Θεσσαλονίκη
Σπυρίδων Σιούτας, Κέρκυρα
Αθανάσιος Τσακαλίδης, Πάτρα
Κωνσταντίνος Τσίχλας, Θεσσαλονίκη

Δεκέμβριος 2008

1

Εισαγωγή

Περιεχόμενα Κεφαλαίου

Η έννοια του **αλγορίθμου** (algorithm) είναι γνωστή στο φοιτητή της Πληροφορικής από αρκετά μαθήματα των πρώτων ήδη εξαμήνων σπουδών του. Η έννοια είναι κεντρική για την Πληροφορική και η μελέτη της είναι πολύ ενδιαφέρουσα γιατί αποτελεί την πρώτη ύλη για την εμβάθυνση στα επιμέρους αντικείμενα της Θεωρητικής Πληροφορικής (όπως Γλώσσες και Αυτόματα, Υπολογιστική Πολυπλοκότητα, Κρυπτογραφία κλπ), καθώς και στις άλλες γνωστικές περιοχές της Πληροφορικής, όπως στις Βάσεις Δεδομένων, τα Δίκτυα, την Επεξεργασία Εικόνας, την Τεχνητή Νοημοσύνη, στον Παγκόσμιο Ιστό κοκ. Γενικότερα, ο όρος αυτός χρησιμοποιείται για να δηλώσει ένα συγκεκριμένο σύνολο βημάτων-ενεργειών για την επίλυση προβλημάτων.

Η λέξη αλγόριθμος προέρχεται από το όνομα ενός Πέρση μαθηματικού Abu Ja'far Mohammed ibn Musa al Khowarizmi που έζησε τον 9ο αιώνα μ.Χ. Η λέξη "al Khowarizmi" σημαίνει "από το Khowarazm" που είναι η σημερινή πόλη Khiva του Ουζμπεκιστάν. Η λέξη, λοιπόν, αλγόριθμος προέρχεται από το Khowarizmi, που ήταν η πατρίδα του μαθηματικού αυτού.

Μολονότι η λέξη αλγόριθμος δεν ανάγεται στο απώτατο παρελθόν, εντούτοις ουσιαστικά η έννοια είχε συλληφθεί στην Αρχαία Ελλάδα. Αρκεί να θυμηθούμε το κόσκινο του Ερατοσθένη για την εύρεση πρώτων αριθμών ή τη μέθοδο του Ευκλείδη για την εύρεση του μέγιστου κοινού διαιρέτη δύο αριθμών. Αλλά μήπως κάθε ενέργεια στην καθημερινή μας ζωή δεν μπορεί να χαρακτηρισθεί ως αλγόριθμος; Για παράδειγμα, τι είναι μία συνταγή μαγειρικής ή η διαδικασία ανάληψης χρημάτων από μία αυτόματη ταμειακή μηχανή σε μία τράπεζα, ή ακόμη η διαδικασία πολλαπλασιασμού δύο ακεραίων, πράξη που μαθαίνουμε από την πρώτη τάξη του δημοτικού σχολείου; Στα πλαίσια του βιβλίου αυτού θα ασχοληθούμε με αλγορίθμους και τεχνικές ανάλυσής τους, οι οποίοι έχουν προταθεί τα τελευταία 25 χρόνια.

1.1 Ο Αλγόριθμος ως Πρώτη Ύλη

Ένας τυπικός ορισμός της έννοιας του αλγορίθμου είναι ο εξής.

Ορισμός.
Αλγόριθμος είναι ένα πεπερασμένο σύνολο εντολών, αυστηρά καθορισμένων και εκτελέσιμων σε πεπερασμένο χρόνο, οι οποίες όταν ακολουθηθούν επιτυγχάνεται ένα επιθυμητό αποτέλεσμα ή επιλύεται ένα συγκεκριμένο πρόβλημα. □

Επιπροσθέτως, μία ακολουθία εντολών πρέπει να ικανοποιεί τα ακόλουθα κριτήρια ώστε να θεωρείται αλγόριθμος:

1. **Είσοδος** (input). Καμία, μία ή περισσότερες ποσότητες να δίνονται ως είσοδοι στον αλγόριθμο.

2. **Έξοδος** (output). Ο αλγόριθμος να δημιουργεί τουλάχιστον μία ποσότητα ως αποτέλεσμα.

3. **Καθοριστικότητα** (definiteness). Κάθε εντολή να καθορίζεται χωρίς καμία αμφιβολία για τον τρόπο εκτέλεσής της.

4. **Περατότητα** (finiteness). Ο αλγόριθμος να τελειώνει μετά από πεπερασμένα βήματα εκτέλεσης των εντολών του. Μία διαδικασία που δεν τελειώνει μετά από ένα πεπερασμένο αριθμό βημάτων λέγεται απλώς υπολογιστική διαδικασία (computational procedure).

5. **Αποτελεσματικότητα** (effectiveness). Κάθε μεμονωμένη εντολή του αλγορίθμου να επαρκώς απλή έτσι ώστε να μπορεί να εκτελεστεί από ένα άτομο με χρήση χαρτιού και μολυβιού. Δεν αρκεί δηλαδή να είναι ορισμένη (κριτήριο 3) αλλά πρέπει να είναι και εκτελέσιμη.

Από τον Niklaus Wirth (που παρουσίασε την πρώτη δομημένη γλώσσα προγραμματισμού, την Pascal) διατυπώθηκε το 1976 διατυπώθηκε η εξίσωση:

$$\text{Αλγοριθμοι} + \Delta\text{ομες } \Delta\text{εδομενων} = \Pi\text{ρογραμματα} \tag{1.1}$$

που δηλώνει ότι οι αλγόριθμοι συνυφασμένοι με τις απαραίτητες δομές σε μία αδιάσπαστη ενότητα, αποτελούν τη βάση κάθε προγράμματος, δηλαδή τελικά κάθε εφαρμογής. Επομένως, η ανάλυση των αλγορίθμων που θα μας απασχολήσει στα πλαίσια αυτού του βιβλίου, μπορεί να διεκπεραιωθεί μόνο έχοντας στο υπόβαθρο ότι οι αλγόριθμοι επενεργούν σε κάποιες συγκεκριμένες δομές δεδομένων.

Ένα άλλο σχόλιο που προκύπτει παρατηρώντας την Εξίσωση 1.1 είναι ότι η έννοια του αλγόριθμου δεν ταυτίζεται με την έννοια του προγράμματος. Το δεύτερο είναι μία υλοποίηση του πρώτου με τη βοήθεια των εντολών και των εργαλείων ενός προγραμματιστικού περιβάλλοντος. Σε σχέση με τα ανωτέρω κριτήρια, ένα πρόγραμμα δεν ικανοποιεί αναγκαστικά το 4ο κριτήριο της περατότητας (για παράδειγμα, ένα λειτουργικό σύστημα εκτελεί διαρκώς έναν ατέρμονα βρόχο περιμένοντας εντολές να εισέλθουν στο σύστημα). Στο βιβλίο αυτό θα ασχοληθούμε με αλγορίθμους που πληρούν το κριτήριο αυτό (δηλαδή, τερματίζουν πάντοτε).

Λόγω της σημαντικότητας του αντικειμένου και της κεντρικότητας τους στην επιστήμη, η Πληροφορική συχνά ορίζεται ως η επιστήμη που μελετά τους αλγορίθμους από τη σκοπιά:

1. **του υλικού**. Οι διάφορες τεχνολογίες υλικού επηρεάζουν την αποδοτικότητα ενός αλγορίθμου (όπως π.χ. στον τομέα της σχεδίασης ολοκληρωμένων κυκλωμάτων VLSI).

2. **των γλωσσών προγραμματισμού**. Το είδος της γλώσσας προγραμματισμού που χρησιμοποιείται (χαμηλού ή υψηλού επιπέδου) αλλάζει τη μορφή και τον αριθμό των εντολών ενός αλγορίθμου (σε γλώσσα assembly π.χ. η υλοποίηση των δομών επανάληψης γίνεται με χρήση της εντολής διακλάδωσης ὐ jump).

3. **τη θεωρητική σκοπιά**. Το ερώτημα που τίθεται είναι αν πράγματι υπάρχει αποδοτικός αλγόριθμος για την επιτυχία ενός αποτελέσματος. Η ερώτηση είναι πολύ σημαντική και θα συζητηθεί με περισσότερη λεπτομέρεια στα τελευταία μαθήματα.

4. **την αναλυτική σκοπιά**. Μελετώνται οι υπολογιστικοί πόροι (computer resources) που χρησιμοποιούνται από έναν αλγόριθμο, όπως το μέγεθος της κύριας και της δευτερεύουσας μνήμης, ο χρόνος για CPU και για I/O κλπ. Το αντικείμενο αυτό θα εξηγηθεί πληρέστερα στη συνέχεια.

Προφανώς εμείς στα πλαίσια αυτού του μαθήματος θα σταθούμε στην τελευταία οπτική γωνία.

Τέλος, για να τελειώνουμε με αυτά που πρέπει να θυμηθούμε και να περάσουμε σε νέα θέματα, πρέπει να αναφέρουμε και τους τρόπους παρουσίασης των αλγορίθμων, οι οποίοι είναι οι εξής τρείς:

1. με την περιγραφή του σε **φυσική γλώσσα** (natural language) ή με **ελεύθερο κείμενο** (free text). Στην περίπτωση αυτή χρειάζεται προσοχή γιατί μπορεί να παραβιασθεί το τρίτο κριτήριο του ορισμού.

2. με τη χρήση ενός **διαγράμματος ροής** (flow chart).

3. με **κωδικοποίηση** (coding), δηλαδή με ένα πρόγραμμα που όταν εκτελεσθεί θα δώσει τα ίδια αποτελέσματα με τον αλγόριθμο.

Τα διαγράμματα ροής χρησιμοποιούνταν εκτεταμένα στο παρελθόν για την αλγοριθμική περιγραφή προβλημάτων. Ωστόσο σήμερα έχουν εγκαταλειφθεί διότι: (α) δεν πληρούν μία σημαντική ιδιότητα του προγραμματισμού, δηλαδή δεν είναι δομημένα, και (β) δεν είναι λειτουργικά για σύνθετα προβλήματα. Οι αλγόριθμοι του βιβλίου αυτού είναι κωδικοποιημένοι είτε σε ψευδογλώσσα, είτε σε γλώσσα προγραμματισμού (όπως C/C++), αλλά είναι αυτονόητο ότι πολύ εύκολα μπορούν να διατυπωθούν σε οποιαδήποτε γλώσσα προγραμματισμού.

1.2 Μία Πρώτη Γεύση Αλγορίθμου

Συχνά για την επίλυση κάποιου προβλήματος μπορούν να προταθούν περισσότεροι του ενός αλγόριθμοι. Σε αυτές τις περιπτώσεις πρέπει να διαπιστωθεί ποιός είναι ο καλύτερος και υπό ποίες συνθήκες. Ας προσέξουμε τον επόμενο ψευδοκώδικα και ας προσπαθήσουμε να κατανοήσουμε τη λειτουργία του.

```
      function russe(X,Y)
1.    A[1] <-- X; B[1] <-- Y;
2.    i <-- 1;
3.    while A[i]>1 do
4.       A[i+1] <-- A[i] div 2;
5.       B[i+1] <-- B[i]+B[i];
6.       i <-- i+1;
7.    prod <-- 0;
8.    while i>0 do
9.       if (A[i] div 2=1) then prod <-- prod+B[1];
10.      i <-- i-1;
11.   return prod;
```

Ο αλγόριθμος αυτός δέχεται στην είσοδο δύο ακεραίους αριθμούς X και Y, που καταχωρίζει στις πρώτες θέσεις δύο πινάκων A και B, αντίστοιχα στις δύο εντολές 2. Στις επόμενες θέσεις του πίνακα A καταχωρίζει το ακέραιο πηλίκο της διαίρεσης του περιεχομένου της προηγούμενης θέσης δια του δύο (στην εντολή 5), ενώ στον πίνακα B καταχωρίζει δύο φορές το περιεχόμενο της προηγούμενης θέσης του ίδιου πίνακα (εντολή 6). Τέλος, με τη βοήθεια της μεταβλητής `prod` αθροίζει τα περιεχόμενα των θέσεων του πίνακα B, όταν το περιεχόμενο των αντίστοιχων θέσεων του πίνακα A είναι περιττό (εντολές 8-11).

Σε πρώτη ανάγνωση δεν είναι εύκολο να γίνει αντιληπτός ο σκοπός του αλγορίθμου αυτού. Ας εξετάσουμε ένα πρακτικό παράδειγμα για να καταλάβουμε το μηχανισμό του. Έστω ότι δίνονται οι αριθμοί 35 και 29. Το περιεχόμενο των δύο πινάκων και η τιμή της μεταβλητής `prod` διαδοχικά έχουν όπως φαίνεται στο Σχήμα 1.2.

Μπορούμε πολύ εύκολα να διαπιστώσουμε ότι η τελική τιμή (δηλαδή, το 1015) της μεταβλητής `prod` ισούται με το αποτέλεσμα του πολλαπλασιασμού 35x29. Σε επιβεβαίωση, λοιπόν, αυτού που διατυπώθηκε προηγουμένως (δηλαδή, ότι για την επίλυση κάποιου προβλήματος μπορούν να προταθούν περισσότεροι του ενός αλγόριθμοι) η συνάρτηση russe είναι ένας εναλλακτικός τρόπος για την εκτέλεση του πολλαπλασιασμού σε σχέση με την τεχνική που έχουμε διδαχθεί στο δημοτικό σχολείο. Ο τρόπος αυτός αναφέρεται σε πολλές πηγές της βιβλιογραφίας

```
          X     Y   prod
i=1      35    29     29
i=2      17    58     58
i=3       8   116
i=4       4   232
i=5       2   464
i=6       1   928    928
                     -----
                     1015
```

Σχήμα 1.1: Πολλαπλασιασμός αλά ρωσικά.

ως "πολλαπλασιασμός αλά ρωσικά" γιατί κατά την παράδοση εφαρμόζονταν στη ρωσική ύπαιθρο στο παρελθόν, αν και η πατρό/μητρότητά του διεκδικείται από πολλούς.

Αξίζει να σημειωθεί ότι αυτός ο τρόπος πολλαπλασιασμού ακεραίων είναι η βάση του αλγορίθμου που πρακτικά χρησιμοποιείται σε επίπεδο κυκλωμάτων του υπολογιστή, γιατί είναι ταχύτερος από αυτόν που γνωρίζουμε. Ο λόγος της ταχύτητάς του είναι ότι στην ουσία δεν εκτελεί πολλαπλασιασμούς αλλά προσθέσεις και πράξεις ολίσθησης που γίνονται εύκολα σε χαμηλό επίπεδο (όπως, για παράδειγμα, σε γλώσσα C/C++).

Όπως αναφέρθηκε, στις σημειώσεις αυτές θα παρουσιάσουμε τους αλγορίθμους τόσο με ψευδοκώδικα όσο και κάποια συγκεκριμένη γλώσσα προγραμματισμού. Ωστόσο, πρέπει η διαφορά μεταξύ αλγορίθμου και προγράμματος να είναι σαφής: ο πρώτος είναι γενικότερος, το δεύτερο είναι ειδικότερο του πρώτου. Αυτό γίνεται κατανοητό αν θεωρήσουμε τα όρια του υλικού. Για παράδειγμα, ένας ακέραιος αποθηκεύεται σε 4 χαρακτήρες και επομένως μπορεί να πάρει τιμές από -65.000 μέχρι 65.000. Επομένως, δεν μπορούμε να εφαρμόσουμε τον αλγόριθμο αλά ρωσικά για πολλαπλασιασμό ακεραίων με τιμές εκτός αυτών των ορίων αλλά ακόμη και για πολλαπλασιασμό ακεραίων που είναι μεν εντός των ανωτέρω ορίων αλλά δεν είναι το γινόμενό τους. Καταλήγουμε στο συμπέρασμα ότι για κάθε αλγόριθμο πρέπει να προσδιορίζουμε και το αντίστοιχο **πεδίο ορισμού** (domain of definition).

Προηγουμένως αναφερθήκαμε σε δύο αλγορίθμους πολλαπλασιασμού ακεραίων που, αν και τελείως διαφορετικοί, καταλήγουν στο σωστό αποτέλεσμα. Στο σημείο αυτό πρέπει να είμαστε προσεκτικοί κατά τη σύγκριση αλγορίθμων γιατί μπορεί δύο αλγόριθμοι να διαφέρουν ελάχιστα μεταξύ τους αλλά ενδέχεται να εκτελούν πολύ διαφορετικές λειτουργίες.

1.3 Ανάλυση Αλγορίθμων. Γιατί;

Όπως αναφέραμε, συνήθως σε κάποιο δεδομένο πρόβλημα μπορούμε να εφεύρουμε πολλές εναλλακτικές λύσεις. Είναι αναγκαίο επομένως η αξιολογική κατάταξη των εναλλακτικών λύσεων ώστε να επιλεγεί η προσφορότερη. Μιλώντας με τεχνικούς όρους, σκοπός μας είναι να προσδιορίσουμε την **επίδοση** (performance) ή την **αποδοτικότητα** (efficiency) του κάθε αλγορίθμου. Τα βασικά κριτήρια για την επιλογή ενός αλγορίθμου είναι οι απαιτήσεις του σε υπολογιστικούς πόρους, δηλαδή:

- ο απαιτούμενος χρόνος εκτέλεσης, και
- ο απαιτούμενος χώρος αποθήκευσης.

Καθώς πλέον η μνήμες (κύριες και δευτερεύουσες) γίνονται συνεχώς μεγαλύτερες και φθηνότερες, ο απαιτούμενος χρόνος εκτέλεσης είναι τελικά το σημαντικότερο κριτήριο. Άλλα κριτήρια, όπως η δικτυακή χωρητικότητα ή ο αριθμός των πυλών, είναι πολύ μικρότερης σημασίας.

Υπάρχουν δύο τρόποι για να εξετάσουμε την επίδοση ενός αλγορίθμου:

- ο **πρακτικός** ή **εκ των υστέρων** (a posteriori), και
- ο **θεωρητικός** ή **εκ των προτέρων** (a priori).

Σύμφωνα με τον πρώτο τρόπο, κάθε αλγόριθμος εξετάζεται σε έναν υπολογιστή και χρονομετρείται καθώς η είσοδός του τροφοδοτείται με διάφορα δεδομένα. Αυτός ο τρόπος δεν είναι σωστός αν θεωρήσουμε ότι δεν λαμβάνει υπόψη του μία σειρά παραμέτρων, όπως:

- το χρησιμοποιούμενο υλικό, δηλαδή, τα συγκεκριμένα χαρακτηριστικά του υπολογιστή, όπως ταχύτητα του επεξεργαστή, το μέγεθος της κύριας ή της δευτερεύουσας μνήμης, τη χρήση κρυφής μνήμης (cache) κοκ.
- τη χρησιμοποιούμενη γλώσσα προγραμματισμού, καθώς είναι γνωστό ότι υπάρχουν ταχύτερες και βραδύτερες γλώσσες με περισσότερη ή λιγότερη πρόσβαση στο φυσικό επίπεδο (όπως, για παράδειγμα οι παλαιότερες γλώσσες σε αντίθεση με τις νεότερες C/C++).
- το χρησιμοποιούμενο μεταγλωττιστή/ερμηνευτή, καθώς είναι δυνατόν να υπάρχουν διαθέσιμοι εμπορικά (ή και δωρεάν μέσα από το διαδίκτυο) περισσότεροι του ενός μεταγλωττιστές που μάλιστα μπορεί να τρέχουν σε διαφορετικά λειτουργικά συστήματα (όπως, για παράδειγμα, Unix, Linux, Windows κοκ).

- τον προγραμματιστή που χρησιμοποιεί όλα τα προηγούμενα, καθώς ο ανθρώπινος παράγοντας είναι σημαντικότατος σε κάθε περίπτωση, και τέλος

- τα χρησιμοποιούμενα δεδομένα, γιατί επί παραδείγματι γνωρίζουμε ότι κάθε αλγόριθμος ταξινόμησης συμπεριφέρεται διαφορετικά ανάλογα με τη φύση των δεδομένων στην είσοδο.

Καταλήγουμε, λοιπόν, ότι δεν πρέπει να χρησιμοποιούμε τον πρακτικό τρόπο ανάλυσης αλγορίθμων αλλά το θεωρητικό που μας γλιτώνει τον απαιτούμενο χρόνο του προγραμματιστή και του μηχανήματος.

Μία βασική παράμετρος για τη θεωρητική ανάλυση είναι το **μέγεθος του προβλήματος** (problem size). Για παράδειγμα, στην περίπτωση της ταξινόμησης n αριθμών, του πολλαπλασιασμού δύο τετραγωνικών πινάκων $n \times n$, ή της διάσχισης ενός δένδρου με n κόμβους, λέγεται ότι το μέγεθος του προβλήματος είναι n. Είναι δυνατόν το μέγεθος ενός προβλήματος να εκφράζεται με δύο αριθμούς αντί για έναν. Για παράδειγμα, ένας γράφος διακρίνεται από το πλήθος των κορυφών n αλλά και το πλήθος των ακμών m.

Δεδομένου, λοιπόν, ενός προβλήματος και ενός ή περισσοτέρων αλγορίθμων για την επίλυση του, σκοπός μας είναι η εύρεση της **χρονικής πολυπλοκότητας** (time complexity) και της **χωρικής πολυπλοκότητας** (space complexity) τους ως συνάρτησης του n. Την πολυπλοκότητα εκφράζουμε με τη βοήθεια ειδικών **συμβολισμών** (notation), που ίσως γνωρίζουμε από τα μαθήματα των προηγουμένων εξαμήνων. Αυτοί οι συμβολισμοί είναι: O, Ω, Θ, ο και ω, και εξ αυτών οι σπουδαιότεροι και συχνότερα χρησιμοποιούμενοι είναι οι τρεις πρώτοι. Η έννοια της πολυπλοκότητας και οι έννοιες των συμβολισμών είναι τεχνικές και για το λόγο αυτό πρέπει να ορισθούν προσεκτικά στο Κεφάλαιο 2.

1.4 Κοστολόγηση Πράξεων και Εντολών

Πριν οποιαδήποτε αναφορά σε ανάλυση αλγορίθμων, είναι αναγκαίο να μελετήσουμε τα μοντέλα υπολογισμού στα οποία εκτελούνται οι αλγόριθμοί μας. Τα μοντέλα που θα εξετάσουμε είναι: (α) η **Μηχανή Τυχαίας Προσπέλασης** (Random Access Machine - RAM), και (β) η **Μηχανή Δεικτών** (Pointer Machine - PM).

Μια υπολογιστική μηχανή με βάση το μοντέλο RAM έχει τα εξής χαρακτηριστικά:

- έχει 1 επεξεργαστή, 4 καταχωρητές, 1 συσσωρευτή , 3 καταχωρητές δείκτη (index registers) και ένα άπειρο πλήθος θέσεων μνήμης, οι οποίες είναι αριθμημένες από το 0.

- είναι δυνατή η προσπέλαση μνήμης με υπολογισμό διεύθυνσης, και
- οι εντολές είναι μονής διεύθυνσης (one-address), δηλαδή κάθε εντολή αναφέρεται μόνο σε μια θέση μνήμης.

Το μοντέλο της μηχανής RAM ανταποκρίνεται στη δομή των σύγχρονων υπολογιστών και θεωρεί ότι οι εντολές του αλγορίθμου μας εκτελούνται η μία μετά την άλλη και όχι ταυτόχρονα. Κάθε εντολή περιλαμβάνει την εκτέλεση βασικών πράξεων σε δύο τιμές (όπως, χαρακτήρες, ακεραίους κλπ) που είναι αποθηκευμένες στη μνήμη του υπολογιστή. Σημειώνεται ότι υπάρχει και το μοντέλο της **Παράλληλης Μηχανής Τυχαίας Προσπέλασης** (Parallel Random Access Machine - PRAM), όπου θεωρείται ότι υπάρχουν πολλοί επεξεργαστές, οπότε είναι δυνατόν εντολές περισσότερες της μίας να εκτελούνται ταυτόχρονα. Το μοντέλο PRAM δεν θα μας απασχολήσει στη συνέχεια.

Το μοντέλο PM παρουσιάζει τα ίδια χαρακτηριστικά με το μοντέλο RAM με τη διαφορά ότι δεν επιτρέπει την προσπέλαση με υπολογισμό διεύθυνσης. Προσπέλαση σε δεδομένη διεύθυνση είναι δυνατή μόνο αν υπάρχει δείκτης σε αυτή τη διεύθυνση. Το μοντέλο PM είναι ασθενέστερο του RAM. Όμως, οι αλγόριθμοι που αναπτύσσονται όμως σε PM είναι συχνά κομψότεροι, ενώ παρέχουν ταυτόχρονα διαίσθηση για την εγγενή δυσκολία του προβλήματος.

Για να εκφράσουμε την αποτελεσματικότητα ενός αλγορίθμου (με βάση το μοντέλο RAM) πρέπει να εκτιμήσουμε το κόστος εκτέλεσης κάθε πράξης ή/και κάθε εντολής, να υπολογίσουμε πόσες φορές θα εκτελεσθεί κάθε εντολή και να αθροίσουμε τα επί μέρους κόστη ώστε να βρούμε το συνολικό κόστος. Αρχικά ας θεωρήσουμε το κόστος των πράξεων.

Ένα πρόγραμμα ή ένας αλγόριθμος μπορεί να αποτελείται από πολλών ειδών πράξεις, όπως: αριθμητικές πράξεις (πρόσθεση, αφαίρεση, πολλαπλασιασμός, διαίρεση, πράξη modulo), λογικές (boolean) πράξεις, συγκρίσεις (εντολές if, for, while, repeat until), καταχωρίσεις (assignment) κοκ. Όλες αυτές τις πράξεις ονομάζουμε **στοιχειώδεις** (elementary). Απαραίτητη προϋπόθεση για να θεωρηθεί μία πράξη ως στοιχειώδης είναι να απαιτεί χρόνο εκτέλεσης φραγμένο από επάνω από μία σταθερά που να εξαρτάται μόνο από το συγκεκριμένο περιβάλλον υλοποίησης (όπως, το χρησιμοποιούμενο μηχάνημα, τη γλώσσα προγραμματισμού, το μεταγλωττιστή, κλπ). Στην πράξη είναι γνωστό ότι αυτές οι στοιχειώδεις πράξεις που αναφέρθηκαν δεν έχουν το ίδιο χρονικό κόστος εκτέλεσης. Για παράδειγμα, η πράξη του πολλαπλασιασμού είναι πλέον χρονοβόρα από την πράξη της πρόσθεσης. Ωστόσο, για λόγους ευκολίας θεωρούμε ότι όλες οι στοιχειώδεις πράξεις έχουν **μοναδιαίο κόστος** (unit cost).

Χρειάζεται προσοχή ώστε να μην παρεξηγηθεί αυτή η έννοια του μοναδιαίου κόστους. Για παράδειγμα, η εντολή `for i <-- 1 to n do` ή η εντολή `x <--`

`n!` αποτελούνται από πολλές απλούστερες πράξεις που πρέπει όλες να καταμετρηθούν. Πιο συγκεκριμένα, στην πρώτη περίπτωση υπεισέρχεται ένα πλήθος προσθέσεων, συγκρίσεων και καταχωρίσεων, ενώ στη δεύτερη περίπτωση υπεισέρχεται αντίστοιχα ένα πλήθος πολλαπλασιασμών, συγκρίσεων και καταχωρίσεων.

Όλα τα ανωτέρω ισχύουν με την προϋπόθεση ότι το περιεχόμενο κάθε μεταβλητής χωρά σε μια θέση μνήμης. Για παράδειγμα, είναι γνωστό ότι ισχύει 8!=40.320 αλλά 9!=362.880, αριθμός που υπερβαίνει τα όρια του απλού ακεραίου. Το ίδιο πρόβλημα μπορεί να προκύψει κατά τον υπολογισμό των αριθμών Fibonacci, σύμφωνα με τον επόμενο αλγόριθμο (θεωρώντας το $n \geq 0$ ακέραιο αριθμό).

```
     function fib(n)
1.   i <-- 1; j <-- 0;
2.   for k <-- 1 to n do
3.      j <-- i+j;
4.      i <-- j-1;
5.   return j;
```

Αν, λοιπόν, χρησιμοποιήσουμε αυτό το απλούστατο ψευδοκώδικα για τον υπολογισμό του F_{47}, τότε θα προκύψει υπερχείλιση, ενώ για την καταχώριση του αριθμού F_{65535} απαιτούνται 45.496 bits. Άρα, όταν τα περιεχόμενα των μεταβλητών δεν χωρούν σε μια θέση μνήμης, τότε οι πράξεις στοιχίζουν όσο και το μήκος των δεδομένων, δηλαδή για τον ακέραιο n, κάθε πράξη στοιχίζει $\lfloor \log n \rfloor + 1$. Το μοντέλο μηχανής που βασίζεται σε αυτή την υπόθεση εργασίας, ονομάζεται RAM **λογαριθμικού κόστους** (logaritmic cost), και θα εξετασθεί στο Κεφάλαιο 6.

Αντίστοιχα προβλήματα μπορεί να εμφανισθούν σε πραγματικούς αριθμούς, όπου μπορεί να απαιτείται μικρότερη ή μεγαλύτερη ακρίβεια. Επομένως, μπορεί για έναν αλγόριθμο ο πολλαπλασιασμός να θεωρείται ως στοιχειώδης πράξη αλλά σε ένα πρόγραμμα θα πρέπει να είμαστε βέβαιοι για το περιβάλλον πραγματικής λειτουργίας του. Ο λόγος είναι ότι ίσως δεν θα αρκούν τα μεγέθη των αντίστοιχων τύπων, οπότε για την αποφυγή αριθμητικής υπερχείλισης θα πρέπει να χρησιμοποιηθούν τύποι μεγαλύτερου μεγέθους, και επομένως μεγαλύτερου κόστους.

Η οπτική γωνία έκφρασης της επίδοσης ενός αλγορίθμου εκτιμώντας και αθροίζοντας τα επί μέρους κόστη των πράξεων χρησιμοποιήθηκε σε μεγάλη έκταση από τον Donald Knuth στο τρίτομο ιστορικό του βιβλίο The Art of Computer Programming, το οποίο εκδόθηκε τη δεκαετία του 70. Η προσέγγιση αυτή είναι η λεπτομερέστερη και ακριβέστερη αλλά και η πλέον επίπονη. Αν και η θεώρηση του κόστους των πράξεων βοηθά στην κατανόηση των επί μέρους παραγόντων κόστους και τελικά στη μείωση του συνολικού κόστους, εν τούτοις

δεν χρησιμοποιείται στη βιβλιογραφία καθώς έχει επικρατήσει μία μακροσκοπικότερη προσέγγιση που εξετάζει το κόστος των εντολών.

Για να εκφράσουμε την πολυπλοκότητα ενός αλγορίθμου θεωρώντας το κόστος των εντολών (και με βάση το μοντέλο RAM) πρέπει να εκτιμήσουμε το κόστος εκτέλεσης κάθε εντολής, να υπολογίσουμε το πλήθος εκτελέσεων κάθε εντολής και να αθροίσουμε τα επί μέρους κόστη ώστε να βρούμε το συνολικό κόστος. Και πάλι είναι ευνόητο ότι το κόστος κάθε εντολής μπορεί να είναι διαφορετικό. Για παράδειγμα, από τη συνάρτηση `russe` αξίζει να προσέξουμε τις τρεις εντολές: `i <-- 1` (εντολή 3), `X[i+1] <-- X[i] div 2` (εντολή 5) και `if X[i] is odd then prod <-- prod+Y[1]` (εντολή 10), ώστε αμέσως να διαπιστώσουμε την αλήθεια της πρότασης αυτής. Για το λόγο αυτό θεωρούμε ότι η i-οστή εντολή ενός αλγορίθμου έχει ένα σταθερό κόστος c_i που είναι φραγμένο από επάνω για οποιοδήποτε περιβάλλον υλοποίησης.

Στη συνέχεια θα εξετάσουμε την προηγούμενη κοστολόγηση εντολών σε ένα λεπτομερές πρόγραμμα (κατ'εξαίρεση και όχι σε έναν απλό αλγόριθμο). Δηλαδή, θα εφαρμόσουμε μία **επακριβή** (exact) ανάλυση σε ένα πρόγραμμα C υπολογίζοντας το κόστος εκτέλεσης γραμμή-γραμμή σε συνδυασμό με το πλήθος των εκτελέσεων της κάθε γραμμής. Το πρόγραμμα που θα δοκιμάσουμε υλοποιεί την ταξινόμηση επιλογής, που γνωρίζουμε από το μάθημα των Δομών Δεδομένων.

```
     procedure select
1.   for (i=0; i<n; ++i) {
2.      min=i;
3.      for (j=i+1; j<=n; ++j)
4.         if (A[j]<A[min]) min=j;
5.      temp=A[i];
6.      A[i]=A[min];
7.      A[min]=temp;
8.   }
```

Ο Πίνακας 1.4 δίνει για κάθε γραμμή το κόστος της γραμμής και τον αντίστοιχο αριθμό επαναλήψεων. Για να βρούμε τη συνολική επίδοση του προγράμματος αθροίζουμε τα επιμέρους κόστη και έχουμε:

Γραμμή	Κόστος	Αριθμός Εκτελέσεων
1	c_1	$n+1$
2	c_2	n
3	c_3	$\sum_{j=1}^{n} j$
4	c_4	$\sum_{j=1}^{n} j-1$
5	c_5	n
6	c_6	n
7	c_7	n

Πίνακας 1.1: Κοστολόγηση ταξινόμησης επιλογής

$$\begin{aligned} T(n) &= c_1(n+1)+c_2n+c_3\sum_{j=1}^{n} j+c_4\sum_{j=1}^{n}(j-1)+c_5n+c_6n+c_7n \\ &= c_1n+c_1+c_2n+c_3\frac{n(n+1)}{2}+c_4\frac{n(n+1)}{2}-c_4n+c_5n+c_6n+c_7n \\ &= (c_3/2+c_4/2)n^2+(c_1+c_2+c_3/2-c_4/2+c_5+c_6+c_7)n+c_1 \\ &= an^2+bn+c \end{aligned}$$

Παρατηρούμε ότι το τελικό κόστος είναι μία τετραγωνική συνάρτηση και ότι εξαρτάται από το μέγεθος της εισόδου, δηλαδή το n, και όχι από το είδος της εισόδου αλλά κυρίως δεν εξαρτάται ασυμπτωτικά από συγκεκριμένες τιμές των κοστών c_i. Στη συνέχεια θα εφαρμόσουμε την ίδια τεχνική για το πρόγραμμα σε C της ταξινόμηση με εισαγωγή, που επίσης γνωρίζουμε από το μάθημα των Δομών Δεδομένων.

```
     procedure insert
1.   for(j=2; j<=n; ++j) {
2.      key=A[j];
3.      i=j-1;
4.      while (i>0 && A[i]>key)
5.         A[i+1]=A[i]
6.         i=i-1;
7.      A[i+1]=key
8.   }
```

Όπως προηγουμένως, στον Πίνακα 1.4 παρουσιάζονται τα αντίστοιχα κόστη ανά γραμμή. Στον πίνακα αυτόν με t_j συμβολίζεται το πλήθος των εκτελέσεων

Γραμμή	Κόστος	Αριθμός Εκτελέσεων
1	c_1	n
2	c_2	$n-1$
3	c_3	$n-1$
4	c_4	$\sum_{j=2}^{n} t_j$
5	c_5	$\sum_{j=2}^{n} t_j - 1$
6	c_6	$\sum_{j=2}^{n} t_j - 1$
7	c_7	$n-1$

Πίνακας 1.2: Κοστολόγηση ταξινόμησης εισαγωγής

του βρόχου `while` για τις διάφορες τιμές του `j`. Όπως προηγουμένως, για να βρούμε το συνολικό κόστος του προγράμματος αθροίζουμε τα επί μέρους κόστη και έχουμε:

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n} (t_j - 1) + \\ &\quad c_6 \sum_{j=2}^{n} (t_j - 1) + c_7(n-1) \\ &= (c_4 + c_5 + c_6) \sum_{j=2}^{n} t_j + (c_1 + c_2 + c_3 - c_5 - c_6 + c_7)n \\ &\quad + \ (-c2 - c3 + c5 + c6 - c7) \end{aligned}$$

Επομένως για να επιλυθεί περαιτέρω η εξίσωση $T(n)$ πρέπει να εστιάσουμε στην τιμή της μεταβλητής t_j. Αν ο πίνακας είναι ήδη ταξινομημένος, τότε ισχύει $t_j = 1$, οπότε με εύκολη άλγεβρα καταλήγουμε ότι η συνάρτηση $T(n)$ είναι γραμμική. Αυτή είναι η καλύτερη δυνατή περίπτωση. Η χειρότερη περίπτωση συμβαίνει όταν θέλουμε να ταξινομήσουμε κατά αύξουσα σειρά αλλά τα στοιχεία εισέρχονται στην είσοδο κατά φθίνουσα σειρά. Τότε ισχύει $t_j = j$, οπότε και πάλι με εύκολη άλγεβρα καταλήγουμε ότι η συνάρτηση $T(n)$ είναι τετραγωνική. Η μέση περίπτωση εξαρτάται από την κατανομή των στοιχείων και τις μεταξύ τους σχέσεις και είναι δυσκολότερο να βρεθεί επακριβώς. Επί του παρόντος, ας αρκεσθούμε στη διαβεβαίωση ότι και στην περίπτωση αυτή η αντίστοιχη $T(n)$ είναι τετραγωνική συνάρτηση.

Όπως και στο προηγούμενο παράδειγμα, καταλήγουμε και τώρα στο απλό συμπέρασμα ότι το συνολικό κόστος δεν εξαρτάται από τις τιμές των επί μέρους

κοστών c_i καθώς αυτά εμφανίζονται στους σταθερούς συντελεστές και δεν επηρεάζουν την ασυμπτωτική συμπεριφορά του αλγορίθμου/προγράμματος, κάτι που ισχύει για το μέγεθος του προβλήματος n. Επομένως συμπεραίνουμε ότι ο τρόπος αυτός κοστολόγησης, αν και επακριβής, είναι υπερβολικός και τελικά μη πρακτικός. Προκειμένου, λοιπόν, να εκτιμήσουμε την επίδοση ενός αλγορίθμου, πρέπει να απλοποιήσουμε την κατάσταση ακόμη περισσότερο και να εστιάσουμε στα κρίσιμα σημεία του αλγορίθμου.

1.5 Επιλογή του Βαρόμετρου

Παρατηρώντας προσεκτικότερα τα δύο προηγούμενα προγράμματα, παρατηρούμε ότι το μεγαλύτερο κομμάτι του κόστους προέρχεται από την εκτέλεση εντολών μέσα σε βρόγχους και όχι από την εκτέλεση εντολών αρχικοποίησης. Άρα πρέπει να εστιάσουμε εκεί και να αγνοήσουμε τα σημεία του προγράμματος/αλγορίθμου όπου το κόστος θα είναι μικρότερο από τα προαναφερθέντα σημεία.

Ας θεωρήσουμε την επόμενη εκδοχή αλγορίθμου ταξινόμησης με επιλογή και ας εστιάσουμε στην εντολή 3 του εσωτερικού βρόχου και πιο συγκεκριμένα τόσο στις εντολές καταχώρισης 5-6 μέσα στην εντολή ελέγχου `if` (εντολή 4), όσο και στην ίδια εντολή `if` που ενσωματώνει άλλες εντολές.

```
     procedure select
1.   for i <-- 1 to n-1 do
2.      minj <-- i; minx <-- A[i];
3.      for j <-- i+1 to n do
4.         if A[j]<minx then
5.            minj <-- j;
6.            minx <-- A[j];
7.      A[minj] <-- A[i];
8.      A[i] <-- minx;
```

Αν υποθέσουμε ότι το κόστος μίας εκτέλεσης του εσωτερικού τμήματος του εσωτερικού βρόχου (δηλαδή, μίας εκτέλεσης των εντολών 4-6) είναι φραγμένο από επάνω από μία σταθερά c_1. Για μία συγκεκριμένη τιμή της μεταβλητής i, ο αριθμός των εκτελέσεων θα είναι $n-i$ και επομένως το αθροιστικό κόστος εκτέλεσης των εντολών 4-6 θα είναι $c_1(n-i)$. Αν συνυπολογίσουμε το κόστος αρχικοποίησης της εντολής 3, τότε το μέχρι στιγμής κόστος για τις εντολές 3-6 είναι $c_2+c_1(n-i)$. Στη συνέχεια εξετάζοντας τον εξωτερικό βρόχο, διακρίνουμε ένα σταθερό κόστος c_3 λόγω των δύο εντολών 2, οπότε το κόστος εκτέλεσης του εσωτερικού τμήματος του εξωτερικού βρόχου (δηλαδή των εντολών 2-8) θα

είναι: $\sum_{i=1}^{n-1}(c_3+c_2+c_1(n-i))$. Τέλος, στο κόστος αυτό πρέπει να προσθέσουμε ένα σταθερό κόστος c_4 για την αρχικοποίηση της εντολής 1. Αν απλοποιήσουμε την προκύπτουσα έκφραση, τότε εύκολα φθάνουμε στην επόμενη έκφραση:

$$T(n) \;=\; \frac{c_1}{2}\; n^2 + (c_2 + c_3 - \frac{c_1}{2})\; n + (c_4 - c_3 - c_2)$$

που διαπιστώνουμε ότι είναι και πάλι τετραγωνική. Καταλήξαμε, λοιπόν, στο ίδιο συμπέρασμα μέσα από έναν άλλο δρόμο, αναζητώντας το κρίσιμο σημείο κόστους που ονομάζουμε **βαρόμετρο** (barometer). Σε πρακτικό επίπεδο η επιλογή του βαρόμετρου είναι σχετικά εύκολη υπόθεση, αρκεί να εφαρμόσουμε μία σειρά απλών κανόνων:

- το κόστος εκτέλεσης μίας εντολής εντός ενός βρόχου προκύπτει από το γινόμενο του κόστους μίας εκτέλεσης της εντολής επί τον αριθμό των επαναλήψεων,

- το κόστος εκτέλεσης μίας εντολής εντός ενός φωλιασμένου βρόχου προκύπτει από το γινόμενο του κόστους μίας εκτέλεσης της εντολής επί τον αριθμό των επαναλήψεων όλων των βρόχων (επιχειρούμε την ανάλυση από μέσα προς τα έξω).

- σε περίπτωση διαδοχικών εντολών εστιαζόμαστε στην εντολή με το μεγαλύτερο κόστος,

- το κόστος εκτέλεσης μίας εντολής ελέγχου (if) ισούται με το κόστος του ελέγχου συν το κόστος της διακλάδωσης με το μεγαλύτερο κόστος.

Συνήθως εστιαζόμαστε στο εσωτερικότερο σημείο των βρόχων, με προτίμηση στους πολυπλοκότερους (δηλαδή, στους φωλιασμένους με περισσότερα επίπεδα). Ωστόσο, χρειάζεται ιδιαίτερη προσοχή στο χειρισμό των ορίων των βρόχων. Το επόμενο παράδειγμα είναι χαρακτηριστικό. Ο μικρός αυτός ψευδοκώδικας υποθέτει ότι υπάρχει ένας πίνακας A με n ακεραίους, για τους οποίους ισχύει $0 \leq A[i] \leq i$, ενώ επίσης δίνεται ότι το άθροισμα αυτών των ακεραίων ισούται με s.

```
1.   k <-- 0;
2.   for i <-- 1 to n do
3.      for j <-- 1 to A[i] do
4.         k <-- k + A[j];
```

Για κάθε τιμή της μεταβλητής i, η εντολή 4 θα εκτελεσθεί $A[i]$ φορές, επομένως συνολικά για όλες τις τιμές της μεταβλητής i (δηλαδή από 1 μέχρι n), το πλήθος των επαναλήψεων θα είναι $\sum_{i=1}^{n} A[i] = s$. Θα μπορούσε κάποιος να υποστηρίξει ότι η επίδοση του μικρού αυτού ψευδοκώδικα είναι της τάξης του s. Για αντιπαράδειγμα, όμως, ας υποθέσουμε ότι $A[i] = 1$ αν το i είναι τέλειο τετράγωνο, αλλιώς ισχύει $A[i] = 0$. Στην περίπτωση αυτή ισχύει $s = \sqrt{n}$. Όμως κάτι τέτοιο δεν μπορεί να ισχύει γιατί οπωσδήποτε θα ελεγχθούν όλες οι θέσεις του πίνακα A μεγέθους n (και τουλάχιστον μία φορά η κάθε μία).

Αυτό που δεν ελήφθη υπόψη στην προηγούμενη προσέγγιση είναι ότι σε κάθε περίπτωση θα γίνουν οι απαραίτητοι έλεγχοι στην εντολή 3, ανεξαρτήτως αν θα εκτελεσθεί η εντολή 4. Σε ενίσχυση του γεγονότος αυτού, ας προσέξουμε και πάλι τα παραδείγματα της ταξινόμησης με επιλογή και εισαγωγή της προηγουμένης παραγράφου, όπου οι εντολές των βρόχων εκτελούνται μία φορά περισσότερο από τις εντολές εντός του βρόχου.

Η ορθή προσέγγιση, λοιπόν, πρέπει να ακολουθήσει τη μεθοδολογία του παραδείγματος της ταξινόμησης με επιλογή. Αν c_1 είναι το κόστος μίας εκτέλεσης του εσωτερικού βρόχου, ενώ c_2 είναι το κόστος αρχικοποίησης του βρόχου αυτού, τότε το κόστος των εντολών 3-4 είναι $c_2 + c_1 A[i]$. Με παρόμοιο τρόπο, αν c_3 είναι το κόστος μίας εκτέλεσης του εξωτερικού βρόχου, ενώ c_4 είναι το κόστος αρχικοποίησης του βρόχου αυτού, τότε προφανώς το κόστος των εντολών 2-4 είναι $c_4 + \sum_{i=1}^{n}(c_3 + c_2 + c_1 A[i])$. Αν αυτή η έκφραση απλοποιηθεί, τότε προκύπτει $(c_3 + c_2)n + c_1 s + c_4$. Συνεπώς, το κόστος του προηγούμενου ψευδοκώδικα είναι γραμμική συνάρτηση δύο μεταβλητών, του n και του s.

Ανεξαρτήτως της ιδιαιτερότητας του προηγούμενου παραδείγματος, ισχύει η παρατήρηση της αρχής της παρούσας παραγράφου, δηλαδή ότι το μεγαλύτερο κομμάτι του κόστους προέρχεται από την εκτέλεση εντολών μέσα σε βρόγχους και εκεί θα πρέπει σε κάθε περίπτωση να εστιάζουμε. Πιο συγκεκριμένα, σε πολλούς αλγορίθμους (και κατ'εξοχήν σε αλγορίθμους ταξινόμησης) είναι το πλήθος των πραγματοποιούμενων συγκρίσεων μέσα σε ένα βρόχο η σημαντικότερη πράξη που θα πρέπει να υπολογίζεται με σκοπό την εύρεση της επίδοσης του αλγορίθμου.

1.6 Ανάλυση Μέσης και Χειρότερης Περίπτωσης

Μία συμβολοσειρά (string) λέγεται **παλίνδρομο** (palindrome) αν διαβάζεται το ίδιο είτε από την αρχή είτε από το τέλος (θυμηθείτε το βυζαντινό "νίψον ανομήματα μη μόναν όψιν"). Ο ψευδοκώδικας που ακολουθεί βρίσκει ένα παλίνδρομο σε μία συμβολοσειρά εισόδου που αποθηκεύεται σε ένα πίνακα `S` μήκους `n`. Τον

αλγόριθμο αυτό θα αναλύσουμε ώστε να γνωρίζουμε τη συμπεριφορά του σε τρεις περιπτώσεις: την καλύτερη, τη χειρότερη και τη μέση.

```
     procedure palindrome
1.   left <-- 1; right <-- n; flag <-- false;
2.   while (left<right) and (S[left]=S[right]) do
3.      left <-- left+1; right <-- right-1;
4.   if (left>=right) then flag <-- true;
```

Για ευκολία της ανάλυσης θα υποθέσουμε ότι η συμβολοσειρά είναι δυαδική. Η κρίσιμη εντολή βαρόμετρο είναι η εντολή 3 μέσα στο βρόχο `while`. Αν ισχύει `S[1]<>S[n]`, δηλαδή διαφέρει ο πρώτος και ο τελευταίος χαρακτήρας, τότε δεν εκτελείται ο βρόχος και ο αλγόριθμος τερματίζει. Αυτή είναι η καλύτερη περίπτωση και επομένως συμπεραίνουμε ότι στην περίπτωση αυτή η πολυπλοκότητα είναι $\Theta(1)$. Η χειρότερη περίπτωση συμβαίνει όταν η συμβολοσειρά είναι ένα παλίνδρομο, οπότε ο βρόχος θα εκτελεσθεί $n/2$ φορές. Στην περίπτωση αυτή η πολυπλοκότητα είναι $\Theta(n)$.

Για να μελετήσουμε τη μέση περίπτωση θα πρέπει να εξετάσουμε όλες τις συμβολοσειρές μήκους n. Καθώς υποθέτουμε ότι το αλφάβητό μας έχει μόνο δύο χαρακτήρες, έπεται ότι υπάρχουν 2^n διαφορετικές συμβολοσειρές. Κατ'αρχήν υποθέτουμε ότι το n είναι άρτιος αριθμός. Επομένως $2^n/2$ συμβολοσειρές διαφέρουν στον πρώτο και τελευταίο χαρακτήρα, όποτε δεν θα εκτελεσθεί το σώμα του βρόχου `while` (εντολή 3). Επίσης, $2^n/2^2$ συμβολοσειρές διαφέρουν στο δεύτερο και προτελευταίο χαρακτήρα, όποτε το σώμα του βρόχου `while` θα εκτελεσθεί μία φορά. Με το ίδιο σκεπτικό, $2^n/2^{n/2}$ συμβολοσειρές διαφέρουν στους δυο κεντρικούς χαρακτήρες, όποτε το σώμα του βρόχου θα εκτελεσθεί $n/2-1$ φορές. Τελικά, ο αριθμός των παλίνδρομων είναι $2^{n/2}$, οπότε θα γίνουν $n/2$ επαναλήψεις του βρόχου. Συνεπώς, η μέση τιμή των επαναλήψεων δίνεται από τη σχέση:

$$T_{even}(n) \;=\; \frac{1}{2^n}\left(\sum_{i=0}^{n/2-1}\frac{2^n}{2^{i+1}}i + 2^{n/2}\frac{n}{2}\right) \;=\; \sum_{i=1}^{n/2-1}\frac{i}{2^{i+1}} + \frac{n/2}{2^{n/2}}$$

Κατ' αρχήν θα επιλύσουμε το άθροισμα και θα επανέλθουμε. Πρώτον, λοιπόν, αναπτύσσουμε το άθροισμα και δεύτερον πολλαπλασιάζουμε επί δύο.

$$\begin{aligned}\sum_{i=0}^{n/2-1}\frac{i}{2^{i+1}} &= A = \frac{1}{2^2}+\frac{2}{2^3}+\frac{3}{2^4}+\ldots+\frac{n/2-1}{2^{n/2}} \Rightarrow \\ 2A &= \frac{1}{2^1}+\frac{2}{2^2}+\frac{3}{2^3}+\ldots+\frac{n/2-1}{2^{n/2-1}}\end{aligned}$$

Αφαιρώντας κατά μέλη και απλοποιώντας προκύπτει:

$$A = \frac{1}{2^1}+\frac{1}{2^2}+\frac{1}{2^3}+\ldots+\frac{1}{2^{n/2-1}}-\frac{n/2-1}{2^{n/2}} = \sum_{i=1}^{n/2}\frac{1}{2^i}-\frac{n/2}{2^{n/2}}$$

Τώρα πρέπει να επιλυθεί το τελευταίο άθροισμα. Και πάλι με ανάπτυξη και πολλαπλασιασμό επί δύο και αφαίρεση κατά μέλη προκύπτει:

$$\begin{aligned}\sum_{i=1}^{n/2}\frac{1}{2^i} &= B = \frac{1}{2^1}+\frac{1}{2^2}+\frac{1}{2^3}+\ldots+\frac{1}{2^{n/2}} \Rightarrow \\ 2B &= \frac{1}{2^0}+\frac{1}{2^1}+\frac{1}{2^2}+\ldots+\frac{1}{2^{n/2-1}} \Rightarrow \\ B &= \frac{1}{2^0}-\frac{1}{2^{n/2}}\end{aligned}$$

Επομένως τελικά ισχύει:

$$T_{even}(n) = 1-\frac{1}{2^{n/2}}$$

Όταν το n είναι περιττό πρέπει να λάβουμε υπόψη ότι τώρα υπάρχουν δύο φορές $2^{\frac{n-1}{2}}$ παλίνδρομα με διαφορετικούς χαρακτήρες στην κεντρική θέση και ότι το προηγούμενο άθροισμα έχει πλέον $\frac{n-1}{2}$ όρους. Με κατάλληλη άλγεβρα προκύπτει ότι:

$$T_{odd}(n) = 1-\frac{1}{2^{\frac{n-1}{2}}}$$

Επομένως, τελικά προκύπτει ότι η πολυπλοκότητα στη μέση περίπτωση είναι σταθερή $\Theta(1)$.

Η ανάλυση για κάθε μία από τις τρεις περιπτώσεις (καλύτερη, χειρότερη και μέση) έχει τη δική της αξία και χρησιμότητα. Συνήθως η καλύτερη περίπτωση δεν μας ενδιαφέρει όσο οι άλλες δύο: η χειρότερη και η μέση. Ειδικότερα, στην πράξη μας ενδιαφέρει κατ'αρχήν η μέση περίπτωση αλλά σε πολλές κρίσιμες

εφαρμογές μας ενδιαφέρει και η χειρότερη. Για παράδειγμα, αν φαντασθούμε ένα σύστημα ελέγχου πυρηνικού αντιδραστήρα, τότε είναι βέβαιο ότι εκεί θα μας ενδιέφερε η χειρότερη περίπτωση των εφαρμοζόμενων αλγορίθμων και όχι η μέση συμπεριφορά τους. Στο σημείο αυτό, λοιπόν, θεωρούμε ότι είναι πλέον κατανοητές οι έννοιες της ανάλυσης της καλύτερης, μέσης και χειρότερης περίπτωσης. Στο Κεφάλαιο 2 θα εισαχθούν με τυπικό τρόπο οι έννοιες της πολυπλοκότητας και των συμβολισμών, όποτε θα εκφράζουμε την επίδοση των αλγορίθμων με τη βοήθειά τους.

1.7 Βιβλιογραφική Συζήτηση

Το αντικείμενο της Σχεδίασης και Ανάλυσης Αλγορίθμων (Design and Analysis of Algorithms) είναι εξαιρετικά πλούσιο, όπως μπορεί να φανεί από την βιβλιογραφία, όπου σκοπίμως παρατίθενται μερικά σημαντικά βιβλία που χρονολογούνται από τη δεκαετία του 1970 [1, 130] και τη δεκαετία του 1980 [2, 57, 123, 143, 170]. Για το ίδιο λόγο στη βιβλιογραφία συμπεριλαμβάνονται και μερικά παλαιά άρθρα, που θα αναφερθούν στην ώρα τους. Πιο πρόσφατα είναι τα βιβλία [27, 28, 70, 85]. Ιδιαιτέρως, στο βιβλίο των Dasgupta-Παπαδημητρίου-Vazirani [28] συμπεριλαμβάνεται και κεφάλαιο με τις τελευταίες εξελίξεις του κβαντικού υπολογισμού.

Το τρίτομο μνημειώδες έργο του Knuth πρωτοεκδόθηκε τη δεκαετία του 1970 αλλά επανεκδίδεται μέχρι τις ημέρες μας [76, 77, 80], καθώς είναι κλασικό βιβλίο αναφοράς. Το 1999 το Scientific American, υψηλού επιπέδου περιοδικό ευρείας κυκλοφορίας, κατέταξε το έργο αυτό μεταξύ των 12 καλύτερων επιστημονικών μονογραφιών του 20ου αιώνα, μαζί με τα έργα των Dirac για την κβαντομηχανική, του Einstein για τη σχετικότητα, του Mandelbrot για τα fractals, του Pauling για τους χημικούς δεσμούς, των Russell-Whitehead για τις θεμελιώσεις των μαθηματικών, των von Neumann-Morgenstern για τη θεωρία παιγνίων, του Wiener για την κυβερνητική, των Woodward-Hoffmann για τις τροχιακές συμμετρίες, του Feynman για την κβαντο-ηλεκτροδυναμική, του Smith για την αναζήτηση δομής, και τη συλλογή των άρθρων του Einstein.

Σημαντικό επίσης είναι και το τρίτομο έργο του Mehlhorn [108, 109, 110], ευρέος φάσματος όπως το τρίτομο βιβλίο του Knuth, χωρίς να περιέχονται αλγόριθμοι Αριθμητικής Ανάλυσης (Numerical Analysis), συμπεριλαμβάνοντας όμως αλγορίθμους Γραφικών (Graphics) και Υπολογιστικής Γεωμετρίας (Computational Geometry).

Εκλαϊκευτικά άρθρα έχουν δημοσιευθεί στο Scientific American από τους Lewis-Παπαδημητρίου [87] και τον Knuth [79]. Εισαγωγικά παραδείγματα ανάλυσης αλγορίθμων μεταξύ άλλων αναφέρονται στα άρθρα [41, 49, 159]. Τέλος, η

Εξίσωση 1.1 προέρχεται από την προμετωπίδα των βιβλίων του Wirth [173, 174].

Τα μοντέλα μηχανής ακολουθούν την προσέγγιση του βιβλίου του Mehlhorn [110]. Το μοντέλο της Μηχανής Δεικτών προτάθηκε με ισοδύναμους ορισμούς από σειρά ερευνητών αλλά η τελική του μορφή δόθηκε από τον Tarjan [157]. Ο Schönhage έδειξε ότι η Μηχανή Δεικτών μπορεί να προσομοιώσει τη λειτουργία της μηχανής Turing [137] .

Ο "πολλαπλασιασμός αλά ρωσικά" αναφέρεται στη βιβλιογραφία κατά κόρον με τη συγκεκριμένη ονομασία, αν και επίσης αναφέρεται ήταν γνωστός στην αρχαία Αίγυπτο αλλά και στον al Khowarizmi. Στις αριθμητικές πράξεις των αρχαίων Αιγυπτίων αναφέρεται το άρθρο [46]. Το υλικό για το παλίνδρομο προέρχεται από το άρθρο [41]. Η Άσκηση 2 αναφέρεται στο βιβλίο του Bentley [7], οι Ασκήσεις 13-14 βασίζονται στο άρθρο [49], ενώ η Άσκηση 15 αναφέρεται στο βιβλίο του Gries [61].

1.8 Ασκήσεις

1. Δίνεται ένα σύνολο με n θετικούς πραγματικούς αριθμούς. Θα εκτελέσουμε το εξής πείραμα. Επιλέγουμε από το σύνολο τυχαία 2 αριθμούς και αντικαθιστούμε τον καθένα με το μέσο όρο τους. Η ερώτηση είναι αν αυτό το πείραμα θα τελειώσει ποτέ, δηλαδή το σύνολο θα αποτελείται από n ίσους αριθμούς; Δηλαδή, μπορούμε να σχεδιάσουμε έναν αλγόριθμο με περατότητα ή πρόκειται απλώς για μία υπολογιστική διαδικασία;

2. Δίνεται ένα δοχείο με κόκκινα και πράσινα μήλα, καθώς επίσης ένας σωρός με έξτρα πράσινα μήλα. Θα εκτελέσουμε το εξής πείραμα. Επιλέγουμε 2 μήλα από το δοχείο. Αν είναι ομοιόχρωμα, τότε τα απομακρύνουμε και εισάγουμε στο δοχείο 2 πράσινα μήλα από τα έξτρα. Αν δεν είναι ομοιόχρωμα, τότε απομακρύνουμε το πράσινο, ενώ το κόκκινο επιστρέφεται στο δοχείο. Η ερώτηση είναι αν αυτό το πείραμα θα τελειώσει ποτέ, δηλαδή θα βρεθεί ποτέ το δοχείο με ένα μόνο μήλο; Δηλαδή, μπορούμε να σχεδιάσουμε έναν αλγόριθμο με περατότητα ή πρόκειται απλώς για μία υπολογιστική διαδικασία; Μπορούμε να βρούμε το χρώμα του τελευταίου μήλου με βάση το αρχικό πλήθος των κόκκινων και των πράσινων μήλων;

3. Να σχεδιασθεί και να αναλυθεί ένας αλγόριθμος διαίρεσης ακεραίων εφαρμόζοντας την τεχνική του πολλαπλασιασμού αλά ρωσικά με την αντίστροφη λογική.

4. Δεδομένου ενός ακεραίου n να βρεθεί ο μεγαλύτερος ακέραιος που είναι μικρότερος του n και δύναμη του 2. Να σχεδιασθεί ένας κακός και ένας καλός αλγόριθμος για το σκοπό αυτό. Οι αλγόριθμοι να αναλυθούν.

5. Για το επόμενο τμήμα ψευδοκώδικα, να γίνει επακριβής ανάλυση και να υπολογισθεί ο συμβολισμός O.

```
sum <-- 0;
for i <-- 1 to n
   for j <-- 1 to i*i
      for k <-- 1 to j
         sum <-- sum+1;
```

6. Για το επόμενο τμήμα ψευδοκώδικα, να γίνει επακριβής ανάλυση και να υπολογισθεί να υπολογισθεί ο συμβολισμός O. Τι παρατηρείτε σε σχέση με την προηγούμενη άσκηση;

```
sum <-- 0;
for i <-- 1 to n
   for j <-- 1 to i*i
      if (j mod i =0) then
         for k <-- 1 to j
            sum <-- sum+1;
```

7. Δίνεται ένας πίνακας `A[0..n-1]` με n διακριτούς ακεραίους. Ο επόμενος ψευδοκώδικας υπολογίζει τον αριθμό των αντιστροφών (inversions), δηλαδή των περιπτώσεων όπου `A[i]<A[j]` αλλά `i>j`. Να υπολογισθεί ο συμβολισμός O. Μπορεί να υπάρξει βελτίωση στον ψευδοκώδικα; Να υπολογισθεί εκ νέου ο συμβολισμός O.

```
procedure InvCount;
counter <-- 0;
for i <-- 1 to n do
   for j <-- 1 to n do
      if (i<j) and (A[i]>A[j]) then counter <-- counter+1;
```

8. Η **ταξινόμηση της φυσαλίδας** (bubblesort) ή ταξινόμηση με ανταλλαγές (exchange sort) είναι γνωστή από το αντικείμενο των Δομών Δεδομένων. Η διαδικασία `bubblesort` ταξινομεί επιτοπίως (in-place) τον πίνακα `A:[1..n]`. Ποιά εντολή είναι το βαρόμετρο; Να γίνει κοστολόγηση του αλγορίθμου και να βρεθεί η πολυπλοκότητα στην καλύτερη, μέση και χειρότερη περίπτωση.

```
procedure bubblesort
```

```
for i <-- 0 to n-2 do
   for j <-- 0 to n-2 do
      if A[j+1]<A[j] then
         temp <-- A[j];
         A[j] <-- A[j+1];
         A[j+1] <-- temp;
```

9. Δίνεται η επόμενη παραλλαγή της ταξινόμησης της φυσαλίδας. Η παραλλαγή αυτή να κοστολογηθεί ομοίως και να συγκριθεί με την προηγούμενη μέθοδο.

```
procedure bubblesort
for i <-- 0 to n-2 do
   for j <-- 0 to n-2-i do
      if A[j+1]<A[j] then
         temp <-- A[j];
         A[j] <-- A[j+1];
         A[j+1] <-- temp;
```

10. Δίνεται πίνακας `A[0..n-1,0..n-1]` και ζητείται να βρεθεί η πρώτη θέση ενός στοιχείου `x` (αν αυτό υπάρχει), όπου με την έκφραση "πρώτη" εννοείται ότι δεν υπάρχει άλλο `x` σε προηγούμενη γραμμή ή προηγούμενη στήλη. Να σχεδιασθεί αλγόριθμος και να αναλυθεί η καλύτερη, η μέση και η χειρότερη περίπτωση.

11. Ο αλγόριθμος `findmax` δέχεται τον πίνακα `A:[1..n]` και εξάγει το ζεύγος `max,i`, όπου `max` είναι το μέγιστο στοιχείο του πίνακα, ενώ `i` είναι η θέση όπου αυτό εμφανίζεται (όπου, $1 \leq i \leq n$). Ο αλγόριθμος να κοστολογηθεί και να βρεθεί η πολυπλοκότητα σε κάθε περίπτωση.

```
function findmax(A,n)
i <-- 1; j <-- 2; m <-- A[1];
while j <= n do
   if A[j]>max then
      max <-- A[j];
      i <-- j;
   j <-- j+1;
return(m,i);
```

12. Το **κόσκινο του Ερατοσθένη** (Eratosthenes sieve) είναι μία μέθοδος για την εύρεση των πρώτων αριθμών (δηλαδή, των αριθμών που διαιρούνται

μόνο από το 1 και τον εαυτό τους) που είναι μικρότεροι του n. Ο αλγόριθμος θεωρεί έναν πίνακα με τους ακεραίους από το 2 μέχρι το n, βρίσκει το μικρότερο ακέραιο i, τον τυπώνει και διαγράφει από τον πίνακα τους ακεραίους $i, 2i, 3i, \ldots$. Η διαδικασία σταματά όταν $i > \sqrt{n}$. Να σχεδιασθεί και να αναλυθεί ο αλγόριθμος.

13. Δίνονται δύο ακέραιοι $y > x > 0$ και ζητείται να δοθούν σε φθίνουσα τάξη όλα τα πολλαπλάσια του x, που είναι μικρότερα του y. Για παράδειγμα, αν $x = 100$ και $y = 550$, τότε η έξοδος είναι: 500, 400, 300, 200, 100. Να σχεδιασθεί και να αναλυθεί αλγόριθμος επίλυσης του προβλήματος.

14. Το έργο n υπαλλήλων είναι να αναβοσβήνουν m λάμπες που βρίσκονται σε έναν ευθύγραμμο δρόμο. Ο πρώτος υπάλληλος ανάβει όλες τις λάμπες. Ο δεύτερος υπάλληλος σβήνει τη δεύτερη, τέταρτη, έκτη κλπ. Ο τρίτος επεμβαίνει στην τρίτη, έκτη, ένατη κλπ και την ανάβει (σβήνει) αν είναι κλειστή (ανοικτή). Ο τέταρτος επεμβαίνει στην τέταρτη, όγδοη, δωδέκατη και αναβοσβήνει αντίστοιχα. Το ζητούμενο είναι να διαπιστωθεί ποίες λάμπες είναι ανοικτές όταν περάσει ο τελευταίος υπάλληλος. Για την επίλυση του προβλήματος να σχεδιασθεί και να αναλυθεί αλγόριθμος.

15. Λέγεται ότι ένας δισδιάστατος πίνακας `A[0..n-1,0..n-1]` έχει ένα **σημείο σέλλας** (saddle point), αν υπάρχει ένα στοιχείο `A[i,j]` που είναι το μικρότερο στοιχείο της γραμμής `i` και το μεγαλύτερο στοιχείο της στήλης `j`. Να σχεδιασθεί και να αναλυθεί αλγόριθμος εντοπισμού του σημείου αυτού (αν υπάρχει).

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

Σχήμα 1.2: Το πρόβλημα του μαγικού τετραγώνου.

16. **Μαγικό τετράγωνο** (magic square) λέγεται ένας πίνακας $n \times n$ που περιέχει τους ακέραιους από 1 ως n^2 σε τέτοιες θέσεις, ώστε το άθροισμα των τιμών κάθε γραμμής, κάθε στήλης και κάθε κύριας διαγωνίου είναι το ίδιο. Αν το n είναι περιττός αριθμός, τότε η τοποθέτηση των στοιχείων στο τετράγωνο γίνεται ως εξής. Αρχικά τοποθετείται το 1 στην κορυφή της

μεσαίας στήλης. Οι ακέραιοι $2, 3, \ldots$ τοποθετούνται σε θέσεις διαγωνίως επάνω και δεξιά. Αν ξεπερασθεί η πρώτη γραμμή, τότε η διαδικασία συνεχίζει στην τελευταία γραμμή, ενώ αν ξεπερασθεί η τελευταία στήλη, τότε η διαδικασία συνεχίζει στην πρώτη στήλη. Αν η επισκεπτόμενη θέση είναι κατειλημμένη τότε η διαδικασία συνεχίζει μία θέση προς τα κάτω στην ίδια στήλη. Στο Σχήμα 1.2 παρουσιάζεται ένα μαγικό τετράγωνο 5×5, όπου το άθροισμα των στοιχείων κάθε γραμμής, στήλης ή κύριας διαγωνίου είναι 65. Να σχεδιασθεί ένας αλγόριθμος που να διαπιστώνει αν ένας δισδιάστατος πίνακας είναι πράγματι μαγικό τετράγωνο. Επίσης, να σχεδιασθεί ένας αλγόριθμος κατασκευής ενός μαγικού τετράγωνου περιττής τάξης. Σε κάθε περίπτωση να αναλυθούν οι αντίστοιχοι αλγόριθμοι.

2

Θεωρητικό Υπόβαθρο

Περιεχόμενα Κεφαλαίου

Στο κεφάλαιο αυτό θα εξετασθούν τρία αντικείμενα. Πρώτον, θα παρουσιασθούν επί τροχάδην μερικά βασικά μαθηματικά εργαλεία που είναι απαραίτητα κατά την ανάλυση των αλγορίθμων. Πολλές από αυτά θα χρησιμοποιηθούν σε επόμενα κεφάλαια του βιβλίου. Δεύτερον, θα εισαχθούν με τυπικό τρόπο οι έννοιες της πολυπλοκότητας και των συμβολισμών O, Ω, Θ, ο και ω, που αναφέρθηκαν ακροθιγώς στο προηγούμενο κεφάλαιο. Τρίτον, θα γίνει μία πρώτη χρήση των μαθηματικών εργαλείων και των συμβολισμών στην πράξη για την μεθοδικότερη ανάλυση μερικών απλών αλγορίθμων.

2.1 Μαθηματικά Εργαλεία

Συνάρτηση Πάτωμα και Συνάρτηση Οροφή
Δεδομένου πραγματικού αριθμού x, το $\lfloor x \rfloor$ ισούται με το ακέραιο μέρος του x, ενώ το $\lceil x \rceil$ ισούται με τον μεγαλύτερο (ή ίσο) ακέραιο αριθμό του x. Για τις συναρτήσεις αυτές ισχύουν οι εξής ιδιότητες (όπου τα n, a, b ακέραιοι).

$$x - 1 \;\leq\; \lfloor x \rfloor \;\leq\; x \;\leq\; \lceil x \rceil \;\leq\; x + 1$$

$$\lfloor n/2 \rfloor + \lceil n/2 \rceil \;=\; n$$

$$\left\lceil \frac{\lceil n/a \rceil}{b} \right\rceil \;=\; \left\lceil \frac{n}{ab} \right\rceil \qquad\qquad \left\lfloor \frac{\lfloor n/a \rfloor}{b} \right\rfloor \;=\; \lfloor n/ab \rfloor$$

Εκθετικά και Δυνάμεις
Για τους πραγματικούς αριθμούς $a \neq 0, m, n$ ισχύουν οι εξής βασικές ιδιότητες:

$$a^0 = 1 \qquad a^1 = a \qquad a^{-1} = 1/a \qquad (a^m)^n = a^{mn} = (a^n)^m \qquad a^m a^n = a^{m+n}$$

Για πραγματικές σταθερές $a > 0, b$ ισχύει:

$$\lim_{n \to \infty} \frac{n^b}{a^n} \;=\; 0$$

Για κάθε πραγματικό αριθμό x ισχύει:

$$e^x \;=\; 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots \;=\; \sum_{i=0}^{\infty} \frac{x^i}{i!} \geq 1 + x$$

Για $|x| \leq 1$ ισχύει:

$$1 + x \leq e^x \;\leq\; 1 + x + x^2$$

Τέλος ισχύει:

$$\lim_{n \to \infty} \left(1 + \frac{x}{n}\right) \;=\; e^x$$

όπου $e \approx 2,71828$, η βάση των φυσικών λογαρίθμων.

Λογάριθμοι
Για κάποιο φυσικό ή πραγματικό αριθμό x, οι λογάριθμοι συμβολίζονται με $\log_b x$, όπου b είναι η βάση του λογάριθμου. Συνήθως, οι χρησιμοποιούμενοι λογάριθμοι αναφέρονται σε δυαδική βάση και θα δηλώνονται με $\lg x$, ενώ οι φυσικοί/νεπέριοι λογάριθμοι θα δηλώνονται με $\ln x$. Για τους πραγματικούς αριθμούς $a > 0, b > 0, c > 0, n$ ισχύουν οι εξής βασικές ιδιότητες:

$$a = b^{\log_b a} \qquad \log_c(ab) = \log_c a + \log_c b \qquad \log_b a^n = n \log_b a \qquad \log_b a = \frac{\log_c a}{\log_c b}$$

$$\log_b(1/a) = -\log_b a \qquad \log_b a = \frac{1}{\log_a b} \qquad a^{\log_b n} = n^{\log_b a}$$

Σε ότι αφορά στις εκφράσεις με τη βοήθεια των συμβολισμών O, Θ κλπ. δεν έχει σημασία αν ο λογάριθμος είναι δυαδικός, νεπέριος ή οποιοσδήποτε άλλος και για το λόγο αυτό μπορεί γενικώς να χρησιμοποιείται η έκφραση lg.

Σημαντικές σχέσεις είναι οι επόμενες. Για $|x| < 1$ ισχύει:

$$\ln(1+x) \;=\; x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} - \dots$$

ενώ για $x > -1$ ισχύει:

$$\frac{x}{1+x} \;\leq\; \ln(1+x) \;\leq\; x$$

Παραγοντικά
Είναι γνωστό ότι

$$n! \;=\; 1 \times 2 \times 3 \times \cdots \times n \;=\; \prod_{i=1}^{n} i$$

Πολύ συχνά χρησιμοποιούμενος είναι ο τύπος του Stirling:

$$n! \;=\; \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \frac{1}{12n} + \frac{1}{288n^2} + \dots\right)$$

Αν απαλείψουμε την παρένθεση, τότε οδηγούμαστε σε μία πολύ καλή προσέγγιση. Για παράδειγμα, εφαρμόζοντας τον τύπο του Stirling καταλήγουμε ότι $1! \approx 0,92$ (λάθος 8%), $2! \approx 1,92$ (λάθος 4%), $5! \approx 118,02$ (λάθος 2%), ενώ για το $100!$ το λάθος είναι 0.08%. Με άλλα λόγια, το λάθος είναι μία φθίνουσα συνάρτηση για αυξανόμενα n. Επίσης, από τον τύπο του Stirling και με απλή άλγεβρα προκύπτει ότι:

$$n! \;\leq\; n^n$$

Αριθμοί Fibonacci
Η ακολουθία αριθμών Fibonacci δεύτερης τάξης ορίζονται ως εξής:

$$F_i \;=\; F_{i-1} + F_{i-2}$$

ενώ για τις αρχικές συνθήκες ισχύει $F_0 = 0$ και $F_1 = 1$. Άρα με βάση τον ορισμό προκύπτει ότι η σειρά των αριθμών Fibonacci είναι:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$$

Δεδομένης της **χρυσής τομής** (golden ratio), ϕ, και της συζυγούς τιμής, $\hat{\phi}$, που ισούνται αντίστοιχα με:

$$\phi \;=\; \frac{1+\sqrt{5}}{2} \;\approx\; 1.61803 \qquad\qquad \hat{\phi} \;=\; \frac{1-\sqrt{5}}{2} \;\approx\; -0.61803$$

μπορεί να αποδειχθεί επαγωγικά η ταυτότητα De Moivre:

$$F_i \;=\; \frac{\phi^i - \hat{\phi}^i}{\sqrt{5}}$$

Καθώς $|\hat{\phi}| < 1$ συνεπάγεται ότι $|\hat{\phi}|/\sqrt{5} < 1/2$. Άρα, ο i-οστός αριθμός Fibonacci ισούται με $\phi^i/\sqrt{5}$ στρογγυλεμένο στον αμέσως μεγαλύτερο ακέραιο.

Αθροίσματα
Μερικές από επόμενες σχέσεις είναι ήδη γνωστές αλλά τις επαναλαμβάνουμε για λόγους πληρότητας.

$$\sum_{i=1}^{n} i \;=\; 1 + 2 + 3 + \ldots + n \;=\; \frac{1}{2}n(n+1)$$

$$\sum_{i=1}^{n} i^2 \;=\; 1 + 2^2 + 3^2 + \ldots + n^2 \;=\; \frac{1}{6}n(n+1)(2n+1)$$

$$\sum_{i=0}^{n} x^i \;=\; 1 + x + x^2 + \ldots + x^n \;=\; \frac{x^{n+1}-1}{x-1}$$

Αν $|x| < 1$ τότε ισχύει:

$$\sum_{i=0}^{\infty} x^i \;=\; \frac{1}{1-x}$$

Παραγωγίζοντας τα δύο σκέλη της σχέσης αυτής και πολλαπλασιάζοντας επί x προκύπτει:

$$\sum_{i=0}^{\infty} i\ x^i \ = \ \frac{x}{(1-x)^2}$$

Για τον αρμονικό αριθμό H_n ισχύει:

$$H_n \ = \ \sum_{i=1}^{n} \frac{1}{i} \ = \ 1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n} \ = \ \ln n+\gamma+\frac{1}{2n}-\frac{1}{12n^2}+\frac{1}{120n^4}-\ldots$$

όπου $\gamma = 0.577$ είναι η σταθερά του Euler.

Επίσης, για κάθε ακολουθία $a_1, a_2, \ldots, a_n$ ισχύουν οι σχέσεις (τηλεσκοπικά αθροίσματα):

$$\sum_{i=1}^{n}(a_i - a_{i-1}) \ = \ a_n - a_0 \qquad\qquad \sum_{i=0}^{n-1}(a_i - a_{i+1}) \ = \ a_0 - a_n$$

Και μία τελευταία χρήσιμη ιδιότητα με γινόμενα:

$$\lg\left(\prod_{i=1}^{n} a_i\right) \ = \ \sum_{i=1}^{n} \lg a_i$$

Διατάξεις, Συνδυασμοί και Δυωνυμικοί Συντελεστές

Τα n στοιχεία ενός συνόλου μπορούν να παράξουν $n!$ διαφορετικές διατάξεις. Το πλήθος των δυνατών διατάξεων επιλέγοντας k από n είναι:

$$n \times (n-1) \times \ldots \times (n-(k-1)) \ = \ \frac{n!}{(n-k)!}$$

Συνδυασμός των k αντικειμένων από n αντικείμενα είναι το πλήθος των τρόπων που μπορούν να επιλεχθούν k διακριτά αντικείμενα από n αντικείμενα και ισούται με:

$$\binom{n}{k}$$

Ο ανωτέρω τύπος προϋποθέτει ότι κάθε ένα από τα n αντικείμενα μπορεί να επιλεγεί μία μόνο φορά. Ο τρόπος αυτός λέγεται **επιλογή χωρίς αντικατάσταση** (selection without replacement). Αντιθέτως, κατά την **επιλογή με αντικατάσταση** (selection with replacement) επιτρέπεται κάποιο αντικείμενο να επιλεγεί και πάλι

χωρίς κάποιον περιορισμό. Για την περίπτωση αυτή, το πλήθος των επιλογών είναι:

$$\binom{n+k-1}{n-1}$$

Βασικές ταυτότητες των συνδυασμών (οι οποίες αποδεικνύονται εύκολα) είναι οι εξής:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \binom{n}{n-k}$$

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

Επίσης ισχύει:

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k}$$

Αν $x = y = 1$, τότε προκύπτει ότι

$$2^n = \sum_{k=0}^{n} \binom{n}{k}$$

Συχνά απαιτούνται επάνω και κάτω όρια. Έτσι έχουμε:

$$\binom{n}{k} = \frac{n(n-1)\dots(n-k+1)}{k(k-1)\dots 1} = \frac{n}{k}\,\frac{n-1}{k-1}\dots\frac{n-k+1}{1} \geq \left(\frac{n}{k}\right)^k \quad (2.1)$$

$$\binom{n}{k} = \frac{n(n-1)\dots(n-k+1)}{k(k-1)\dots 1} \leq \frac{n^k}{k!} \leq \left(\frac{en}{k}\right)^k$$

Τέλος, ο καταλανικός αριθμός ισούται με:

$$C_n = \frac{1}{n+1}\binom{2n}{k} = \frac{4^n}{\sqrt{\pi n^3}}\left(1 - \frac{9}{8n} + \frac{145}{128n^2} - \dots\right)$$

Πιθανότητες

Από το μάθημα των Πιαθανοτήτων/Στατιστικής είναι γνωστές οι επόμενες έννοιες αλλά επίσης τις επαναλαμβάνουμε για λόγους πληρότητας. Ας υποθέσουμε ότι μία διακριτή τυχαία μεταβλητή X λαμβάνει αριθμητικές τιμές: $X_1, X_2, X_3, \dots$. Η πιθανότητα να προκύψει η τιμή X_i συμβολίζεται με $P(X_i)$ ή με $P(X = X_i)$ και ισχύουν οι σχέσεις:

$$0 \leq P(X_i) \leq 1 \qquad \sum_{i=1}^{\infty} P(X_i) = 1$$

Η μέση (ή προσδοκητή) τιμή μίας διακριτής τυχαίας μεταβλητής X ισούται με:

$$\mu \;=\; \mathrm{E}(X) \;=\; \sum_{i=1}^{\infty} X_i \; P(X_i) \;=\; \sum_{i=1}^{\infty} P(X \geq X_i)$$

Η απόκλιση μίας τυχαίας μεταβλητής X δίνεται από τον τύπο:

$$\sigma^2 \;=\; \mathrm{Var}[X] \;=\; \mathrm{E}[(X - \mathrm{E}[X])^2] \;=\; \ldots \;=\; \mathrm{E}[X^2] - \mathrm{E}^2[X] \Rightarrow$$

$$\mathrm{E}[X^2] \;=\; \mathrm{Var}[X] - \mathrm{E}^2[X]$$

Η μέση τιμή του αθροίσματος δύο διακριτών τυχαίων μεταβλητών ισούται με:

$$\mathrm{E}(X + Y) \;=\; \mathrm{E}(X) + \mathrm{E}(Y)$$

Για δύο γεγονότα ανεξάρτητα μεταξύ τους ισχύει:

$$\mathrm{E}(X + Y) \;=\; \mathrm{E}(X) \;\; \mathrm{E}(Y)$$

Η πιθανότητα υπό συνθήκη να συμβεί ένα γεγονός A δεδομένου ενός γεγονότος B είναι:

$$p(A|B) \;=\; \frac{p(A \cap B)}{p(B)}$$

με την προϋπόθεση ότι $p(B) \neq 0$. Έτσι προκύπτει το Θεώρημα του Bayes:

$$p(A|B) \;=\; \frac{p(A)\; p(B|A)}{p(B)}$$

2.2 Συμβολισμοί Πολυπλοκότητας

Κατ'αρχήν παραθέτουμε τους ορισμούς τριών συμβολισμών πολυπλοκότητας (O, Ω και Θ), ενώ στη συνέχεια θα παραθέσουμε άλλους δύο συμβολισμούς πολυπλοκότητας (ο και ω).

Συμβολισμός O.
Μία συνάρτηση $f(n)$ λέγεται ότι έχει πολυπλοκότητα της τάξης $\mathrm{O}(g(n))$, και συμβολίζεται με $f(n) = \mathrm{O}(g(n))$ ή με $f(n) \in \mathrm{O}(g(n))$, αν υπάρχει μία θετική σταθερά c και μία τιμή n_0 έτσι ώστε για κάθε $n > n_0$ να ισχύει η σχέση $f(n) < cg(n)$. □

Συμβολισμός Ω.
Μία συνάρτηση $f(n)$ λέγεται ότι έχει πολυπλοκότητα της τάξης $\Omega(g(n))$, και συμβολίζεται με $f(n) = \Omega(g(n))$ ή με $f(n) \in \Omega(g(n))$, αν υπάρχει μία θετική σταθερά c και μία τιμή n_0 έτσι ώστε για κάθε $n > n_0$ να ισχύει η σχέση $f(n) > cg(n)$. □

Συμβολισμός Θ.
Μία συνάρτηση $f(n)$ λέγεται ότι έχει πολυπλοκότητα της τάξης $\Theta(g(n))$, και συμβολίζεται με $f(n) = \Theta(g(n))$ ή με $f(n) \in \Theta(g(n))$, αν υπάρχουν δύο θετικές σταθερές c_1, c_2 και μία τιμή n_0 έτσι ώστε κάθε για $n > n_0$ να ισχύει η σχέση $c_2 g(n) < f(n) < c_1 g(n)$. □

Ας εξετάσουμε ένα απλό παράδειγμα για να καταλάβουμε τη χρήση του συμβολισμού Θ. Έστω, λοιπόν, ότι πρέπει να αποδείξουμε τη σχέση:

$$\frac{n^2}{2} - \frac{n}{2} \;=\; \Theta(n^2)$$

Ικανή και αναγκαία συνθήκη για να ισχύει η ανωτέρω σχέση, είναι να ισχύει η επόμενη:

$$c_1\, n^2 \;\leq\; \frac{n^2}{2} - \frac{n}{2} \;\leq\; c_2\, n^2 \Rightarrow$$

$$c_1 \;\leq\; \frac{1}{2} - \frac{1}{2n} \;\leq\; c_2$$

όπου υπενθυμίζουμε ότι τα c_1, c_2 πρέπει να είναι θετικοί πραγματικοί αριθμοί, ενώ το n πρέπει να είναι θετικός ακέραιος. Εύκολα βλέπουμε ότι για $c_2 = 1/2$ και για $n \geq 1$ ισχύει το δεξιό σκέλος. Για το αριστερό σκέλος αρκεί να ισχύει $c_1 = 1/4$ και $n \geq 2$. Συνεπώς, για $c_1 = 1/4, c_2 = 1/2$ και $n \geq 2$ ισχύει η ανωτέρω σχέση. Αποδείξεις που αφορούν στους άλλους δύο συμβολισμούς (O και Ω) μπορούν να διεκπεραιωθούν με αντίστοιχο τρόπο.

Με απλά λόγια, χρησιμοποιούμε το συμβολισμό O και το συμβολισμό Ω για να δηλώσουμε ότι η επίδοση ενός αλγορίθμου είναι ασυμπτωτικά φραγμένη από επάνω και από κάτω, αντίστοιχα. Με το συμβολισμό Θ δηλώνουμε ότι η επίδοση ενός αλγορίθμου είναι ασυμπτωτικά φραγμένη από επάνω και από κάτω ταυτόχρονα. Οι συμβολισμοί O, Ω και Θ μπορεί να είναι περισσότερο ή λιγότερο **περιοριστικοί** ή σφικτοί (tight). Για παράδειγμα, είναι ευνόητο ότι ισχύει τόσο $2n^2 = \mathrm{O}(n^2)$ όσο και $2n = \mathrm{O}(n^2)$, όπου όμως η δεύτερη έκφραση είναι λιγότερη περιοριστική. Χρειαζόμαστε, λοιπόν, περισσότερο σφικτούς συμβολισμούς.

Συμβολισμός ο.
Μία συνάρτηση $f(n)$ λέγεται ότι έχει πολυπλοκότητα της τάξης $o(g(n))$, και συμβολίζεται με $f(n) = o(g(n))$ ή με $f(n) \in o(g(n))$, αν για κάθε θετική σταθερά $c > 0$ υπάρχει μία τιμή n_0 έτσι ώστε για κάθε $n > n_0$ να ισχύει η σχέση $f(n) < cg(n)$. □

Συμβολισμός ο. (εναλλακτικά)
Μία συνάρτηση $f(n)$ λέγεται ότι έχει πολυπλοκότητα της τάξης $o(g(n))$, και συμβολίζεται με $f(n) = o(g(n))$, αν ισχύει: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$. □

Συμβολισμός ω.
Μία συνάρτηση $f(n)$ λέγεται ότι έχει πολυπλοκότητα της τάξης $\omega(g(n))$, και συμβολίζεται με $f(n) = \omega(g(n))$ ή με $f(n) \in \omega(g(n))$, αν για κάθε θετική σταθερά $c > 0$ υπάρχει μία τιμή n_0 έτσι ώστε για κάθε $n > n_0$ να ισχύει η σχέση $f(n) > cg(n)$. □

Συμβολισμός ω. (εναλλακτικά)
Μία συνάρτηση $f(n)$ λέγεται ότι έχει πολυπλοκότητα της τάξης $\omega(g(n))$, και συμβολίζεται με $f(n) = \omega(g(n))$ αν ισχύει: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$. □

Κοινός τόπος σε όλους τους συμβολισμούς, λοιπόν, είναι η λέξη "ασυμπτωτικά". Η έννοια αυτή είναι δάνειο από τα καθαρά μαθηματικά (θεωρία αριθμών) και εμπεδώθηκε στην Πληροφορική από τον Knuth. Για τους συμβολισμούς αυτούς ισχύουν πολλές από τις ιδιότητες των πραγματικών αριθμών. □

Μεταβατική ιδιότητα. (transitivity)
$f(n)=O(g(n))$ και $g(n)=O(h(n)) \Rightarrow f(n)=O(h(n))$
$f(n)=\Omega(g(n))$ και $g(n)=\Omega(h(n)) \Rightarrow f(n)=\Omega(h(n))$
$f(n)=\Theta(g(n))$ και $g(n)=\Theta(h(n)) \Rightarrow f(n)=\Theta(h(n))$
$f(n)=o(g(n))$ και $g(n)=o(h(n)) \Rightarrow f(n)=o(h(n))$
$f(n)=\omega(g(n))$ και $g(n)=\omega(h(n)) \Rightarrow f(n)=\omega(h(n))$

Ανακλαστική ιδιότητα. (reflexivity)
$f(n) = O(f(n))$
$f(n) = \Omega(f(n))$
$f(n) = \Theta(f(n))$

Συμμετρική ιδιότητα. (symmetry)

$f(n) = \Theta(g(n))$ αν και μόνο αν $g(n) = \Theta(f(n))$

Ανάστροφη Συμμετρική ιδιότητα. (transpose symmetry)
$f(n) = \mathrm{O}(g(n))$ αν και μόνο αν $g(n) = \Omega(f(n))$
$f(n) = \mathrm{o}(g(n))$ αν και μόνο αν $g(n) = \omega(f(n))$

Για μία καλύτερη κατανόηση των πέντε αυτών συμβολισμών, παρουσιάζουμε τον Πίνακα 2.1. Στο αριστερό σκέλος μέσω του αντίστοιχου συμβολισμού δίνεται η σχέση μεταξύ των συναρτήσεων $f(n)$ και $g(n)$, ενώ στο δεξιό σκέλος δίνεται η σχέση μεταξύ των πραγματικών αριθμών a και b. Διδακτική αξία έχει η κατανόηση της αναλογίας μεταξύ των πέντε συμβολισμών με τις πέντε σχέσεις διάταξης. Σημειώνεται, όμως, ότι αν και δύο πραγματικοί αριθμοί είναι πάντοτε συγκρίσιμοι, εντούτοις δεν συμβαίνει το ίδιο πάντοτε για δύο ασυμπτωτικές εκφράσεις. Για παράδειγμα, δεν είναι δυνατόν οι συναρτήσεις $f(n) = n$ και $g(n) = n^{1+\cos n}$ να συσχετισθούν με κάποιο συμβολισμό.

$f(n)=\mathrm{O}(g(n))$	$a \leq b$
$f(n)=\Omega(g(n))$	$a \geq b$
$f(n)=\Theta(g(n))$	$a = b$
$f(n)=\mathrm{o}(g(n))$	$a < b$
$f(n)=\omega(g(n))$	$a > b$

Πίνακας 2.1: Συσχέτιση ασυμπτωτικών συμβολισμών.

Οι συμβολισμοί o και θ, που χρησιμοποιούνται λιγότερα συχνά στη βιβλιογραφία, εκφράζουν επίσης την ασυμπτωτική συμπεριφορά αλγορίθμων. Ωστόσο, σε αντίθεση με τους συμβολισμούς O, Ω και Θ, οι συμβολισμοί o και ω χρησιμοποιούνται για να εκφράσουν λιγότερο περιοριστικές καταστάσεις ασυμπτωτικά. Έτσι ισχύει $2n = \mathrm{o}(n^2)$ αλλά $2n^2 \neq \mathrm{o}(n^2)$.

Ας εξετάσουμε ένα ακόμη απλό παράδειγμα σε σχέση με το συμβολισμό o. Έστω, λοιπόν, ότι για a θετικό ακέραιο πρέπει να αποδείξουμε τη σχέση:

$$n^a \in \mathrm{o}(2^n)$$

Λαμβάνουμε το όριο το λόγου των δύο συναρτήσεων και εφαρμόζουμε τον κανόνα του L'Hopital διαδοχικά:

$$\lim_{n\to\infty} \frac{n^a}{2^n} = \lim_{n\to\infty} \frac{n^a}{e^{n\ln 2}} = \lim_{n\to\infty} \frac{an^{a-1}}{\ln 2\ e^{n\ln 2}} = \ldots = \lim_{n\to\infty} \frac{a!}{(\ln 2)^a\ e^{n\ln 2}} = 0$$

Για να γίνουν συγκριτικά αντιληπτοί οι προηγούμενοι συμβολισμοί ας θεωρήσουμε τη σχέση $f(n) = 4n^3 + 3$. Στην περίπτωση αυτή ισχύει:
$f(n) = \Theta(n^3)$
$f(n) = \mathrm{O}(n^3) = \mathrm{O}(n^4) \ldots$
$f(n) = \Omega(n^3) = \Omega(n^2)) = \Omega(n) = \Omega(1)$
$f(n) = \mathrm{o}(n^4) = \mathrm{o}(n^5) \ldots$
$f(n) = \omega(n^2) = \omega(n) = \omega(1)$

Συνδέοντας με το υλικό του προηγουμένου καφαλαίου, λοιπόν, σκοπός μας είναι αρχικά η εύρεση του χρονικού κόστους ενός αλγορίθμου με τη βοήθεια μίας συνάρτησης $f(n)$, ενώ στη συνέχεια πρέπει να βρούμε κάποια συνάρτηση $g(n)$ με τον αντίστοιχο συμβολισμό. Αυτό που γίνεται συχνότερα στην πράξη, όπου προκύπτουν εκθετικές ή πολυωνυμικές συναρτήσεις, είναι να απομονώσουμε τον όρο με το μεγαλύτερο ειδικό βάρος αγνοώντας τους άλλους όρους καθώς και τους σταθερούς συντελεστές.

2.3 Χρήση Συμβολισμών στην Ανάλυση

Ας θυμηθούμε από τα Κεφάλαια 1.4-1.5 τις εκφράσεις που έχουν προκύψει σχετικά με την επίδοση των αλγορίθμων ταξινόμησης με επιλογή και με εισαγωγή. Πιο συγκεκριμένα, κατά την επακριβή ανάλυση για την ταξινόμηση με επιλογή είχαμε καταλήξει στις εκφράσεις:

$$\begin{aligned} T(n) &= (c_3/2 + c_4/2)n^2 + (c_1 + c_2 + c_3/2 - c_4/2 + c_5 + c_6 + c_7)n + c_1 \\ &= an^2 + bn + c \end{aligned}$$

Επίσης, είχαμε διαπιστώσει ότι η ταξινόμηση με επιλογή έχει την ίδια επίδοση για την καλύτερη, τη μέση και τη χειρότερη περίπτωση ανεξαρτήτως των δεδομένων εισόδου και συνεπώς ο αλγόριθμος αυτός θεωρείται **σταθερός** (robust). Για το συγκεκριμένο αλγόριθμο και τη συγκεκριμένη ανάλυση εύκολα συμπεραίνεται ότι $T(n) = g(\mathrm{O}(\mathrm{n}^2))$ αλλά και $T(n) = g(\Omega(\mathrm{n}^2))$, και επομένως τελικά προκύπτει $T(n) = g(\Theta(\mathrm{n}^2))$. Έτσι, τελικά καταλήγουμε σε μία τυπική έκφραση (δηλαδή, το $\Theta(n^2)$) για να εκφράσουμε με συμπυκνωμένο τρόπο την πολυπλοκότητα του αλγορίθμου.

Τώρα θα εξετάσουμε με περισσότερη λεπτομέρεια και πάλι την ταξινόμηση με εισαγωγή. Όπως έχουμε αναφέρει προηγουμένως, για μία ολόκληρη αλγορίθμων ταξινόμησης, ως βαρόμετρο επιλέγεται η πράξη της σύγκρισης. Στη συγκεκριμένη περίπτωση, ως βαρόμετρο επιλέγεται η σύγκριση `A[i]>key` στην εντολή 4 (αγνοώντας

τις εσωτερικές συγκρίσεις της εντολής του βρόχου `for` στην εντολή 1, καθώς και τη σύγκριση `i>0` στην εντολή 4). Εύκολα προκύπτει ότι στη χειρότερη περίπτωση, για συγκεκριμένο `i`, το `key` είναι μικρότερο από κάθε `A[i]`, όπου το `i` κυμαίνεται από 1 μέχρι `n-1`, οπότε διαδοχικά το `key` θα συγκριθεί με τα `A[i-1]`, `A[i-2]`, ..., `A[1]` πριν εξέλθουμε από το βρόχο επειδή ισχύει η συνθήκη `i=0`. Από αυτή τη βασική σκέψη συνεπάγεται ότι καθώς το `j` μεταβάλλεται από 2 μέχρι `n`, στη χειρότερη περίπτωση ο συνολικός αριθμός των συγκρίσεων είναι:

$$\sum_{j=2}^{n}(j-1) \;=\; n(n-1)/2 \;=\; \Theta(n^2)$$

Στη συνέχεια θα εξετάσουμε λεπτομερέστερα τη μέση περίπτωση. Ας υποθέσουμε ότι τα στοιχεία του πίνακα είναι διακριτά (δηλαδή, διαφορετικά μεταξύ τους) και ότι είναι ισοπίθανο να εμφανισθεί μία οποιαδήποτε διάταξη των n στοιχείων. Άρα, όταν θεωρούμε την τιμή `key=A[j]` (εντολή 2), που πρέπει να παρεμβληθεί μεταξύ στοιχείων `A[1]`, `A[2]`, ..., `A[j-1]`, δεχόμαστε ότι το `key` μπορεί με ίδια πιθανότητα να είναι το k-οστό μεγαλύτερο, για $1 \leq k \leq j$. Αν, λοιπόν, το `key` είναι το μεγαλύτερο, αυτό θα γίνει αντιληπτό με 1 σύγκριση με το στοιχείο `A[j-1]`, αν είναι το δεύτερο μεγαλύτερο, αυτό θα γίνει αντιληπτό με 2 συγκρίσεις, κοκ, αν είναι το $(j-1)$-οστό μεγαλύτερο, αυτό θα γίνει αντιληπτό με $j-1$ συγκρίσεις, ενώ τέλος, αν είναι το j-οστό μεγαλύτερο (δηλαδή το μικρότερο), αυτό θα γίνει αντιληπτό και πάλι με $j-1$ συγκρίσεις. Συνεπώς, η μέση τιμή των συγκρίσεων για μία δεδομένη τιμή του j είναι:

$$\begin{aligned} c_j &= \frac{1}{j}\left(1+2+3+\ldots+(j-1)+(j-1)\right) \\ &= \frac{1}{j}\left(\sum_{i=1}^{j} i-1\right) \;=\; \frac{j+1}{2}-\frac{1}{j} \end{aligned}$$

Θεωρώντας ότι το j μεταβάλλεται από 2 μέχρι n, έχουμε:

$$\sum_{j=2}^{n} c_j \;=\; \sum_{j=2}^{n}\left(\frac{j+1}{2}-\frac{1}{j}\right) \;=\; \frac{n^2+3n}{4}-H_n \;=\; \Theta(n^2)$$

Στο τελευταίο στάδιο της ανωτέρω σχέσης ισχύει ότι $H_n = \Theta(\lg n)$ και επομένως αγνοείται ως προς το $n^2/4$. Τελικά, λοιπόν, προκύπτει και τυπικά ότι τόσο στη χειρότερη περίπτωση, όσο και στη μέση περίπτωση η πολυπλοκότητα της ταξινόμησης με εισαγωγή είναι $\Theta(n^2)$. Ωστόσο, εύκολα διαπιστώνεται ότι στην καλύτερη περίπτωση ισχύει $\Theta(n)$.

2.4 Χειρισμός Αθροισμάτων

Επαγωγή
Τη μέθοδο αυτή γνωρίζουμε από το μάθημα των Διακριτών Μαθηματικών, αλλά εδώ απλώς θα αναπτύξουμε ένα παράδειγμα ως μία εναλλακτική απόδειξη σε σχέση με την ανάλυση των παλίνδρομων που αναφέραμε στο Κεφάλαιο 1.6. Με λίγα λόγια, δοθείσης μίας σχέσης προς απόδειξη, κατά την επαγωγή αποδεικνύουμε ότι η σχέση ισχύει για μικρά n, υποθέτουμε ότι ισχύει για τυχόν k και τέλος αποδεικνύουμε ότι ισχύει για $k+1$.

Για παράδειγμα, θα αποδείξουμε επαγωγικά ότι για άρτια n ισχύει:

$$\sum_{i=0}^{n/2-1} \frac{i}{2^{i+1}} \;=\; 1 - \frac{n/2+1}{2^{n/2}}$$

Εύκολα φαίνεται ότι για $n=2$, τότε αριστερό και δεξιό μέλος ισούνται με 0. Δεχόμαστε ότι ισχύει η σχέση για τυχόν άρτιο k:

$$\sum_{i=0}^{k/2-1} \frac{i}{2^{i+1}} \;=\; 1 - \frac{k/2+1}{2^{k/2}}$$

και θα αποδείξουμε ότι ισχύει για $k+2$:

$$\sum_{i=0}^{k/2} \frac{i}{2^{i+1}} \;=\; 1 - \frac{k/2+2}{2^{k/2+1}}$$

Λαμβάνουμε το αριστερό σκέλος και διαδοχικά έχουμε:

$$\begin{aligned}\sum_{i=0}^{k/2} \frac{i}{2^{i+1}} \;&=\; \sum_{i=0}^{k/2-1} \frac{i}{2^{i+1}} + \frac{k/2}{2^{k/2+1}} \;=\; 1 - \frac{k/2+1}{2^{k/2}} + \frac{k/2}{2^{k/2+1}} \\ &=\; 1 - \frac{k+2}{2^{k/2+1}} + \frac{k/2}{2^{k/2+1}} \;=\; 1 - \frac{k/2+2}{2^{k/2+1}}\end{aligned}$$

Περιορισμός όρων
Δεδομένου ενός αθροίσματος, αντικαθιστούμε κάθε όρο του αθροίσματος με μία μεγαλύτερη ποσότητα, που μπορούμε ευκολότερα να χειρισθούμε. Επί παραδείγματι, αν θεωρήσουμε ως γενικό πρότυπο τη σχέση $\sum_{i=1}^{n} a_i \;\leq\; n a_{max}$, τότε για το γνωστό μας άθροισμα μπορεί εναλλακτικά να προκύψει:

$$\sum_{i=1}^{n} i \;\leq\; \sum_{i=1}^{n} n \;=\; n^2 \;=\; \mathrm{O}(n^2)$$

Δεδομένου ενός αθροίσματος $\sum_{i=0}^{n} a_i$, ας υποθέσουμε ότι $a_{k+1}/a_k \leq r$, όπου $k \geq 0$, ενώ ισχύει για τη σταθερά $r < 1$. Έτσι, λοιπόν, ισχύει:

$$\sum_{i=0}^{n} a_i \leq \sum_{i=0}^{\infty} a_0 r^i = a_0 \sum_{i=0}^{\infty} r^i = a_o \frac{1}{1-r}$$

Θα προσεγγίσουμε το άθροισμα $\sum_{i=1}^{\infty}(i/3^i)$ με βάση τη μέθοδο αυτή και λαμβάνουμε το λόγο:

$$\frac{a_{k+1}}{a_k} = \frac{(i+1)/3^{i+1}}{i/3^i} = \frac{i+1}{3i} \leq \frac{2}{3}$$

Δηλαδή, για κάθε $k \geq 1$ ισχύει $r = 2/3$, οπότε:

$$\sum_{i=1}^{\infty}(i/3^i) \leq \sum_{i=1}^{\infty} \frac{1}{3} \left(\frac{2}{3}\right)^i = \frac{1}{3} \frac{1}{1-2/3} = 1$$

Διάσπαση αθροισμάτων

Δεδομένου ενός αθροίσματος, το επιμερίζουμε σε μικρότερα που μπορούν να επιλυθούν ευκολότερα. Επί παραδείγματι, για το γνωστό άθροισμα ισχύει:

$$\begin{aligned}\sum_{i=1}^{n} i &= \sum_{i=1}^{n/2} i + \sum_{i=n/2+1}^{n} i \\ &\geq \sum_{i=1}^{n/2} 0 + \sum_{i=n/2+1}^{n} (n/2) \geq (n/2)^2 = \Omega(n^2)\end{aligned}$$

Τώρα θα χρησιμοποιήσουμε την παρούσα τεχνική μαζί με την προηγούμενη τεχνική (περιορισμός όρων) για το άθροισμα $\sum_{i=0}^{\infty} i^2/2^i$. Λαμβάνοντας το λόγο δύο διαδοχικών όρων έχουμε:

$$\frac{a_{k+1}}{a_k} = \frac{(i+1)^2/2^{i+1}}{i^2/2^i} = \frac{(i+1)^2}{2i^2} \leq \frac{8}{9}$$

για κάθε $k \geq 3$. Επομένως, θα επιμερίσουμε αναλόγως το άθροισμα:

$$\begin{aligned}\sum_{i=0}^{\infty} i^2/2^i &= \sum_{i=0}^{2} i^2/2^i + \sum_{i=3}^{\infty} i^2/2^i \\ &= \mathrm{O}(1) + \frac{9}{8} \sum_{i=3}^{\infty} \left(\frac{8}{9}\right)^i = \mathrm{O}(1)\end{aligned}$$

καθώς το δεύτερο άθροισμα είναι φθίνουσα γεωμετρική πρόοδος.

Χρήση ολοκληρωμάτων
Ένα άθροισμα με μία αύξουσα συνάρτηση $f(k)$ μπορεί να περιορισθεί από ολοκληρώματα με βάση το γενικό τύπο:

$$\int_{m-1}^{n} f(x)dx \leq \sum_{m}^{n} f(k) \leq \int_{m}^{n+1} f(x)dx$$

θεωρώντας ότι σε μία γραφική παράσταση η καμπύλη της συνάρτησης $f(k)$ προσεγγίζεται από επάνω και κάτω από μεγαλύτερα και μικρότερα ορθογώνια. Αντίστοιχα, μία φθίνουσα συνάρτηση $f(k)$ μπορεί να περιορισθεί με βάση το γενικό τύπο:

$$\int_{m}^{n+1} f(x)dx \leq \sum_{m}^{n} f(k) \leq \int_{m-1}^{n} f(x)dx$$

Για τον αρμονικό αριθμό H_n ισχύει:

$$\sum_{i=1}^{n} \frac{1}{i} \;\geq\; \int_{1}^{n+1} \frac{dx}{x} \;=\; \ln(n+1)$$

και

$$\sum_{i=2}^{n} \frac{1}{i} \;\leq\; \int_{1}^{n} \frac{dx}{x} \;=\; \ln n \Rightarrow \sum_{i=1}^{n} \frac{1}{i} \;\leq\; \ln n + 1$$

Επομένως σε περιπτώσεις εύρεσης της πολυπλοκότητας αρκεί η χρήση της σχέσης $H_n = \Theta(\lg n)$.

2.5 Κατηγοριοποίηση Αλγορίθμων

Αφού λοιπόν ορίσθηκαν και τυπικά οι συμβολισμοί της πολυπλοκότητας, είναι δυνατόν να κατηγοριοποιήσουμε τους διαφόρους αλγορίθμους σε μία από τις επόμενες κατηγορίες:

O(1). Κάθε εντολή εκτελείται σε πεπερασμένο πλήθος φορές, οπότε λέγεται ότι ο αλγόριθμος είναι "σταθερής πολυπλοκότητας".

O($\log n$). Ο αλγόριθμος είναι "λογαριθμικής πολυπλοκότητας". Με "log" και με "ln" θα συμβολίζεται ο δυαδικός και ο φυσικός λογάριθμος, αντιστοίχως. Πρακτικά, οι λογάριθμοι που θα χρησιμοποιηθούν είναι κυρίως δυαδικοί.

O(n). Η πολυπλοκότητα λέγεται "γραμμική". Γενικώς, αυτή είναι η επίδοση ενός αλγορίθμου που πρέπει να εξετάσει ή να δώσει στην έξοδο n στοιχεία.

$O(n \log n)$. Διαβάζεται όπως ακριβώς γράφεται (δηλαδή, "n log n") χωρίς να χρησιμοποιείται κάποιο επίθετο (π.χ. γραμμολογαριθμική). Στην κατηγορία αυτή ανήκουν πολλοί αλγόριθμοι ταξινόμησης.

$O(n^2)$. Αναφέρεται ως "τετραγωνική πολυπλοκότητα".

$O(n^3)$. Αναφέρεται ως "κυβική πολυπλοκότητα". Οι αλγόριθμοι αυτοί πρέπει να χρησιμοποιούνται μόνο για προβλήματα μικρού μεγέθους.

$O(2^n)$. Σπάνια στην πράξη χρησιμοποιούνται αλγόριθμοι "εκθετικής πολυπλοκότητας", ενώ πάρα πολλά γνωστά προβλήματα έχουν αυτή την επίδοση.

Πολυπλοκότητα	$n = 20$	$n = 40$	$n = 60$
$O(n)$	0.00002 sec	0.00004 sec	0.00006 sec
$O(n^2)$	0.0004 sec	0.0016 sec	0.0036 sec
$O(n^3)$	0.008 sec	0.064 sec	0.216 sec
$O(2^n)$	1 sec	12.7 ημέρες	366 αιώνες
$O(n!)$	771 αιώνες	$3 \cdot 10^{32}$ αιώνες	$3 \cdot 10^{66}$ αιώνες

Πίνακας 2.2: Σύγκριση πολυπλοκότητας αλγορίθμων

Στον Πίνακα 2.5 υπολογίζεται ο χρόνος που απαιτείται από αλγορίθμους διαφόρων πολυωνυμικών και εκθετικών πολυπλοκοτήτων ως συνάρτηση του μεγέθους του προβλήματος. Για να γίνουν οι χρονικές εκτιμήσεις υποτίθεται ότι κάθε στοιχειώδης πράξη απαιτεί ένα msec στη CPU. Έτσι, αν για έναν αλγόριθμο τάξης $O(n^3)$ διπλασιασθεί το μέγεθος του προβλήματος, τότε απαιτείται οκταπλάσιος (2^3) χρόνος για να περατωθεί ο αλγόριθμος. Διαπιστώνεται αμέσως ότι οι αλγόριθμοι εκθετικής πολυπλοκότητας δεν είναι πρακτικής χρησιμότητας ακόμη και για μικρό αριθμό δεδομένων.

Επειδή η βελτίωση των αλγορίθμων είναι ζωτικής σημασίας, μία ιδιαίτερη περιοχή της Πληροφορικής, η Υπολογιστική Πολυπλοκότητα (Computational Complexity), διερευνά από θεωρητική και αναλυτική άποψη τους αλγορίθμους και τους ταξινομεί αναλόγως με την επίδοσή τους. Συνεχώς αναπτύσσονται νέοι καλύτεροι αλγόριθμοι, ενώ άλλοι περιθωριοποιούνται ως μη αποτελεσματικοί. Ένας αλγόριθμος λέγεται **βέλτιστος** (optimal) αν αποδειχθεί ότι δεν μπορεί να κατασκευασθεί καλύτερος.

Αιτιοκρατικοί (deterministic) λέγονται οι αλγόριθμοι των οποίων τα βήματα εκτέλεσης είναι σε κάθε χρονική στιγμή καθορισμένα και μοναδικά. Η έννοια του **μη αιτιοκρατικού** (ή πιθανοτικού-στοχαστικού) αλγορίθμου είναι μία θεωρητική

έννοια και παραπέμπει σε μηχανές που αποφασίζουν για τη σωστή μεταξύ πολλών επιλογή με βάση κάποιες πιθανότητες εξέλιξης των συμβάντων.

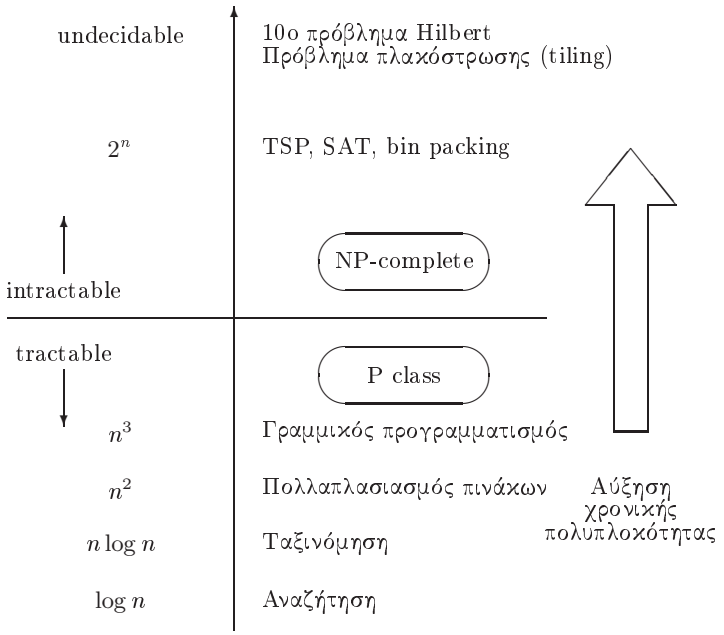
Πολυωνυμικοί (polynomial) λέγονται οι αιτιοκρατικοί αλγόριθμοι με πολυπλοκότητα που φράσσεται άνω από μία πολυωνυμική έκφραση. Για παράδειγμα, πολυωνυμικοί είναι οι αλγόριθμοι τάξης $O(n)$, $O(n^{3/2})$, $O(n^2)$ κ.τ.λ. Συνήθως δεν απαιτούν μεγάλη υπολογιστική προσπάθεια σε αντίθεση με τους αλγορίθμους πολυπλοκότητας τάξης $O(n!)$, $O(2^n)$ ή $O(n^n)$, που ονομάζονται μη πολυωνυμικοί.

Το σύνολο των προβλημάτων που μπορούν να επιλυθούν από αιτιοκρατικούς αλγορίθμους σε πολυωνυμικό χρόνο λέγεται σύνολο P, ενώ το σύνολο των προβλημάτων που μπορούν να επιλυθούν από μη αιτιοκρατικούς αλγορίθμους σε χρόνο πολυωνυμικό λέγεται σύνολο NP, που σημαίνει **μη αιτιοκρατικά πολυωνυμικά** (nondeterministic polynomial) προβλήματα. Προφανώς το σύνολο P είναι υποσύνολο του συνόλου NP, το αντίστροφο όμως φαίνεται ότι δεν ισχύει.

Μεταξύ των προβλημάτων του συνόλου NP διακρίνεται η σημαντική κατηγορία των NP-πλήρων (NP-complete). Ένα πρόβλημα αυτής της κατηγορίας θα επιλυθεί σε πολυωνυμικό χρόνο αν και μόνο αν όλα τα άλλα προβλήματα της ίδιας κλάσης επιλυθούν σε πολυωνυμικό χρόνο. Η σημασία της NP-πληρότητας επομένως είναι σπουδαία, διότι μόλις λυθεί ένα από τα NP-πλήρη προβλήματα με κάποιον πολυωνυμικό αλγόριθμο, τότε όλα τα προβλήματα που ανήκουν στο NP θα μπορούσαν να λυθούν με αντίστοιχους αλγορίθμους.

Το Σχήμα 2.5 απεικονίζει αυτό που αποκαλούμε "φάσμα της υπολογιστικής πολυπλοκότητας", δηλαδή ένα σχεδιάγραμμα των βασικών προβλημάτων μαζί με την πολυπλοκότητα των γρηγορότερων γνωστών αλγορίθμων που τα επιλύουν. Το σχήμα χωρίζεται σε δύο μεγάλες περιοχές προβλημάτων, τα δύσκολα ή δυσχείριστα (intractable) και τα εύκολα ή βατά (tractable). Στην κορυφή της πρώτης περιοχής ανήκουν τα προβλήματα για τα οποία δεν υπάρχουν σχετικοί αλγόριθμοι (undecidable). Πιο κάτω βρίσκονται τα προβλήματα για τα οποία υπάρχουν πρακτικά χρήσιμοι μη αλγόριθμοι, διότι απαιτούν εκθετικό χρόνο. Τα προβλήματα αυτά ανήκουν στην κατηγορία NP-πλήρη. Στη δεύτερη περιοχή βρίσκονται προβλήματα για τα οποία υπάρχουν γνωστοί, πρακτικά χρήσιμοι αλγόριθμοι. Είναι τα προβλήματα του είδους που εξετάζουμε σε αυτό το βιβλίο.

Ως προς το ανωτέρω σχήμα δίνονται οι επόμενες δύο επεξηγήσεις. Πρώτον, ο Hilbert ως 10ο πρόβλημα έθεσε το εξής: Δεδομένου ενός πολυωνύμου P με n μεταβλητές και ακεραίους συντελεστές, να βρεθεί ένας αλγόριθμος που να προσδιορίζει αν η εξίσωση $P = 0$ έχει ή όχι ακέραιες λύσεις. Η απάντηση σε αυτό το πρόβλημα είναι αρνητική, όπως έδειξε ο Matiyasevich το 1970. Δεύτερον, η διατύπωση του προβλήματος SAT (satisfiability) έχει ως εξής: Έστω μία λογική έκφραση που αποτελείται από n λογικές μεταβλητές και λογικοί τελεστές and,



Σχήμα 2.1: Φάσμα υπολογιστικής πολυπλοκότητας

or, not. Υπάρχει συνδυασμός τιμών (δηλαδή, αληθής/ψευδής) των μεταβλητών αυτών έτσι ώστε η τελική τιμή της έκφρασης να είναι αληθής; Ο συνδυασμός όλων των δυνατών εισόδων που πρέπει να δοκιμασθούν είναι 2^n, άρα το πρόβλημα ανήκει στην κλάση NP. Μάλιστα το πρόβλημα SAT κατέχει κεντρική θέση στη θεωρία της πολυπλοκότητας διότι τα περισσότερα NP-πλήρη προβλήματα προκύπτουν με πολυωνυμικές αναγωγές από αυτό. Επομένως, το πρόβλημα αυτό είναι και NP-πλήρες.

Η αδυναμία των ερευνητών να προτείνουν έναν πολυωνυμικό αλγόριθμο για πολλά δύσκολα προβλήματα επί του παρόντος δείχνει ότι δεν ισχύει η ισότητα P=NP, αλλά ισχύει P⊆NP. Η εικασία περί την ισότητα P=NP αποτελεί ένα από τα διασημότερα ανοικτά προβλήματα της σύγχρονης επιστήμης της πληροφορικής και δεν αναμένεται να επιλυθεί εύκολα με τη χρήση των σημερινών συμβατικών υπολογιστικών μοντέλων. Η έρευνα ωστόσο παρακάμπτει το σημείο αυτό προχωρώντας στην ανάπτυξη προσεγγιστικών αλγορίθμων για την επίλυση των δύσκολων προβλημάτων, έτσι ώστε να επιτυγχάνεται μία αποδεκτή λύση σε πολυωνυμικό χρόνο. Περισσότερα στοιχεία σχετικά με τη θεωρητική πλευρά της ανάλυσης των αλγορίθμων είναι πέρα από τους σκοπούς αυτού του βιβλίου.

2.6 Βιβλιογραφική Συζήτηση

Στο κεφάλαιο αυτό έγινε μία ταχεία παρουσίαση μερικών μαθηματικών εργαλείων που είναι χρήσιμα κατά την ανάλυση αλγορίθμων. Η παρουσίαση αυτή έγινε χωρίς αυστηρή προσέγγιση αλλά υπό τύπον υπενθύμισης. Το υλικό αυτό μπορεί να βρεθεί σε εισαγωγική μορφή σε βιβλία Διακριτών Μαθηματικών (Discrete Mathematics), όπως το βιβλίο [35] που εξετάζει εις πλάτος πολλά αντικείμενα. Τα διδακτικά εγχειρίδια με μία εις βάθος προσέγγιση και πλούσιο σχετικό υλικό υπογράφονται (και) από τον Knuth [59, 76]. Ιδιαιτέρως, το βιβλίο [59] διακρίνεται για τη βαθειά του μαθηματική προσέγγιση. Το υλικό του κεφαλαίου αυτό παρουσιάζεται σε παρόμοια μορφή σε πολλά αντίστοιχα εγχειρίδια, όπως των Brassard-Bratley [13] και των Cormen-Leiserson-Rivest-Stein [27]. Συμπληρωματικά στοιχεία μπορούν να βρεθούν σχετικά με δυωνυμικούς συντελεστές στο βιβλίο του Bryant [18] και του Govinda Rao [58]. Απαιτητικό αλλά αξεπέραστο σε θεμελιώσεις ανάλυσης αλγορίθμων είναι το βιβλίο των Sedgewick-Flajolet [144].

Άπειρες είναι οι αναφορές στη βιβλιογραφία σχετικά με τους ασυμπτωτικούς συμβολισμούς. Στο βιβλίο του Knuth [59] αναφέρεται ότι ο συμβολισμός O προτάθηκε για πρώτη φορά το 1894 από το μαθηματικό Paul Bachmann [4]. Στο άρθρο [78] αναφέρεται το ιστορικό του καθορισμού του κάθε συμβολισμού, ενώ το άρθρο [134] βοηθά στην περαιτέρω κατανόησή τους. Η βιβλιογραφία σχετικά με την Υπολογιστική Πολυπλοκότητα είναι ιδιαίτερα εκτενής. Σημειώνεται το κλασικό βιβλίο των Garey-Johnson [45], όπου γίνεται μία εις πλάτος παρουσίαση των προβλημάτων της κλάσης NP, καθώς και τα βιβλία του Παπαδημητρίου για μία γενική θεώρηση του αντικειμένου [88, 122].

Η Άσκηση 6 βασίζεται στο βιβλίο [147], που αποτελεί μία καλή εισαγωγική προσέγγιση στο αντικείμενο, ενώ οι Ασκήσεις 3-5 και 23-24 στο βιβλίο των Graham-Knuth-Patashnik [59], που αποτελεί μία τεράστια πηγή προβλημάτων. Η Άσκηση 7 έχει τεθεί στη Βαλκανική Ολυμπιάδα Πληροφορικής του 1997, η Άσκηση 18 βασίζεται στο άρθρο [81], ενώ η Άσκηση 28 βασίζεται στο άρθρο [46].

2.7 Ασκήσεις

1. **Η αρχή των περιστερώνων** (Pigeonhole principle). Δεδομένων n περιστερώνων και $m > n$ περιστεριών να αποδειχθεί ότι τουλάχιστον ένας περιστερώνας θα δεχθεί περισσότερο από ένα περιστέρι.

2. **Η αρχή του κουτιού του Dirichlet** (Dirichlet box principle). Δεδομένων n αντικειμένων και m κουτιών να αποδειχθεί ότι κάποια κουτιά περιέχουν

περισσότερο από $\lceil n/m \rceil$ αντικείμενα και κάποια κουτιά περιέχουν λιγότερο από $\lfloor n/m \rfloor$ αντικείμενα.

3. Σε ένα καζίνο υπάρχει ρουλέτα με 1000 αριθμούς. Αν η ρουλέτα φέρει έναν αριθμό n που διαιρείται με την ποσότητα $\lfloor \sqrt[3]{n} \rfloor$, τότε ο παίκτης κερδίζει 5 ευρώ αλλιώς χάνει 1 ευρώ. Ασυμπτωτικά, ο παίκτης θα κερδίσει ή θα χάσει.

4. Να αποδειχθεί ότι η επόμενη έκφραση ισούται με $\lfloor x \rfloor$ ή με $\lceil x \rceil$. Πότε εμφανίζεται το κάθε αποτέλεσμα;

$$\left\lceil \frac{2x+1}{2} \right\rceil - \left\lceil \frac{2x+1}{4} \right\rceil = \left\lfloor \frac{2x+1}{4} \right\rfloor$$

5. Να αποδειχθούν οι σχέσεις για n, m ακεραίους θετικούς:

   - $n = \lfloor \frac{n}{m} \rfloor + \lfloor \frac{n-1}{m} \rfloor + \ldots \lfloor \frac{n-m+1}{m} \rfloor$
   - $n = \lceil \frac{n}{m} \rceil + \lceil \frac{n-1}{m} \rceil + \ldots \lceil \frac{n-m+1}{m} \rceil$
   - $\lfloor \frac{n}{m} \rfloor = \lceil \frac{n-m+1}{m} \rceil$

6. Ένας άνδρας αποφασίζει να επισκεφθεί μία φίλη του που μένει 60 χλμ μακριά. Αρχίζει το ταξίδι οδηγώντας με ταχύτητα 60 χλμ/ώρα. Μετά από 1 χλμ πέφτει ο ενθουσιασμός του και αυτομάτως η ταχύτητα μειώνεται στα 59 χλμ/ώρα. Και πάλι, μετά από ένα 1 χλμ η ταχύτητα μειώνεται στα 58 χλμ/ώρα κοκ. Σε πόση ώρα θα φθάσει στο σπίτι της φίλης του;

7. Δεδομένου ενός θετικού ακέραιου αριθμού n, να βρεθούν όλοι οι θετικοί ακέραιοι x (αν υπάρχουν) για τους οποίους ισχύει ότι ο αριθμός $x!$ έχει n δεκαδικά ψηφία ακριβώς. Να υποτεθεί ότι $n \leq 150.000$.

8. Να αποδειχθεί ότι:

   - $n^{n/2} \leq n! \leq \frac{(n+1)^n}{2^n}$

9. Να αποδειχθεί ότι:

   - $\sum_{i=k}^{n-1} \binom{i}{k} = \binom{n}{k+1}$
   - $\sum_{i=1}^{n} \binom{n-i+1}{k-i+1} = \binom{n+1}{k}$
   - $\binom{r}{m}\binom{m}{k} = \binom{r}{k}\binom{r-k}{m-k}$

10. Να αποδειχθεί ότι για κάθε $n \geq 0$ και $0 \leq k \leq n$, η ποσότητα $\binom{n}{k}$ μεγιστοποιείται όταν $k = \lfloor n/2 \rfloor$ ή $k = \lceil n/2 \rceil$.

11. Να αποδειχθεί ότι:

    - $(F_n)^2 \;=\; F_{n-1}\, F_{n+1} - (-1)^n$
    - $\sum_{i=0}^{n} F_i \;=\; F_{n+2} - 1$
    - $\sum_{i=0}^{n} (F_i)^2 \;=\; F_n\, F_{n+1}$
    - $F_n \;<\; \left(\frac{5}{3}\right)^n$
    - $F_{2n} \;=\; F_n\, F_{n+1} + F_{n-1} F_n$
    - $F_{2n+1} \;=\; (F_{n+1})^2 + (F_n)^2$
    - τα $F_{3n}, F_{3n+1}, F_{3n+2}$ είναι άρτιος, περιττός και άρτιος αριθμός, αντιστοίχως.

12. Να υπολογισθούν οι ποσότητες:

    - $\sum_{i=\lfloor n/2 \rfloor}^{n} \frac{1}{i}$
    - $\sum_{i=0}^{\lg n} 2^i$
    - $\sum_{i=0}^{\infty} \frac{i-1}{2^i}$
    - $\sum_{i=1}^{\infty} (2i+1)\, x^{2i}$
    - $\sum_{i=1}^{\infty} (-1)^i\, i/(4i^2-1)$

13. Να υπολογισθούν τα αθροίσματα:

    - $\sum_{i=0}^{\infty} \frac{1}{4^i}$
    - $\sum_{i=0}^{\infty} \frac{i}{4^i}$
    - $\sum_{i=0}^{\infty} \frac{i^2}{4^i}$
    - $\sum_{i=0}^{\infty} \frac{i^n}{4^i}$

14. Να υπολογισθούν οι ποσότητες:

    - $\sum_{i=1}^{n} (2i+1)/(i(i+1))$
    - $\sum_{i=0}^{n-1} H_i$
    - $\sum_{i=0}^{n} i\, H_i$
    - $\sum_{i=0}^{n-1} H_i/(i+1)(i+2)$

15. Σε κάθε μία από τις επόμενες περιπτώσεις να βρεθούν οι συμβολισμοί O και Ω, καθώς επίσης να υπολογισθούν τιμές για τις σταθερές c_i και n_0.

- $c_1\ n^3 + c_2$
- $c_3\ n \log n + c_4\ n$
- $c_5\ 2^n + c_6\ n^5$
- $c_7\ n! + c_8\ n^3$

16. Σε κάθε μία από τις επόμενες περιπτώσεις να σημειωθεί αν ισχύει: $f = O(g(n))$, $f = \Omega(g(n))$ ή $f = \Theta(g(n))$.

- $f(n) = \sqrt{n}$ $\quad g(n) = \log^3 n$
- $f(n) = n^{1/2}$ $\quad g(n) = 5^{\log n}$
- $f(n) = 2^n$ $\quad g(n) = 2^{n+1}$
- $f(n) = \sum_{i=1}^{n} i^k$ $\quad g(n) = n^{k+1}$

17. Να διαταχθούν κατά αύξουσα σειρά οι συναρτήσεις: $n, \sqrt{n}, n^{1.5}, n^2, n \log n$, $10, n \log \log n, n\log^2 n, n^3, n \log n^2, 1/n, 2^n, 2^n/2, n^2 \log n$. Ποιές συναρτήσεις είναι της ιδίας τάξης;

18. Δίνεται ο επόμενος ψευδοκώδικας. Να βρεθεί ο συμβολισμός Ω. Μπορεί να βρεθεί ο συμβολισμός O;

```
while (n>1)
   if (n mod 2 = 1) then
      n <-- 3*n+1;
   else
      n <-- n/2;
```

19. Δίνονται τα επόμενα δύο τμήματα ψευδοκώδικα. Να βρεθεί ο συμβολισμός O.

```
for i <-- 0 to n do
   j <--i;
   while j<>0 do j <-- j div 2;

for i <-- 0 to n do
   j <--i;
   while (j mod 2 = 1) do j <-- j div 2;
```

20. Δίνονται τα επόμενα δύο τμήματα κώδικα C. Να βρεθεί ο συμβολισμός Θ.

```
sum1=0;
for (i=1; i<=n; i*=2)
   for (j=1; j<=n; j++)
      sum1++;

sum2=0;
for (i=1; i<=n; i*=2)
   for (j=1; j<=i; j++)
      sum2++;
```

21. Δίνονται το επόμενο τμήμα κώδικα C, όπου ο πίνακας `A[0..n-1]` περιέχει μία τυχαία διάταξη των αριθμών από 1 μέχρι `n`. Να βρεθεί ο συμβολισμός Θ.

```
sum3=0;
for (i=0; i<n; i++)
   for (j=0; A[j]!=i; j++)
      sum3++;
```

22. Να αποδειχθεί ότι (αν) ισχύει:
    - $\log^k n = \mathrm{o}(n)$ για κάθε σταθερά k
    - $\log n! = \Theta(n \log n)$
    - $2^n = \Theta(3^n)$

23. Να εκφρασθεί με συμβολισμό O το γινόμενο $(\ln n + \gamma + \mathrm{O}(1/n))$ επί $(n + \mathrm{O}(\sqrt{n}))$.

24. Να αποδειχθεί ότι ισχύουν οι επόμενες σχέσεις για $n \to \infty$:
    - $1 + \frac{2}{n} + \mathrm{O}(n^{-2}) \;=\; \left(1 + \frac{2}{n}\right)\left(1 + \mathrm{O}(n^{-2})\right)$
    - $\mathrm{O}\left(\left(\frac{n^2}{\log\log n}\right)^{1/2}\right) \;=\; \mathrm{O}(\lfloor\sqrt{n}\rfloor^2)$
    - $\mathrm{e}^{(1+\mathrm{O}(1/n))^2} \;=\; \mathrm{e} + \mathrm{O}(1/n)$

25. Να αποδειχθεί επαγωγικά ότι:
    - $\left(1 + \frac{1}{1}\right) \cdot \left(1 + \frac{1}{2}\right) \cdot \ldots \cdot \left(1 + \frac{1}{n}\right) \;=\; n + 1 \quad \text{για} \quad n \geq 1$
    - $\left(1 - \frac{1}{2^2}\right) \cdot \left(1 - \frac{1}{3^2}\right) \cdot \ldots \cdot \left(1 - \frac{1}{n^2}\right) \;=\; \frac{n+1}{2n} \quad \text{για} \quad n \geq 2$

- $\prod_{i=0}^{n} \left(\frac{1}{2i+1} \cdot \frac{1}{2i+2} \right) = \frac{1}{(2n+2)!}$ για $n \geq 0$

26. Να αποδειχθεί επαγωγικά ότι το άθροισμα των γωνιών ενός κυρτού πολυγώνου με n κορυφές είναι $(n-2) \cdot 180^o$.

27. Να αποδειχθεί επαγωγικά ότι ένας τετραγωνικός πίνακας $2^k \times 2^k$ μπορεί να καλυφθεί πλήρως με πλακάκια που αποτελούνται από 3 τετράγωνα όπως στο Σχήμα 2.2, εκτός από ένα μόνο τετράγωνο.

Σχήμα 2.2: Το πρόβλημα με τα πλακάκια.

28. Να αποδειχθεί επαγωγικά ότι κάθε κλάσμα $\frac{p}{q}$, όπου $0 < \frac{p}{q} < 1$ μπορεί να παρασταθεί με τη λεγόμενη Αιγυπτιακή μορφή $\frac{p}{q} = \frac{1}{n_1} + \frac{1}{n_2} + \ldots \frac{1}{n_m}$, όπου τα n_i (για $1 \leq i \leq m$) είναι θετικοί ακέραιοι που ικανοποιούν τη συνθήκη $n_1 < n_2 < \ldots < n_m$.

3

Αναδρομικές Εξισώσεις

Περιεχόμενα Κεφαλαίου

Όπως κατά την μελέτη των ταξινομήσεων με εισαγωγή και επιλογή στο Κεφάλαιο 1 προέκυψαν εξισώσεις με την έκφραση $T(n)$ στο αριστερό σκέλος, συχνά κατά την ανάλυση αλγορίθμων προκύπτουν αντίστοιχες εξισώσεις, που όμως είναι αναδρομικές. Αυτό συμβαίνει όταν σχεδιάζεται ένας αλγόριθμος που σπάει το πρόβλημα σε μικρότερα υποπροβλήματα και εν συνεχεία εκτελεί αναδρομικά την ίδια ακολουθία βημάτων για κάθε υποπρόβλημα. Η επίλυση αναδρομικών εξισώσεων είναι ένα απαραίτητο εργαλείο για την εύρεση κλειστών εκφράσεων που περιγράφουν την πολυπλοκότητα πολλών αλλά και βασικών αλγορίθμων.

Ορισμός.
Για μία ακολουθία αριθμών $a_0, a_1, a_2, \ldots, a_n, \ldots$ μία εξίσωση που εκφράζει τον όρο a_n με βάση τους προηγούμενους στην ακολουθία καλείται **αναδρομική σχέση**. Οι τιμές (ή τιμή) που πρέπει να γνωρίζουμε ώστε να ξεκινήσει ο υπολογισμός ενός στοιχείου της ακολουθίας με βάση την αναδρομική σχέση καλούνται **αρχικές συνθήκες**. □

Στο Κεφάλαιο 2 αναφέρθηκε η ακολουθία των αριθμών Fibonacci, 1,1,2,3,5,8, 13,21, κοκ, όπου οι δύο πρώτοι αριθμοί της ακολουθίας ισούνται με 1, ενώ κάθε επόμενος ισούται με το άθροισμα των δύο προηγουμένων. Η ακολουθία αυτή περιγράφεται εύκολα από την αναδρομική σχέση $a_n = a_{n-1} + a_{n-2}$, για $n \geq 2$ με $a_0 = 1, a_1 = 1$. Ωστόσο, είναι δύσκολο να βρεθεί μέσω της απλής παρατήρησης μία γενική έκφραση για το a_n. Αυτό αποτελεί και το κύριο ενδιαφέρον διότι αν το πλήθος των βημάτων ενός αλγορίθμου εκφρασθεί μέσω μίας αναδρομικής σχέσης, τότε μπορούμε να εκτιμήσουμε τη χρονική πολυπλοκότητα τού προσδιορίζοντας τον γενικό όρο a_n.

Στον Πίνακα 3.1 παρουσιάζεται η κατηγοριοποίηση των αναδρομικών σχέσεων αναλόγως με το πλήθος των απαραίτητων όρων για την έκφραση του a_n και με τον τρόπο χρήσης αυτών των όρων. Στη συνέχεια θα εξετάσουμε τρόπους προσδιορισμού ή φραγής του γενικού όρου a_n μίας αναδρομικής σχέσης αναλόγως με την κλάση όπου ανήκει. Γενικώς, μία αναδρομική εξίσωση μπορεί να επιλυθεί με τους εξής μεθόδους:

- με αντικατάσταση,
- με επανάληψη,
- με αναγωγή στη χαρακτηριστική εξίσωση,
- με το γενικό θεώρημα, και

Αναδρομικές Σχέσεις	**Παράδειγμα**
Πρώτης Τάξης	
◦ γραμμικές	$a_n = na_{n-1} - 1$
◦ μη-γραμμικές	$a_n = \frac{1}{1+a_{n-1}}$
Δεύτερης Τάξης	
◦ γραμμικές	$a_n = a_{n-1} + 2a_{n-2}$
◦ μη-γραμμικές	$a_n = a_{n-1}a_{n-2} + 2\sqrt{a_{n-2}}$
◦ γραμμικές με μεταβλητούς συντελεστές	$a_n = na_{n-1} + (n-1)a_{n-2} + 1$
k **Τάξης**	$a_n = F(a_{n-1}, a_{n-2}, \ldots, a_{n-k})$
Πλήρους Ιστορίας	$a_n = F(a_{n-1}, a_{n-2}, \ldots, a_1)$
Διαίρει και Βασίλευε	$a_n = a_{\lceil n/2 \rceil} + a_{\lfloor n/2 \rfloor} + n$

Πίνακας 3.1: Κατηγοριοποίηση αναδρομικών εξισώσεων.

- με γεννήτριες συναρτήσεις.

Κατ'αρχήν, λοιπόν, θα εξετασθούν οι μέθοδοι της αντικατάστασης και της επανάληψης, στη συνέχεια μία σειρά τεχνικών αναγωγής στη χαρακτηριστική εξίσωση (Κεφάλαια 3.3-3.6), και τέλος η μέθοδος με βάση το γενικό θεώρημα. Η μέθοδος των γεννητριών συναρτήσεων θα παρουσιασθεί εκτενέστερα στο Κεφάλαιο 4.

3.1 Η Μέθοδος της Αντικατάστασης

Στη μέθοδο της αντικατάστασης γίνεται μία ασυμπτωτική εκτίμηση της λύσης, η οποία στην συνέχεια επαληθεύεται επαγωγικά.

Δίνεται η σχέση a_n=$2a_{n/2}$+n και εκτιμάται ότι η λύση είναι a_n=O($n \log n$). Για να είναι σωστή αυτή η εκτίμηση, θα πρέπει να ισχύει $a_n \leq cn \log n$ για κάποια σταθερά $c > 0$. Θα δείξουμε επαγωγικά ότι η πρόβλεψη αυτή είναι ορθή. Υποθέτουμε ότι για τιμές μικρότερες του n η ανισότητα ισχύει, οπότε:

$$\begin{aligned} a_n &= 2a_{n/2} + n \\ &\leq 2\ c\ \frac{n}{2} \log \frac{n}{2} + n \\ &= c\ n \log \frac{n}{2} + n \\ &= c\ n \log n - c\ n + n \\ &\leq c\ n \log n \end{aligned}$$

κάτι που ισχύει για κάθε σταθερά $c \geq 1$.

Ας εξετάσουμε ένα ακόμη παράδειγμα. Έστω η σχέση $a_n = a_{n-1} + n$ για $n > 0$ και $a_0 = 0$. Υποπτευόμαστε ότι ο γενικός όρος είναι της τάξης $\mathrm{O}(n^2)$, δηλαδή ότι ισχύει $a_n \leq cn^2$ για κάποια σταθερά $c > 0$ και $n > n_0$ για κάποιο n_0. Θα δείξουμε και πάλι επαγωγικά ότι η υποψία μας είναι ορθή. Υποθέτουμε ότι για τιμές μικρότερες του n η ανισότητα ισχύει, οπότε:

$$\begin{aligned} a_n &= a_{n-1} + n \\ &\leq c\,(n-1)^2 + n \\ &= c\,n^2 + (1-2c)\,n + c \\ &\leq c\,n^2 \end{aligned}$$

όπου η τελευταία ανισοϊσότητα ισχύει, για παράδειγμα για $c = 2$ και $n_0 \geq 1$.

3.2 Η Μέθοδος της Επανάληψης

Πολλές φορές αυτές οι αναδρομικές εξισώσεις επιλύονται με τη μέθοδο της επανάληψης. Η μέθοδος αυτή, όπως θα φανεί με τα επόμενα δύο παραδείγματα, ίσως είναι η απλούστερη (όταν μπορεί να εφαρμοσθεί), καθώς είναι αρκετά μηχανιστική.

Για παράδειγμα, δίνεται η αναδρομική εξίσωση:

$$T(n) \;=\; 4\,T(n/2) + n$$

όπου $n > 1$, ενώ η αρχική συνθήκη είναι $T(1) = 1$. Χωρίς βλάβη της γενικότητας, δεχόμαστε ότι ισχύει $n = 2^k$, οπότε $k = \log n$. Αν στην προηγούμενη σχέση αντικαταστήσουμε την ποσότητα $T(n/2)$, επαναλαμβάνοντας την εφαρμογή της αναδρομής με κατάλληλα μειωμένα τα ορίσματα, τότε προκύπτει:

$$\begin{aligned} T(n) &= 4\;[4\,T(n/4) + n/2] + n \\ &= 4^2\,T(n/2^2) + 2n + n \end{aligned}$$

Με μία-δύο ακόμη επαναλήψεις αντιλαμβανόμαστε το μηχανισμό παραγωγής των επόμενων σχέσεων με τις διαδοχικές αντικαταστάσεις των αναδρομών με μειωμένα ορίσματα. Άρα ισχύει ότι:

$$T(n) \;=\; 4^k\,T(n/2^k) + (2^{k-1}n + \ldots + 2^1 n + 2^0 n)$$

Λόγω της αρχικής συνθήκης ισχύει:

$$\begin{aligned} T(n) &= 4^k + (2^{k-1}n + \ldots + 2^1 n + 2^0 n) \\ &= 4^k + n\ (1 + 2 + \ldots + 2^{k-1}) \\ &= 4^k + n\ \frac{2^k - 1}{2 - 1} \\ &= 4^{\log n} + n(n-1) \\ &= 2n^2 - n \end{aligned}$$

Έστω τώρα ένα δεύτερο παράδειγμα. Δίνεται η αναδρομική εξίσωση:

$$T(n) \ = \ 4\ T(n/2) + n^2/\log n$$

όπου και πάλι δεχόμαστε ότι $n = 2^k$ και $T(1) = 1$. Με μία πρώτη επανάληψη προκύπτει ότι:

$$\begin{aligned} T(n) &= 4^2\ T(n/2^2) + n^2/\log n + n^2/\log(n/2) \\ &= 4^2\ T(n/2^2) + n^2 \left(\frac{1}{\log n} + \frac{1}{\log n - 1}\right) \end{aligned}$$

και με διαδοχικές επανλήψεις βρίσκουμε ότι:

$$\begin{aligned} T(n) &= 4^k\ T(n/2^k) + n^2 \left(\frac{1}{\log n} + \frac{1}{\log n - 1} + \ldots + \frac{1}{\log n - (k-1)}\right) \\ &= 4^k + n^2\ \left(\frac{1}{\log n} + \frac{1}{\log n - 1} + \ldots + 1\right) \\ &= n^2 + n^2\ H_{\log n} \ \approx \ n^2 + n^2\ \log(\log n) \\ &= \Theta(n^2\ \log(\log n)) \end{aligned}$$

3.3 Ομογενείς Γραμμικές Αναδρομές

Η απλούστερη περίπτωση αναδρομικής εξίσωσης είναι της μορφής:

$$a_0 t_n + a_1 t_{n-1} + a_2 t_{n-2} + \ldots a_k t_{n-k} \ = \ 0$$

και λέγεται γραμμική (linear) γιατί δεν περιέχει δυνάμεις και γινόμενα των t_i, λέγεται ομογενής (homogeneous) γιατί ο γραμμικός συνδυασμός των t_i ισούται με 0, ενώ επίσης έχει σταθερούς συντελεστές a_i. Έστω ότι αναζητούμε μία λύση της μορφής:

$$t_n \ = \ x^n$$

όπου το x είναι μία σταθερά. Αντικαθιστώντας στην αναδρομική εξίσωση έχουμε:

$$a_0x^n + a_1x^{n-1} + a_2x^{n-2} + \ldots a_kx^{n-k} \;=\; 0 \Rightarrow$$

$$p(x) \;=\; a_0x^k + a_1x^{k-1} + a_2x^{k-2} + \ldots a_k \;=\; 0$$

που αποτελεί τη **χαρακτηριστική εξίσωση** της αναδρομής. Ας υποθέσουμε ότι η εξίσωση έχει k διακριτές (πραγματικές) ρίζες $r_1, r_2, \ldots, r_k$. Στην περίπτωση αυτή κάθε γραμμικός συνδυασμός:

$$t_n \;=\; \sum_{i=1}^{k} c_i r_i^n$$

είναι λύση της αναδρομής, όπου οι σταθερές c_i εξαρτώνται από τις αρχικές συνθήκες.

Για παράδειγμα, έστω ότι δίνεται η αναδρομική εξίσωση:

$$t_k - 5\, t_{k-1} + 6\, t_{k-2} \;=\; 0$$

με αρχικές συνθήκες $t_0 = 3$ και $t_1 = 7$. Η χαρακτηριστική εξίσωση είναι $x^2 - 5x + 6 = 0$ με ρίζες $r_1 = 2$ και $r_2 = 3$. Συνεπώς καταλήγουμε ότι:

$$t_k \;=\; c_0\, 2^k + c_1\, 3^k$$

Επιλύοντας το σύστημα δύο εξισώσεων με δύο αγνώστους, που προκύπτει από τις αρχικές συνθήκες, έχουμε:

$$\begin{aligned} 3 &= c_0 + c_1 \\ 7 &= 2\, c_0 + 3\, c_1 \end{aligned}$$

από όπου προκύπτει ότι $c_0 = 2, c_1 = 1$ και επομένως η λύση της αναδρομής είναι $t_k = 2^{k+1} + 3^k$.

Τώρα ας υποθέσουμε ότι οι λύσεις δεν είναι όλες διακριτές μεταξύ τους αλλά υπάρχει μία πολλαπλή ρίζα r βαθμού d. Επομένως το πολυώνυμο $p(x)$ μπορεί να γραφεί ως εξής:

$$p(x) \;=\; (x-r)^d g(x)$$

όπου το $g(x)$ δεν έχει για ρίζα το r οπότε δεν διαιρείται με το $(x-r)$. Παραγωγίζουμε και λαμβάνουμε:

$$p'(x) \;=\; (x-r)^d g'(x) + d(x-r)^{d-1} g(x) \;=\; (x-r)^{d-1}\left((x-r)g'(x) + dg(x)\right)$$

όπου παρατηρούμε ότι το $p'(x)$ έχει πολλαπλή ρίζα r βαθμού $d-1$. Το συμπέρασμα αυτό θα το χρησιμοποιήσουμε στη συνέχεια. Τώρα, εφ'όσον το πολυώνυμο $p(x)$ έχει πολλαπλή ρίζα r βαθμού d, έπεται ότι το r είναι πολλαπλή ρίζα βαθμού d και του πολυωνύμου:

$$x^{k-m}p(x) \;=\; a_0x^k + a_1x^{k-1} + \ldots + a_mx^{k-m}$$

Παραγωγίζουμε αυτή την έκφραση, πολλαπλασιάζουμε επί x και λαμβάνουμε:

$$a_0kx^k + a_1(k-1)x^{k-1} + \ldots + a_m(k-m)x^{k-m}$$

Με βάση την ανωτέρω παρατήρηση, έπεται ότι η τελευταία έκφραση έχει πολλαπλή ρίζα r βαθμού $d-1$. Άρα ισχύει:

$$a_0kr^k + a_1(k-1)r^{k-1} + \ldots + a_m(k-m)r^{k-m} \;=\; 0$$

Αν προσέξουμε καλύτερα αυτήν την αναδρομική εξίσωση, διαπιστώνουμε ότι έχει λύση $t_k = kr^k$. Περαιτέρω, παραγωγίζοντας και πολλαπλασιάζοντας επί x και πάλι προκύπτει η σχέση:

$$a_0k^2x^k + a_1(k-1)^2x^{k-1} + \ldots + a_m(k-m)^2x^{k-m}$$

με πολλαπλή ρίζα r βαθμού $d-2$. Επομένως προκύπτει η αναδρομική εξίσωση:

$$a_0k^2r^k + a_1(k-1)^2r^{k-1} + \ldots + a_m(k-m)^2r^{k-m} \;=\; 0$$

με λύση $t_k = k^2r^k$. Η διαδικασία αυτή μπορεί να συνεχισθεί για $d-1$ βήματα συνολικά. Έτσι καταλήγουμε στο εξής συμπέρασμα: Αν ένα χαρακτηριστικό πολυώνυμο μίας ομογενούς γραμμικής αναδρομικής εξίσωσης έχει πολλαπλή ρίζα r βαθμού d, τότε οι λύσεις της εξίσωσης είναι $t_k = r^k, kr^k, k^2r^k, \ldots, k^{d-1}r^k$. Όπως και πριν, η γενική λύση της αναδρομικής εξίσωσης είναι γραμμικός συνδυασμός αυτών των λύσεων και πρέπει να ληφθούν υπ'όψη οι αρχικές συνθήκες.

Για παράδειγμα, έστω ότι δίνεται η αναδρομική εξίσωση:

$$t_k - 4\,t_{k-1} + 4\,t_{k-2} \;=\; 0$$

με αρχικές συνθήκες $t_0 = 2$ και $t_1 = 10$. Η χαρακτηριστική εξίσωση είναι $x^2 - 4x + 4 = (x-2)^2 = 0$ με διπλή ρίζα $r = 2$. Συνεπώς καταλήγουμε ότι:

$$t_k \;=\; c_0\,2^k + c_1\,k2^k$$

Επιλύοντας το σύστημα δύο εξισώσεων με δύο αγνώστους, που προκύπτει από τις αρχικές συνθήκες, έχουμε:

$$\begin{aligned} 2 &= c_0 \\ 10 &= 2\ c_0 + 2\ c_1 \end{aligned}$$

από όπου προκύπτει ότι $c_0 = 2, c_1 = 3$ και επομένως η λύση της αναδρομής είναι $t_k = 2^{k+1} + 3\ k2^k$.

3.4 Μη Ομογενείς Γραμμικές Αναδρομές

Ας θεωρήσουμε τώρα μη ομογενείς γραμμικές αναδρομικές εξισώσεις, δηλαδή εξισώσεις υπό την επόμενη μορφή, όπου ο γραμμικός συνδυασμός δεν ισούται με μηδέν:

$$a_0 t_n + a_1 t_{n-1} + a_2 t_{n-2} + \ldots a_k t_{n-k} \ = \ b_1^n p_1(n) + b_2^n p_2(n) + \ldots$$

ενώ τα $b_1, b_2, \ldots$ είναι σταθερές. Στο σημείο αυτό χωρίς απόδειξη δίνεται ότι ισοδύναμο είναι να θεωρηθεί η ομογενής γραμμική αναδρομική εξίσωση:

$$(a_0 x^k + a_1 x^{k-1} + a_2 x^{k-2} + \ldots a_k)(x - b_1)^{d_1+1}(x - b_2)^{d_2+1} \ldots \ = \ 0$$

όπου η πρώτη παρένθεση αποτελεί τη χαρακτηριστική εξίσωση του αριστερού σκέλους της μη ομογενούς αναδρομικής εξίσωσης, ενώ d_i είναι ο βαθμός του πολυωνύμου $p_i(n)$.

Για παράδειγμα, έστω ότι δίνεται η μη ομογενής αναδρομική εξίσωση:

$$t_k - 2\ t_{k-1} \ = \ 2^k - 1$$

με αρχική συνθήκη $t_0 = 0$. Θεωρώντας ότι ισχύει: $b_1 = 2, b_2 = 1, p_1(n) = 1$ και $p_2(n) = -1$, η εξίσωση μετασχηματίζεται στην ομογενή:

$$(x - 2)(x - 1)(x - 2) \ = \ 0$$

Έτσι φθάνουμε σε μία ομογενή γραμμική αναδρομική εξίσωση με γενική λύση:

$$t_k \ = \ c_0\ 1^x + c_1\ 2^x + c_2\ k2^k$$

Πρέπει να επιλύσουμε ένα σύστημα τριών εξισώσεων με τρεις αγνώστους. Εύκολα υπολογίζουμε ότι $t_1 = 1, t_2 = 5$:

$$\begin{aligned} 0 &= c_0 + c_1 \\ 1 &= c_0 + 2\ c_1 + 2\ c_2 \\ 5 &= c_0 + 4\ c_1 + 8\ c_2 \end{aligned}$$

από όπου προκύπτει ότι $c_0 = 1, c_1 = -1, c_3 = 1$ και επομένως η λύση της αναδρομής είναι $t_k = (k-1)\ 2^k + 1$.

Συνήθως οι μη ομογενείς αναδρομικές εξισώσεις μπορούν να επιλυθούν και με απλή άλγεβρα. Ας θεωρήσουμε και πάλι το προηγούμενο παράδειγμα με την εξίσωση $t_k - 2\ t_{k-1} = 2^k - 1$. Πρώτον, πολλαπλασιάζουμε και τα δύο σκέλη της εξίσωσης επί δύο, οπότε προκύπτει:

$$2\ t_k - 4\ t_{k-1} = 2^{k+1} - 2$$

Δεύτερον, αντικαθιστούμε όπου k με $k+1$, οπότε προκύπτει:

$$t_{k+1} - 2\ t_k = 2^{k+1} - 1$$

Από τις δύο τελευταίες σχέσεις προκύπτει ότι:

$$t_{k+1} - 4\ t_k + t_{k-1} = 1$$

Ούτε και αυτή η εξίσωση είναι ομογενής αλλά έχουμε κάνει ένα βήμα. Τρίτον, λοιπόν, στην τελευταία αυτή σχέση αντικαθιστούμε το k με $k+1$, οπότε προκύπτει:

$$t_{k+2} - 4\ t_{\varkappa+1} + t_\varkappa = 1$$

Εξισώνοντας τα αριστερά σκέλη των δύο τελευταίων εξισώσεων προκύπτει η ομογενής εξίσωση:

$$t_{k+2} - 5\ t_{k+1} + 8\ t_k - 4\ t_{k-1} = 0$$

που επιλύεται κατά τα γνωστά, καθώς έχει χαρακτηριστική εξίσωση:

$$x^3 - 5x^2 + 8x - 4 = (x-1)(x-2)^2 = 0$$

3.5 Αλλαγή Μεταβλητής

Είναι δυνατόν κατά την ανάλυση αλγορίθμων να προκύψουν εξισώσεις με την έκφραση $T(n)$ στο αριστερό σκέλος, οι οποίες όμως δεν επιλύονται με τη μέθοδο της επανάληψης. Τότε αντικαθιστούμε το αριστερό σκέλος με την έκφραση t_n με κάποια κατάλληλη αλλαγή της μεταβλητής (variable change), ώστε να προκύψει μία ομογενής ή μη ομογενής γραμμική αναδρομική εξίσωση, που να επιλύεται με βάση τη χαρακτηριστική εξίσωση.

Για παράδειγμα, έστω η απλή αναδρομική εξίσωση που επιλύθηκε με τη μέθοδο της επανάληψης στο Κεφάλαιο 3.2:

$$T(n) \;=\; 4\,T(n/2) + n$$

όπου το $n > 1$ είναι δύναμη του 2. Έστω ότι ισχύει $n = 2^k$, οπότε $k = \log n$. Έτσι προκύπτει η σχέση:

$$T(2^k) \;=\; 4\,T(2^{k-1}) + 2^k$$

που την ξαναγράφουμε με τη μορφή:

$$t_k \;=\; 4\,t_{k-1} + 2^k$$

Για αυτή τη μη ομογενή αναδρομική εξίσωση μπορούμε εύκολα να βρούμε τη χαρακτηριστική εξίσωση:

$$(x-4)(x-2) \;=\; 0$$

οπότε η γενική λύση είναι της μορφής:

$$t_k \;=\; c_1\,4^k + c_2\,2^k$$

Επιστρέφοντας στην αρχική μεταβλητή, καταλήγουμε στην εξίσωση

$$T(n) \;=\; c_1\,n^2 + c_2\,n$$

που επιλύεται αν μας δοθούν οι αρχικές συνθήκες.

Προχωρούμε τώρα σε ένα συνθετότερο παράδειγμα που δεν μπορεί να επιλυθεί με τα μέχρι στιγμής γνωστά. Γενικώς, θα πρέπει να γίνονται μετασχηματισμοί ώστε να αναγόμαστε στα γνωστά. Μία μέθοδος, λοιπόν, είναι να αλλάζουμε το εύρος της μεταβλητής (range transformation). Για παράδειγμα, έστω ότι δίνεται η αναδρομική εξίσωση:

$$T(n) \;=\; n\,T^2(n/2)$$

όπου το $n > 1$ είναι δύναμη του 2, ενώ δίνεται και η αρχική συνθήκη $T(1) = 6$. Σύμφωνα με την προηγούμενη τεχνική (δηλαδή, αλλαγή μεταβλητής θέτοντας $n = 2^k$) καταλήγουμε στην ισοδύναμη έκφραση:

$$t_k \;=\; 2^k\,t_{k-1}^2$$

όπου $k > 0$, ενώ πλέον η αρχική συνθήκη είναι $t_0 = 6$. Τώρα προχωρούμε σε μία αλλαγή μεταβλητής, όπου όμως ταυτόχρονα αλλάζουμε και το πεδίο ορισμού. θέτουμε, λοιπόν, $V_k = \log t_k$, οπότε προκύπτει:

$$V_k \;=\; k + 2\; V_{k-1}$$

με αρχική συνθήκη $V_0 = \log 6$. Η τελευταία αναδρομική εξίσωση έχει χαρακτηριστική εξίσωση:

$$(x-2)(x-1)^2 \;=\; 0$$

οπότε σύμφωνα με όσα αναπτύξαμε στην παράγραφο σχετικά με πολλαπλές ρίζες, προκύπτει η γενική λύση:

$$V_k \;=\; c_1\; 2^k + c_2\; 1^k + c_3\; k1^k$$

Από την αναδρομική εξίσωση ως προς V_k βρίσκουμε τις αρχικές συνθήκες:

$$\begin{aligned} V_0 &= 1 + \log 3 \\ V_1 &= 3 + 2\;\log 3 \\ V_2 &= 8 + 4\;\log 3 \end{aligned}$$

εκ των οποίων προκύπτουν οι εξής τιμές για τις σταθερές επιλύοντας ένα σύστημα τριών εξισώσεων με τρεις αγνώστους:

$$\begin{aligned} c_1 &= 3 + \log 3 \\ c_2 &= -2 \\ c_3 &= -1 \end{aligned}$$

Επομένως προκύπτει:

$$V_k \;=\; (3 + \log 3)\; 2^k - k - 2$$

από όπου με δύο αντίστροφους μετασχηματισμούς (δηλαδή, $t_k = 2^{V_k}$ και $T(n) = t_{\log n}$) σταδιακά προκύπτει ότι:

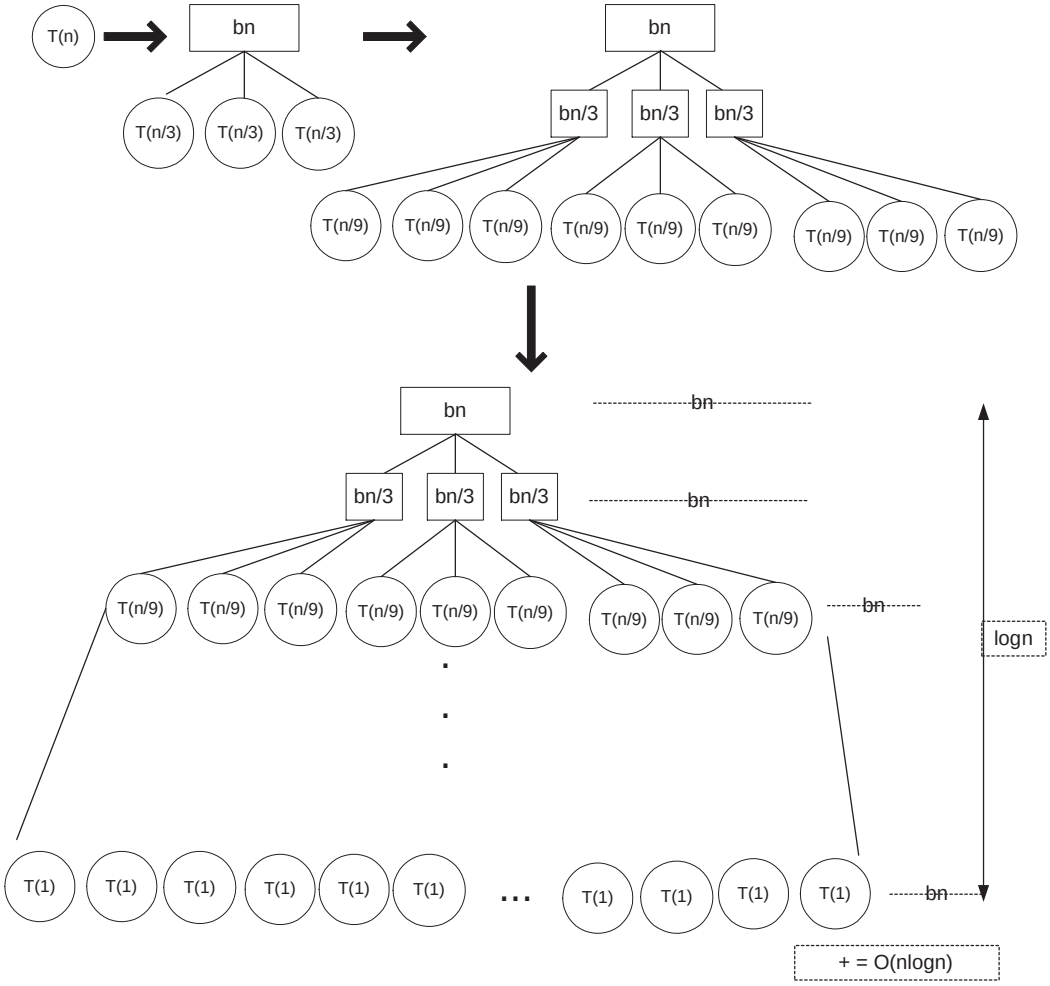
$$t_k \;=\; \frac{2^{3\times 2^k}\;\; 2^{\log 3\times 2^k}}{2^k\;\; 2^2} \;=\; \frac{8^{2^k}\;\; 3^{2^k}}{2^{k+2}}$$

οπότε τελικώς:

$$T(n) \;=\; \frac{2^{3n-2}\;\; 3^n}{n}$$

3.6 Δένδρο Αναδρομής

Αρκετές φορές η οπτικοποίηση μιας σχέσης αναδρομής μέσω ενός **δένδρου αναδρομής** (recursion tree) βοηθά στη κατανόηση του τι συμβαίνει όταν η σχέση επαναλαμβάνεται. Αυτό μπορεί να βοηθήσει στην οργάνωση των πράξεων με τέτοιο τρόπο έτσι ώστε να γίνει εκτίμηση του γενικού όρου.



Σχήμα 3.1: Δένδρο Αναδρομής

Έστω ότι έχουμε την αναδρομική σχέση $T_n = 3T_{n/3} + bn, T_0 = T_1 = T_2 = b$, που εκφράζει το πλήθος των βημάτων που θα εκτελέσει ένας αλγόριθμος

για είσοδο μεγέθους n. Όπως φαίνεται και στο Σχήμα 3.1 για ένα στιγμιότυπο μεγέθους n θα εκτελέσει bn βήματα συν το πλήθος των βημάτων που θα εκτελέσει για να επιλύσει τα τρία υποπροβήματα μεγέθους $\frac{n}{3}$. Κάθε τέτοιο υποπρόβλημα χρειάζεται $\frac{bn}{3}$ βήματα συν τα βήματα για την επίλυση των τριών υποπροβλημάτων μεγέθους $\frac{n}{9}$, κοκ. Με αυτό τον τρόπο σχεδιάζεται το δένδρο αναδρομής, το οποίο θα έχει ύψος $\log_3 n$, αφού σε κάθε επίπεδο τα μεγέθη των προβλημάτων υποτριπλασιάζονται, ενώ για να φθάσουμε σε προβλήματα μεγέθους 1 χρειάζονται $\log_3 n$ επίπεδα. Κάθε κόμβος του δένδρου αυτού αναπαριστά και ένα κόστος βημάτων. Αθροίζοντας όλα τα κόστη σε κάθε επίπεδο παρατηρείται ότι για κάθε επίπεδο το συνολικό κόστος είναι bn. Συνεπώς καταλήγουμε ότι αφού ο αλγόριθμος χρειάζεται bn βήματα για κάθε επίπεδο τότε συνολικά για τα $\log_3 n$ επίπεδα θα χρειασθεί χρόνο $\mathrm{O}(n \log n)$.

3.7 Το Γενικό Θεώρημα

Το γενικό θεώρημα (master theorem) είναι ένας εναλλακτικός τρόπος επίλυσης αναδρομικών εξισώσεων. Αν γίνει κατανοητό στα λεπτά του σημεία, τότε είναι ένας εύκολος τρόπος στη χρήση του. Παραθέτουμε το θεώρημα χωρίς απόδειξη (στηρίζεται στο δένδρο αναδρομής).

Θεώρημα.
Έστω μία αναδρομική εξίσωση τη μορφής:

$$T(n) \;=\; aT(n/b) + f(n)$$

όπου $a, b \geq 1$, ενώ η $f(n)$ είναι μία ασυμπτωτικά θετική συνάρτηση. Τότε:

$$T(n) \;=\; \begin{cases} \Theta(n^{\log_b a}) & \text{αν } f(n) = \mathrm{O}(n^{\log_b a - \epsilon}),\ \epsilon > 0 \\ \Theta(n^{\log_b a} \log n) & \text{αν } f(n) = \Theta(n^{\log_b a}) \\ \Theta(f(n)) & \text{αν } f(n) = \Omega(n^{\log_b a + \epsilon}),\ \epsilon > 0,\ af(\frac{n}{b}) \leq cf(n),\ c < 1 \end{cases}$$

□

Σημειώνεται ότι παρόμοιες αναδρομικές εξισώσεις προκύπτουν κατά την εφαρμογή της αλγοριθμικής μεθόδου Διαίρει και Βασίλευε. Στην περίπτωση αυτή, κατά τη φάση του Διαίρει προκύπτουν υποπίνακες με ακέραιο μέγεθος, δηλαδή υπεισέρχεται η συνάρτηση πάτωμα ή/και η συνάρτηση ταβάνι. Κατά την εφαρμογή του γενικού θεωρήματος αγνοούμε τις συναρτήσεις αυτές καθώς ασυμπτωτικά δεν αλλάζει τίποτε.

Με λίγα λόγια, το γενικό θεώρημα μας προτρέπει να συγκρίνουμε τη συνάρτηση $f(n)$ με την έκφραση $n^{\log_b a}$. Αναλόγως με τη σχέση μεταξύ τους (ποιά είναι

μικρότερη, μεγαλύτερη ή αν τυχόν είναι ίσες), τότε υιοθετούμε την αντίστοιχη εκδοχή. Προσοχή στο εξής σημείο. Δεν αρκεί απλώς μία εκ των δύο να είναι μικρότερη αλλά να είναι *πολυωνυμικά* μικρότερη. Λόγου χάριν, το n^2 είναι πολυωνυμικά μικρότερο από το n^3, όπως και το $\sqrt{n}$ από το n. Όμως το n δεν είναι πολυωνυμικά μικρότερο από το $n \log n$, ούτε μπορούμε να συγκρίνουμε πολυωνυμικά τις εκφράσεις n και $n^{1+\cos n}$. Επομένως, το θεώρημα δεν εφαρμόζεται πάντοτε. Στη συνέχεια θα εξετάσουμε αντίστοιχα παραδείγματα.

Έστω ότι δίνεται η συνάρτηση:

$$T(n) \;=\; 4T(n/2) + n$$

Ισχύει ότι $n^{\log_b a} = n^{\log_2 4} = n^2 > f(n) = n$. Άρα $\epsilon = 1$ και ακολουθούμε το πρώτο σκέλος του θεωρήματος, δηλαδή $T(n) = \Theta(n^2)$.

Έστω ότι δίνεται η συνάρτηση:

$$T(n) \;=\; 4T(n/2) + n^2$$

Ισχύει ότι $n^{\log_b a} = n^{\log_2 4} = n^2 = f(n)$. Άρα ακολουθούμε το δεύτερο σκέλος του θεωρήματος, οπότε $T(n) = \Theta(n^2 \log n)$.

Έστω ότι δίνεται η συνάρτηση:

$$T(n) \;=\; 4T(n/2) + n^3$$

Ισχύει ότι $n^{\log_b a} = n^{\log_2 4} = n^2 < f(n) = n^3$ για $\epsilon = 1$. Άρα ακολουθούμε το τρίτο σκέλος του θεωρήματος. Πρέπει να αποδείξουμε την απαραίτητη συνθήκη του σκέλους αυτού. Έχουμε, λοιπόν, ότι: $af(n/b) = 4(n/2)^3 = \frac{1}{2}n^3$, το οποίο θα πρέπει να είναι μικρότερο από cn^3. Άρα αρκεί να ισχύει $c = 1/2$. Συνεπώς πράγματι μπορούμε να ακολουθήσουμε το τρίτο σκέλος, οπότε προκύπτει ότι $T(n) = \Theta(n^3)$.

Ένα τελικό παράδειγμα όπου δεν μπορεί να εφαρμοσθεί το γενικό θεώρημα. Έστω η συνάρτηση:

$$T(n) \;=\; 4T(n/2) + n^2/\log n$$

Ισχύει ότι $n^{\log_b a} = n^{\log_2 4} = n^2$? $f(n) = n^2/\log n$. Το ερωτηματικό τίθεται στην προηγούμενη σχέση για να δηλώσει ότι αδυνατούμε να συγκρίνουμε πολυωνυμικά τις δύο εκφράσεις (για παράδειγμα, δεν υπάρχει κάποιο ϵ). Έτσι στο σημείο

αυτό δεν μπορεί να υπάρξει πρόοδος γιατί δεν μπορεί να γίνει αναγωγή σε κάποια από τις τρεις κατηγορίες. Βέβαια, η αναδρομική αυτή εξίσωση έχει ήδη λυθεί με τη μέθοδο της επανάληψης.

3.8 Βιβλιογραφική Συζήτηση

Τεχνικές επίλυσης αναδρομικών εξισώσεων βρίσκονται σε πλείστα διδακτικά βιβλία. Μεταξύ άλλων σημειώνονται τα κλασικά του είδους, όπως των Brassard-Bratley [13] και των Cormen-Leiserson-Rivest-Stein [27]. Συμπληρωματικά στοιχεία και ασκήσεις μπορούν να βρεθούν στα βιβλία του Govinda Rao [58], των Graham-Knuth-Patashnik [59], του Liu [90] και του Shaffer [147]. Λεπτομέρειες σχετικά με το γενικό θεώρημα μπορούν να βρεθούν στο άρθρο [9], καθώς και στα βιβλία [27, 28, 169]. Η Άσκηση 13 είναι μία γενίκευση του δημοφιλούς προβλήματος του Josephus, για το οποίο ιστορικά στοιχεία αναφέρονται στα βιβλία [59, 85], ενώ οι Ασκήσεις 14-15 βασίζονται επίσης στο πρόβλημα αυτό.

3.9 Ασκήσεις

1. Από τις επόμενες γραμμικές αναδρομικές εξισώσεις ποιές είναι ομογενείς και ποιές όχι:
   - $t_n + 2t_{n-1} + t_{n-2} = 0$
   - $t_n = {t_{n-1}}^2 + {t_{n-2}}^2$
   - $t_n = \sqrt{t_{n-1} + t_{n-2}}$
   - $t_n = 5t_{n-1} - 6t_{n-3} + t_{n-5}$.
2. Να επιλυθούν οι επόμενες αναδρομικές εξισώσεις με τη μέθοδο της αντικατάστασης.
   - $t_n \ = \ b\ n^2 + n\ t(n-1),\ \ \text{ενώ}\ \ t_1 = a$
   - $t_n \ = \ b\ n^k + n\ t(n-1),\ \ \text{ενώ}\ \ t_1 = a$
   - $t_n \ = \ 7\ n_{n/2} + n^2,\ \ \text{ενώ}\ \ t_1 = a$
   - $t_n \ = \ 7\ n_{n/3} + n^2,\ \ \text{ενώ}\ \ t_1 = t_2 = a$
3. Να επιλυθούν οι επόμενες αναδρομικές εξισώσεις με τη μέθοδο της επανάληψης θεωρώντας όπου χρειασθεί ότι το n είναι δύναμη του 2.
   - $T(n) \ = \ c\ T(n-1),\ \ \text{όπου}\ \ T(0) = 1$
   - $T(n) \ = \ n\ T(n-1),\ \ \text{όπου}\ \ T(0) = 1$

- $T(n) = 1 + T(\lfloor n/2 \rfloor)$, όπου $T(1) = 1$
- $T(n) = 1 + 2\,T(\lfloor n/2 \rfloor)$, όπου $T(1) = 1$
- $T(n) = T(\lfloor n/2 \rfloor) + \sqrt{n}$, όπου $T(1) = 1$
- $T(n) = c\,\lceil \log n \rceil + T(\lceil n/2 \rceil)$, όπου $T(1) = 0$
- $T(n) = \lceil n \log n \rceil + T(\lceil n/2 \rceil)$, όπου $T(1) = 0$.

4. Να επιλυθούν οι εξής ομογενείς γραμμικές αναδρομικές εξισώσεις με τη μέθοδο της χαρακτηριστικής εξίσωσης:

   - $t_k = 4\,t_{k-1} + 5\,t_{k-2}$, όπου $k \geq 2$ και $t_0 = 3, t_1 = 6$
   - $t_k = 3\,t_{k-1} + 4\,t_{k-2}$, όπου $k \geq 2$ και $t_0 = 0, t_1 = 1$
   - $t_k = t_{k-1} + t_{k-2}$, όπου $k \geq 2$ και $t_0 = 0, t_1 = 1$
   - $t_k = 2\,t_{k-1} - 2\,t_{k-2}$, όπου $k \geq 2$ και $t_0 = 0, t_1 = 1$
   - $t_k = 5\,t_{k-1} - 8\,t_{k-2} + 4\,t_{k-3}$, όπου $k \geq 3$ και $t_0 = 0, t_1 = 1, t_2 = 2$.

5. Να επιλυθούν οι εξής μη ομογενείς γραμμικές αναδρομικές εξισώσεις με τη μέθοδο της χαρακτηριστικής εξίσωσης:

   - $t_k = 2\,t_{k-1} + 3^n$, όπου $k \geq 1$
   - $t_k = 2\,t_{k-1} + (n+5)\,3^n$, όπου $k \geq 1$
   - $t_k = 2\,t_{k-1} + 1$, όπου $k \geq 1$ και $t_0 = 0$
   - $t_k = 2\,t_{k-1} + k$, όπου $k \geq 1$
   - $t_k = 2\,t_{k-1} + k + 2^k$, όπου $k \geq 1$ και $t_0 = 0$.

6. Να επιλυθούν οι εξής αναδρομικές εξισώσεις με τη μέθοδο της αλλαγής μεταβλητής θεωρώντας όπου χρειασθεί ότι το n είναι δύναμη του 2:

   - $T(n) = 2\,T(\lfloor \sqrt{n} \rfloor) + 1$, όπου $n > 1$
   - $T(n) = 2\,T(\lfloor \sqrt{n} \rfloor) + \log n$, όπου $n > 1$
   - $T(n) = 4\,T(n/2) + n^2$, όπου $n > 1$
   - $T(n) = 2\,T(n/2) + n \log n$ όπου $n > 1$
   - $T(n) = 3\,T(n/2) + c\,n$, όπου $n > 1$ και c σταθερά.

7. Να επιλυθούν οι εξής αναδρομικές εξισώσεις με τη μέθοδο του γενικού θεωρήματος, θεωρώντας όπου χρειασθεί την κατάλληλη μορφή του n:

   - $T(n) = 2\,T(n/2) + \log n$, όπου $n \geq 2$ και $T(1) = 1$

- $T(n) = 2\,T(\sqrt{n}) + \log n$, όπου $n \geq 4$ και $T(2) = 1$
- $T(n) = 9\,T(n/3) + n$, όπου $n \geq 3$ και $T(1) = 1$
- $T(n) = T(2n/3) + 1$, όπου $n \geq 3$ και $T(1) = 1$
- $T(n) = 3\,T(n/4) + n \log n$, όπου $n \geq 4$ και $T(1) = 1$
- $T(n) = 2\,T(n/2) + n \log n$, όπου $n \geq 2$ και $T(1) = 1$.

8. Να επιλυθούν οι εξής αναδρομικές εξισώσεις θεωρώντας όπου χρειασθεί ότι το n είναι δύναμη του 2:

   - $T(n) = 5\,T(n/2) + (n \log n)^2$, όπου $n \geq 2$ και $T(1) = 1$
   - $T(n) = \frac{3}{2}\,T(n/2) - \frac{1}{2}\,T(n/4) - \frac{1}{n}$, όπου $n \geq 3$ και $T(1)=1, T(2)=\frac{3}{2}$
   - $t_k = t_{k-1} + t_{k-3} - t_{k-4}$, όπου $k \geq 4$ και $t_k = k$ για $0 \leq k \leq 3$
   - $t_k = 1/(4 - t_{k-1})$, όπου $k > 1$ και $t_1 = 1/4$.

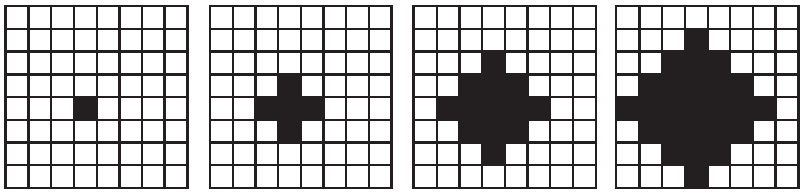
9. Να επιλυθούν οι εξής αναδρομικές εξισώσεις:

   - $t_k = t_{k-1} + 2\,t_{k-2} - 2\,t_{k-3}$, όπου $k \geq 3$ και $t_k = 9k^2 - 15k + 106$ για $0 \leq k \leq 2$
   - $t_k = (1 + t_{k-1})/t_{k-2}$, όπου $t_0 = a$ και $t_1 = b$
   - $2t_k = k\,t_{k-1} + 3\,k!$, όπου $t_0 = 5$
   - $t_k = t_{k-m} + 1$ για $k \geq m$ και $t_k = k$ για $0 \leq k \leq m$

10. Για δύο αλγορίθμους δίνονται οι αντίστοιχες αναδρομικές εξισώσεις: $T_1(n) = 7\,T_1(n/2) + n^2$ και $T_2(n) = a\,T_2(n/4) + n^2$. Για ποιες τιμές του a ο δεύτερος αλγόριθμος είναι ασυμπτωτικά ταχύτερος του πρώτου;

11. Να βρεθεί το πλήθος των τρόπων που μπορούμε να ανεβούμε μία σκάλα με n σκαλοπάτια, αν είναι δυνατόν σε κάθε βήμα να ανεβαίνουμε είτε 1 σκαλοπάτι είτε 2 σκαλοπάτια. Για παράδειγμα, μία σκάλα 3 σκαλοπατιών μπορούμε να την ανεβούμε με 3 τρόπους: 1-1-1, 1-2, 2-1.

12. Σε ένα πίνακα μεγάλου μεγέθους αρχικά τα τετράγωνα είναι κενά. Στην πρώτη κίνηση μαυρίζεται ένα τετράγωνο, όπως φαίνεται στο Σχήμα 3.2. Στη συνέχεια μαυρίζεται το τετράγωνο προς τα επάνω, κάτω, αριστερά και δεξιά. Στην επόμενη κίνηση, για κάθε από τα μαύρα τετράγωνα και πάλι μαυρίζουμε τα τετράγωνα προς τα επάνω, κάτω, αριστερά και δεξιά. Από πόσα μαύρα τετράγωνα αποτελείται το μαύρο σχήμα στη n-οστή κίνηση;



Σχήμα 3.2: Το πρόβλημα με τα μαύρα τετράγωνα.

13. Μία συμμορία από n ληστές στην Άγρια Δύση περικυκλώνεται από το απόσπασμα του σερίφη. Οι ληστές έχουν μόνο ένα άλογο για να διαφύγουν. Έτσι συμφωνούν στην εξής μέθοδο επιλογής του τυχερού που θα πάρει το άλογο και θα σωθεί. Οι ληστές σχηματίζουν ένα κύκλο, ο αρχηγός αντιστοιχίζεται στον αριθμό 1 και η αρίθμηση συνεχίζει ωρολογιακά. Σε ένα καπέλο τοποθετούν n λαχνούς με αριθμούς από 1 ως το n. Ο αρχηγός τραβά ένα λαχνό με τον αριθμό, έστω, i. Αυτό σημαίνει ότι ο i-οστός ληστής πρέπει να εξαιρεθεί. Η διαδικασία των εξαιρέσεων του i-οστού ληστή συνεχίζεται θεωρώντας για αρχή της καινούργιας αρίθμησης τον επόμενο ληστή αυτού που μόλις εξαιρέθηκε. Αυτός που μένει τελευταίος παίρνει το άλογο, όλα τα κλοπιμαία και φεύγει. Να σχεδιασθεί ένας αλγόριθμος υλοποίησης της μεθόδου. Να βρεθεί μέσω αναδρομικής εξίσωσης αυτός που θα αποφύγει την εξαίρεση, δεδομένων των i και n.

14. Σε ένα κύκλο τοποθετούνται $2n$ άτομα. Τα πρώτα n άτομα είναι οι "καλοί", ενώ τα επόμενα n είναι οι "κακοί". Δεδομένου του n, να βρεθεί ένα m τέτοιο ώστε, αν εξαιρούμε από τον κύκλο κάθε m-οστό άτομο, στο τέλος θα μείνουν μόνο οι "καλοί".

15. Σε ένα κύκλο τοποθετούνται 10 άτομα και εξαιρείται από τον κύκλο κάθε m-οστό άτομο (όπου το m οποιοσδήποτε ακέραιος). Να αποδειχθεί ότι δεν είναι δυνατόν τα τρία πρώτα άτομα που θα εξαιρεθούν να είναι ο 10ος, ο k-οστός και ο $(k+1)$-οστός (με τη συγκεκριμένη σειρά), για οποιοδήποτε k.

4

Γεννήτριες Συναρτήσεις

Περιεχόμενα Κεφαλαίου

Έχει λεχθεί ότι οι **γεννήτριες συναρτήσεις** (generating functions) είναι "ένα νήμα όπου κρεμούμε μία ακολουθία αριθμών για επίδειξη". Επί της ουσίας, οι γεννήτριες συναρτήσεις είναι ένας κομψός και συνοπτικός συμβολισμός σειρών δυνάμεων, με τη βοήθεια των οποίων προβλήματα σειρών μετατρέπονται σε προβλήματα συναρτήσεων. Αυτό είναι σημαντικό καθώς η διαχείρισης συναρτήσεων είναι γενικώς ευκολότερη υπόθεση. Το αντικείμενο των συναρτήσεων αυτών είναι τεράστιο και έχουν γραφεί πολλά βιβλία μαθηματικών για αυτό. Στη συνέχεια του κεφαλαίου αυτού θα παρουσιασθούν μερικές πτυχές του αντικειμένου αυτού ώστε εν τέλει με τη βοήθεια των γεννητριών συναρτήσεων: (α) να απαριθμούμε δυνατούς τρόπους εμφάνισης διαφορετικών καταστάσεων, και (β) να επιλύουμε αναδρομικές εξισώσεις .

4.1 Κανονικές Γεννήτριες Συναρτήσεις

Δεδομένης μίας ακολουθίας απείρων όρων $a_0, a_1, a_2, \ldots$, η αντίστοιχη γεννήτρια συνάρτηση είναι μία σειρά δυνάμεων:

$$f(x) \;=\; a_0 + a_1 x + a_2 x^2 + \ldots \;=\; \sum_{n=0}^{\infty} a_n x^n$$

που είναι **τυπική** (formal) με την έννοια ότι ορίζεται αλγεβρικά και όχι αναλυτικά. Η ανωτέρω συνάρτηση ονομάζεται **κανονική** (ordinary) γεννήτρια συνάρτηση και είναι η απλούστερη μορφή γεννήτριας συνάρτησης, καθώς στη βιβλιογραφία αναφέρεται ένα πλήθος άλλων συνθετότερων γεννητριών συναρτήσεων, οι οποίες όμως δεν θα μας απασχολήσουν στα πλαίσια του βιβλίου αυτού. Στη συνέχεια, με τον επόμενο συμβολισμό θα ισοδυναμούμε την αρχική ακολουθία αριθμών με την αντίστοιχη γεννήτρια συνάρτηση:

$$< a_0, a_1, a_2, a_3, \ldots > \quad \leftrightarrow \quad a_0 + a_1 x + a_2 x^2 + a_3 x^3 \ldots$$

Για παράδειγμα, έστω ότι δίνεται η απλή ακολουθία $< 1, 1, 1, 1, \ldots >$, δηλαδή ισχύει $a_n = 1$ για κάθε $n \geq 0$. Η αντίστοιχη γεννήτρια συνάρτηση είναι:

$$f(x) \;=\; 1 + x + x^2 + x^3 + \ldots \;=\; \sum_{n=0}^{\infty} x^n$$

Εύκολα προκύπτει ότι:

$$\begin{aligned} f(x) &= 1 + x + x^2 + x^3 + \ldots \\ &= 1 + x \cdot (1 + x + x^2 + x^3 + \ldots) \\ &= 1 + x \cdot f(x) \Rightarrow \\ f(x) &= \frac{1}{1-x} \end{aligned}$$

Επομένως, δοθείσης της ακολουθίας $a_n = 1$ για κάθε $n \geq 0$, η αντίστοιχη γεννήτρια συνάρτηση είναι $f(x) = 1/(1-x)$. Βέβαια, γνωρίζουμε από τις φθίνουσες γεωμετρικές προόδους, ότι η σχέση αυτή ισχύει μόνο αν $|x| < 1$, αλλά προς το παρόν δεν μας απασχολούν ζητήματα σύγκλισης. Ωστόσο, σημειώνεται το συμπέρασμα ότι με τη βοήθεια ιδιοτήτων της $f(x)$ μπορούμε να εξάγουμε πληροφορία για την αντίστοιχη ακολουθία αριθμών.

Ας εξετάσουμε ένα άλλο απλό παράδειγμα. Δίνεται η ακολουθία αριθμών $< 1, -1, 1, -1, \ldots >$. Η αντίστοιχη γεννήτρια συνάρτηση είναι:

$$f(x) \;=\; 1 - x + x^2 - x^3 + \ldots \;=\; \sum_{n=0}^{\infty} (-1)^n x^n$$

Με την ίδια τεχνική όπως προηγουμένως, προκύπτει ότι η αντίστοιχη γεννήτρια συνάρτηση είναι $f(x) = 1/(1+x)$, δηλαδή ισχύει:

$$< 1, -1, 1, -1, \ldots > \quad \leftrightarrow \quad \frac{1}{1+x}$$

Διαισθητικά, λοιπόν, καταλαβαίνουμε ότι οι γεννήτριες συναρτήσεις ονομάζονται έτσι γιατί βοηθούν να γεννήσουμε τους συντελεστές της σειράς δυνάμεων.

4.2 Πράξεις σε Γεννήτριες Συναρτήσεις

Στη συνέχεια εξετάζεται μία σειρά ιδιοτήτων. Οι ιδιότητες αυτές είναι εύκολο να αποδειχθούν από τον αναγνώστη.

Πολλαπλασιασμός
Αν πολλαπλασιάσουμε μία γεννήτρια συνάρτηση επί μία σταθερά, τότε κάθε όρος της αντίστοιχης ακολουθίας πολλαπλασιάζεται επί την αντίστοιχη σταθερά. Με βάση, λοιπόν, τον ανωτέρω κανόνα, εφόσον ισχύει:

$$< 1, 1, 1, 1, \ldots > \quad \leftrightarrow \quad \frac{1}{1-x}$$

έπεται ότι ισχύει:

$$< 2, 2, 2, 2, \ldots > \quad \leftrightarrow \quad \frac{2}{1-x}$$

Πρόσθεση
Πρόσθεση δύο γεννητριών συναρτήσεων σημαίνει ότι πρέπει να προστεθούν οι αντίστοιχοι όροι των δύο ακολουθιών. Για παράδειγμα, λοιπόν, με βάση αυτόν τον κανόνα, εφόσον ισχύει ότι:

$$< 1, 1, 1, 1, \ldots > \quad \leftrightarrow \quad \frac{1}{1-x}$$

και

$$< 1, -1, 1, -1, \ldots > \quad \leftrightarrow \quad \frac{1}{1+x}$$

έπεται ότι ισχύει:

$$< 2, 0, 2, 0, \ldots > \quad \leftrightarrow \quad \frac{1}{1-x} + \frac{1}{1+x} = \frac{2}{1-x^2}$$

Δεξιά Ολίσθηση
Αν ισχύει ότι:

$$< a_0, a_1, a_2, a_3, \ldots > \quad \leftrightarrow \quad f(x)$$

τότε αν προστεθούν στην αρχή της ακολουθίας k μηδενικά, έπεται ότι θα ισχύει:

$$< 0, 0, \ldots, 0, 0, a_0, a_1, a_2, a_3, \ldots > \quad \leftrightarrow \quad x^k \cdot f(x)$$

Για παράδειγμα, αν προστεθούν k μηδενικά εμπρός από τη γνωστή ακολουθία $< 1, 1, 1, 1, \ldots >$, τότε η αντίστοιχη γεννήτρια συνάρτηση είναι $x^k/(1-x)$.

Παραγώγιση
Αν παραγωγισθεί το ένα σκέλος, τότε πρέπει να παραγωγισθεί και το άλλο. Για παράδειγμα, από τη γνωστή ακολουθία προκύπτει:

$$\begin{aligned} \frac{d}{dx}\,\frac{1}{1-x} &= \frac{d}{dx}\,(1 + x + x^2 + x^3 + \ldots) \\ \frac{1}{(1-x)^2} &= 1 + 2x + 3x^2 + 4x^3 + \ldots \\ \frac{1}{(1-x)^2} &= \sum_{n=0}^{\infty} (n+1)x^n \end{aligned}$$

Από την τελευταία σχέση προκύπτει ότι η γεννήτρια συνάρτηση $f(x) = 1/(1-x)^2$ αντιστοιχεί στην ακολουθία $< 1, 2, 3, 4, \ldots >$, ή αλλιώς ότι ισχύει γενικώς $a_n = n$ για κάθε $n \geq 0$.

Γενικότερα, μπορεί να αποδειχθεί ότι δεδομένης της αντιστοιχίας:

$$< a_0, a_1, a_2, a_3, \ldots > \quad \leftrightarrow \quad f(x)$$

ισχύει επίσης:

$$< a_1, 2a_2, 3a_3, 4a_4, \ldots > \quad \leftrightarrow \quad f'(x)$$

Δηλαδή, παρατηρούμε ότι με τον κανόνα αυτό εκτελείται ταυτόχρονα: (α) μία αριστερή ολίσθηση, και (β) ένας πολλαπλασιασμός κάθε όρου με το δείκτη του. Αν σε κάποια περίπτωση χρειαζόταν μόνο ο πολλαπλασιασμός, τότε θα μπορούσε να ακυρωθεί η αριστερή ολίσθηση εφαρμόζοντας και μία δεξιά ολίσθηση. Έστω λοιπόν ότι πρέπει να βρεθεί η γεννήτρια συνάρτηση της ακολουθίας $< 0, 1, 4, 9, \ldots >$. Ισχύει κατά σειρά:

$$\begin{aligned}
< 1, 1, 1, 1, \ldots > \quad &\leftrightarrow \quad \frac{1}{1-x} \\
< 1, 2, 3, 4, \ldots > \quad &\leftrightarrow \quad \frac{d}{dx}\frac{1}{1-x} = \frac{1}{(1-x)^2} \\
< 0, 1, 2, 3, \ldots > \quad &\leftrightarrow \quad \frac{x}{(1-x)^2} \\
< 1, 4, 9, 16, \ldots > \quad &\leftrightarrow \quad \frac{d}{dx}\frac{x}{(1-x)^2} = \frac{1+x}{(1-x)^3} \\
< 0, 1, 4, 9, \ldots > \quad &\leftrightarrow \quad \frac{x(1+x)}{(1-x)^3}
\end{aligned}$$

4.3 Ακολουθία Fibonacci

Στο προηγούμενο κεφάλαιο δόθηκε ορισμός της ακολουθίας Fibonacci δεύτερης τάξης. Στο σημείο αυτό θα συνδεθεί η έννοια αυτή με τις γεννήτριες συναρτήσεις. Συγκεκριμένα θα αποδειχθεί ότι:

$$< 0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots > \quad \leftrightarrow \quad \frac{x}{1-x-x^2}$$

Με τη βοήθεια αυτής της σχέσης, στη συνέχεια θα βρεθεί ο κλειστός τύπος του n-οστού αριθμού Fibonacci, ενώ επίσης οι τεχνικές για την εύρεση αυτού του κλειστού τύπου θα εφαρμοσθούν και σε άλλες περιπτώσεις.

Ορίζουμε, λοιπόν, τη συνάρτηση:

$$f(x) \;=\; f_0 + f_1 x + f_2 x^2 + f_3 x^3 + f_4 x^4 + \ldots$$

Είναι ευνόητο ότι ισχύουν οι σχέσεις:

$$\begin{aligned} < 0, 1, 0, 0, 0, \ldots > \quad &\leftrightarrow \quad x \\ < 0, f_0, f_1, f_2, f_3, \ldots > \quad &\leftrightarrow \quad x f(x) \\ < 0, 0, f_0, f_1, f_2, \ldots > \quad &\leftrightarrow \quad x^2 f(x) \end{aligned}$$

Αθροίζοντας τις σχέσεις αυτές κατά μέλη, προκύπτει:

$$< 0, 1 + f_0, f_1 + f_0, f_2 + f_1, f_3 + f_2, \ldots > \quad \leftrightarrow \quad x + x f(x) + x^2 f(x)$$

Το αριστερό σκέλος της σχέσης αυτής ταυτίζεται με το αριστερό σκέλος της αποδεικτέας σχέσης. Συνεπώς, πρέπει να αποδειχθεί και η ισότητα των δύο δεξιών σκελών. Όμως το δεξιό σκέλος της τελευταίας σχέσης ισούται με $f(x)$, δηλαδή ισχύει:

$$\begin{aligned} f(x) \;&=\; x + x\ f(x) + x^2\ f(x) \quad \Rightarrow \\ f(x) \;&=\; \frac{x}{1 - x - x^2} \end{aligned}$$

Δεδομένης, λοιπόν, της γεννήτριας συνάρτησης για την ακολουθία Fibonacci, θα προσπαθήσουμε να εξάγουμε ένα κλειστό τύπο για το n-οστό αριθμό Fibonacci. Αρχικά εκτελούμε μία παραγοντοποίηση του παρονομαστή της γεννήτριας συνάρτησης:

$$1 - x - x^2 \;=\; (1 - c_1\ x)(1 - c_2\ x)$$

όπου οι σταθερές c_1, c_2 ισούνται με:

$$c_1 = \frac{1 + \sqrt{5}}{2} \qquad c_2 = \frac{1 - \sqrt{5}}{2}$$

Στη συνέχεια για τη διάσπαση του κλάσματος της γεννήτριας συνάρτησης γράφουμε τη σχέση ως εξής:

$$\frac{x}{1 - x - x^2} \;=\; \frac{c_3}{1 - c_1\ x} + \frac{c_4}{1 - c_2\ x}$$

Αν δώσουμε δύο τυχαίες τιμές στη μεταβλητή x, λαμβάνουμε δύο γραμμικές εξισώσεις με αγνώστους τα c_3, c_4. Λύνοντας το παραγόμενο γραμμικό σύστημα με δύο αγνώστους και δύο εξισώσεις προκύπτει:

$$c_3 = \frac{1}{\sqrt{5}} \qquad c_4 = -\frac{1}{\sqrt{5}}$$

Συνεπώς ισχύει:

$$\frac{x}{1-x-x^2} = \frac{1}{\sqrt{5}}\left(\frac{1}{1-c_1x} - \frac{1}{1-c_2x}\right)$$

Όμως, για κάθε όρο της ανωτέρω παρένθεσης ισχύει:

$$\begin{aligned}\frac{1}{1-c_1x} &= 1 + c_1x + c_1^2x^2 + \ldots \\ \frac{1}{1-c_2x} &= 1 + c_2x + c_2^2x^2 + \ldots\end{aligned}$$

Αντικαθιστώντας τις τιμές αυτές προκύπτει:

$$\begin{aligned}f(x) &= \frac{1}{\sqrt{5}}\left(\frac{1}{1-c_1x} - \frac{1}{1-c_2x}\right) \\ &= \frac{(1 + c_1x + c_1^2x^2 + \ldots) - (1 + c_2x + c_2^2x^2 + \ldots)}{\sqrt{5}} \Rightarrow \\ f_n &= \frac{c_1^n - c_2^n}{\sqrt{5}} = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)\end{aligned}$$

4.4 Γεννήτριες Συναρτήσεις για Απαρίθμηση

Είναι γνωστά τα κλασικά αναπτύγματα των $(a+b)^2$ και $(a+b)^3$. Γενικεύοντας ισχύει:

$$(a+b)^n = a^n + \binom{n}{1}a^{n-1}b + \binom{n}{2}a^{n-2}b^2 + \ldots + \binom{n}{n-1}ab^{n-1} + b^n$$

Αντικαθιστώντας $a = 1$ και $b = x$ προκύπτει:

$$(1+x)^n = 1 + \binom{n}{1}x + \binom{n}{2}x^2 + \ldots + \binom{n}{n-1}x^{n-1} + x^n = \sum_{i=0}^{n}\binom{n}{i}x^i$$

Προκύπτει, λοιπόν, ότι η γεννήτρια συνάρτηση $(1+x)^n$ μπορεί να χρησιμεύσει για την εύρεση του πλήθους των συνδυασμών των n αντικειμένων λαμβανόμενων ανά k, για $0 \leq k \leq n$. Δηλαδή, ο συντελεστής του όρου x^k είναι το πλήθος των συνδυασμών που μπορούμε να επιλέξουμε k αντικείμενα από ένα σύνολο n αντικειμένων. Για το λόγο αυτό, μία γεννήτρια συνάρτηση που υπολογίζει το πλήθος συνδυασμών ή διατάξεων ονομάζεται **απαριθμήτρια** (enumerator).

Ότι η γεννήτρια συνάρτηση $(1+x)^n$ πράγματι εκτελεί την απαρίθμηση των συνδυασμών μπορεί να εξηγηθεί ως εξής. Έστω ένα μονομελές σύνολο $\{a_1\}$. Υπάρχει 1 τρόπος να επιλεγούν 0 αντικείμενα, 1 τρόπος να επιλεγεί 1 αντικείμενο και 0 τρόποι να επιλεγούν περισσότερα αντικείμενα από το σύνολο αυτό. Άρα, για την περίπτωση αυτή, η γεννήτρια συνάρτηση είναι $(1+x)$. Δοθέντος ενός άλλου μονομελούς συνόλου $\{a_2\}$, ισχύει το ίδιο σκεπτικό ως προς τη γεννήτρια συνάρτηση. Είναι ευνόητο ότι η γεννήτρια συνάρτηση για την επιλογή από την ένωση ανεξαρτήτων συνόλων είναι το γινόμενο των αντίστοιχων γεννητριών συναρτήσεων. Επομένως, η γεννήτρια συνάρτηση για την επιλογή αντικειμένων από το διμελές σύνολο $\{a_1, a_2\}$ είναι $(1+x)^2$. Προφανώς, υπάρχει 1 τρόπος να επιλεγούν 0 αντικείμενα, 2 τρόποι να επιλεγεί 1 αντικείμενο, 1 τρόπος να επιλεγούν 2 αντικείμενα και 0 τρόποι να επιλεγούν περισσότερα αντικείμενα από το σύνολο αυτό. Αυτό το σκεπτικό μπορεί να γενικευθεί.

Συνέλιξη
Έστω $A(x)$ η γεννήτρια συνάρτηση για την απαρίθμηση επιλογών από ένα σύνολο A, και $B(x)$ η γεννήτρια συνάρτηση για την απαρίθμηση επιλογών από ένα σύνολο B. Αν τα σύνολα A και B είναι ανεξάρτητα μεταξύ τους, τότε η γεννήτρια συνάρτηση για την απαρίθμηση επιλογών από το σύνολο $A \cup B$ είναι $A(x) \cdot B(x)$. Πιο συγκεκριμένα, αν

$$A(x) \;=\; \sum_{n=0}^{\infty} a_n x^n \qquad \text{και} \qquad B(x) \;=\; \sum_{n=0}^{\infty} b_n x^n$$

τότε:

$$C(x) \;=\; A(x) \cdot B(x) \;=\; \sum_{n=0}^{\infty} c_n x^n$$

όπου:

$$c_n \;=\; a_0 b_0 + a_1 b_{n-1} + a_2 b_{n-2} + \ldots + a_n b_0$$

ενώ η ακολουθία $c_0, c_1, c_2, \ldots$ είναι η **συνέλιξη** (convolution) των ακολουθιών $a_0, a_1, a_2, \ldots$ και $b_0, b_1, b_2, \ldots$. Πράγματι το c_n δίνει το πλήθος των επιλογών n αντικειμένων από το σύνολο $A \cup B$. Γενικώς, μπορούμε να επιλέξουμε n αντικείμενα από την ένωση των συνόλων, επιλέγοντας j αντικείμενα από το σύνολο A και $n-j$ από το B. Αυτό μπορεί να γίνει κατά $a_j b_{n-j}$ τρόπους. Προσθέτοντας για όλες τις δυνατές τιμές του j, προκύπτει το ζητούμενο.

Μέχρι στιγμής θεωρήθηκε ότι η επιλογή ενός συγκεκριμένου αντικειμένου από ένα σύνολο μπορεί να γίνει μόνο μία φορά, δηλαδή πρόκειται για επιλογή

χωρίς αντικατάσταση. Ας θεωρήσουμε την περίπτωση της επιλογής με αντικατάσταση. Έστω, λοιπόν, ένα μονομελές σύνολο. Από το σύνολο αυτό μπορούμε να επιλέξουμε 0 αντικείμενα κατά 1 τρόπο, 1 αντικείμενο κατά 1 τρόπο, 2 αντικείμενα κατά 1 τρόπο κοκ. Συνεπώς, η γεννήτρια συνάρτηση για την επιλογή n αντικειμένων με αντικατάσταση από ένα μονομελές σύνολο είναι:

$$< 1, 1, 1, 1, \ldots > \quad \leftrightarrow \quad 1 + x + x^2 + x^3 + \ldots \; = \; \frac{1}{1-x}$$

Με βάση τον κανόνα της συνέλιξης, η γεννήτρια συνάρτηση για την επιλογή n αντικειμένων από την ένωση ανεξαρτήτων συνόλων δίνεται από γινόμενο των αντίστοιχων γεννητριών συναρτήσεων. Επομένως, προκύπτει ότι η ζητούμενη γεννήτρια συνάρτηση είναι $f(x) = 1/(1-x)^n$. Το ερώτημα είναι ποιά είναι η τιμή κάποιου συντελεστή της σειράς των δυνάμεων. Για την εύρεση αυτού του κλειστού τύπου θα χρησιμοποιηθεί το Θεώρημα του Taylor. Ισχύει, λοιπόν, ότι:

$$f(x) \; = \; f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 + \frac{f'''(0)}{3!}x^3 + \ldots + \frac{f^{(k)}(0)}{k!}x^k + \ldots$$

Το θεώρημα λέει ότι η τιμή του k-οστού συντελεστή της γεννήτριας συνάρτησης $1/(1-x)^n$ δίνεται από την τιμή της k-οστής παραγώγου στο 0, διαιρεμένη δια $k!$. Επομένως, κατά σειρά έχουμε:

$$\begin{aligned} f'(x) &= n(1-x)^{-n-1} \\ f''(x) &= n(n+1)(1-x)^{-n-2} \\ f'''(x) &= n(n+1)(n+2)(1-x)^{-n-3} \\ f^{(k)}(x) &= n(n+1)(n+2)\cdots(n+k-1)(1-x)^{-n-k} \end{aligned}$$

Επομένως, η τιμή του ζητούμενου k-οστού συντελεστή της γεννήτριας συνάρτησης είναι:

$$\begin{aligned} f^{(k)}(0)/k! &= \frac{n(n+1)\cdots(n+k-1)}{k!} \\ &= \frac{(n+k-1)!}{(n-1)!\;k!} \; = \; \binom{n+k-1}{k} \end{aligned}$$

Ο τύπος αυτός είχε αναφερθεί στο Κεφάλαιο 2. Στο σημείο αυτό συνοψίζουμε για ευκολία μερικά σημαντικά μέχρι στιγμής ευρεθέντα στον Πίνακα 4.4.

Ακολουθία	Γεννήτρια συνάρτηση
$<1,1,1,1,\ldots>$	$\frac{1}{1-x}$
$<1,-1,1,-1,\ldots>$	$\frac{1}{1+x}$
$<2,2,2,2,\ldots>$	$\frac{2}{1-x}$
$<2,0,2,0,\ldots>$	$\frac{2}{(1-x)^2}$
$<1,2,3,4,\ldots>$	$\frac{1}{(1-x)^2}$
$<0,1,2,3,\ldots>$	$\frac{x}{(1-x)^2}$
$<1,4,9,16,\ldots>$	$\frac{1+x}{(1-x)^3}$
$<0,1,4,9,\ldots>$	$\frac{x(1+x)}{(1-x)^3}$
$<0,1,1,2,3,5,8,\ldots>$	$\frac{x}{1-x-x^2}$

Πίνακας 4.1: Γεννήτριες συναρτήσεις ευρείας χρήσης

4.5 Γεννήτριες Συναρτήσεις και Αναδρομικές Εξισώσεις

Στην επίλυση αναδρομικών εξισώσεων έχει εξ ολοκλήρου αφιερωθεί το Κεφάλαιο 3. Στο σημείο αυτό θα παρουσιασθεί ένας ακόμη τρόπος επίλυσης αναδρομικών εξισώσεων, ο οποίος στηρίζεται στην προηγούμενη θεωρία των γεννητριών συναρτήσεων.

Αρχικά ας εξετάσουμε μία απλή περίπτωση. Δίνεται, λοιπόν, η αναδρομική εξίσωση: $t_k = 3t_{k-1}$. Έστω ότι $f(x)$ είναι η ζητούμενη γεννήτρια συνάρτηση. Διαδοχικά έχουμε:

$$\begin{aligned} f(x) &= a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + \ldots \\ 3x\, f(x) &= 3a_0x + 3a_1x^2 + 3a_2x^3 + \ldots \\ f(x) - 3xf(x) &= a_0 + (a_1 - 3a_0)x + (a_2 - 3a_1)x^2 + \ldots \end{aligned}$$

Όμως οι παρενθέσεις της τελευταίας σχέσης ισούνται με 0. Επομένως, ισχύει:

$$\begin{aligned} f(x) - 3xf(x) &= a_0 \quad \Rightarrow \\ f(x) &= \frac{a_0}{1-3x} = a_0(1 + 3x + 3^2x^2 + 3^3x^3 + \ldots) \end{aligned}$$

Συνεπώς ισχύει $t_k = 3^k a_0$, όπου βέβαια a_0 είναι η αρχική συνθήκη.

Ας εξετάσουμε τώρα μία συνθετότερη αναδρομική εξίσωση:

$$t_k = 2t_{k-1} + t_{k-2}$$

όπου $k \geq 2$, ενώ επίσης ισχύει $t_0 = t_1 = 1$. Η προκύπτουσα ακολουθία είναι: 1, 1, 5, 13, 41, 121, 365, ... Για την εύρεση κλειστού τύπου για το t_k θεωρούμε τη γεννήτρια συνάρτηση $f(x)$ όπου:

$$f(x) \;=\; 1 + x + 5x^2 + 13x^3 + 41x^4 + 121x^5 + 365x^6 + \ldots$$

Και πάλι με μία σειρά αλγεβρικών χειρισμών μπορούμε να βρούμε τον κλειστό τύπο της γεννήτριας συνάρτησης και επομένως τον τύπο κάθε συντελεστή:

$$\begin{array}{rcl} f(x) & = & a_0 \;+\; a_1 x \;+\; a_2 x^2 \;+ a_3 x^3 \;+\; a_4 x^4 + \ldots \\ 2x\, f(x) & = & 2a_0 x \;+\; 2a_1 x^2 \;+\; 2a_2 x^3 \;+\; \ldots \\ 3x^2\, f(x) & = & 3a_0 x^2 \;+\; 3a_1 x^3 \;+\; \ldots \end{array}$$

Αφαιρώντας κατά μέλη έχουμε:

$$f(x) - 2xf(x) - 3x^2 f(x) \;=\; a_0 + (a_1 - 2a_0)x + (a_2 - 2a_1 - 3a_0)x^2 + (a_3 - 2a_2 - 3a_1)x^3 + \ldots$$

Όμως εξ ορισμού είναι γνωστό ότι $a_n - 2a_{n-1} - 3a_{n-2} = 0$ και επομένως ισχύει:

$$\begin{array}{rcll} f(x) - 2x\, f(x) - 3x^2\, f(x) & = & a_0 + (a_1 - 2a_0)x & \Rightarrow \\ f(x) & = & \dfrac{1-x}{1 - 2x - 3x^2} & \\ & = & \dfrac{1-x}{(1-3x)(1+x)} & \\ & = & \dfrac{1}{2}\,\dfrac{1}{1-3x} + \dfrac{1}{2}\,\dfrac{1}{1+x} & \end{array}$$

Στην τελευταία σχέση διαπιστώνουμε γνωστούς κλειστούς τύπους. Έτσι καταλήγουμε ότι:

$$t_n \;=\; \frac{1}{2}\, 3^n + \frac{1}{2}\, (-1)^n$$

Ας εξετάσουμε ένα τελευταίο αλλά συνθετότερο παράδειγμα. Δίνονται οι αναδρομικές εξισώσεις:

$$\begin{array}{rcl} a_n & = & 2a_{n-1} + b_{n-1} \\ b_n & = & a_{n-1} + 2b_{n-1} \end{array}$$

όπου $a_0 = 1$ και $b_0 = 0$. Παρατηρούμε, όμως, ότι οι δύο εξισώσεις αυτές είναι "διαπλεκόμενες", γεγονός που καθιστά την επίλυσή τους δυσκολότερη.

Αρχικά, λοιπόν, δεχόμαστε ότι ισχύει:

$$A(x) = \sum_{n=0}^{\infty} a_n x^n \quad \text{και} \quad B(x) = \sum_{n=0}^{\infty} b_n x^n$$

Πολλαπλασιάζουμε τις αρχικές αναδρομικές εξισώσεις επί x^n, οπότε προκύπτει:

$$\begin{aligned} a_n x^n &= 2a_{n-1}x^n + b_{n-1}x^n \\ b_n x^n &= a_{n-1}x^n + 2b_{n-1}x^n \end{aligned}$$

Λόγω της ύπαρξης του δείκτη n–1, στις επόμενες σχέσεις το άθροισμα αρχίζει από 1 και όχι από 0. Έχουμε λοιπόν:

$$\begin{aligned} \sum_{n=1}^{\infty} a_n x^n &= \sum_{n=1}^{\infty} 2a_{n-1}x^n + \sum_{n=1}^{\infty} b_{n-1}x^n \\ \sum_{n=1}^{\infty} b_n x^n &= \sum_{n=1}^{\infty} a_{n-1}x^n + \sum_{n=1}^{\infty} 2b_{n-1}x^n \end{aligned}$$

Συνεπώς ισχύει:

$$\begin{aligned} A(x) - a_0 x^0 &= 2x \sum_{n=1}^{\infty} 2a_{n-1}x^{n-1} + x \sum_{n=1}^{\infty} b_{n-1}x^{n-1} \\ B(x) - b_0 x^0 &= x \sum_{n=1}^{\infty} a_{n-1}x^{n-1} + 2x \sum_{n=1}^{\infty} 2b_{n-1}x^{n-1} \end{aligned}$$

ή ισοδύναμα:

$$\begin{aligned} A(x) - 1 &= 2x \sum_{n=0}^{\infty} 2a_{n-1}x^{n-1} + x \sum_{n=0}^{\infty} b_{n-1}x^{n-1} \\ B(x) - 0 &= x \sum_{n=0}^{\infty} a_{n-1}x^{n-1} + 2x \sum_{n=0}^{\infty} 2b_{n-1}x^{n-1} \end{aligned}$$

και

$$\begin{aligned} A(x) - 1 &= 2xA(x) + xB(x) \\ B(x) - 0 &= xA(x) + 2xB(x) \end{aligned}$$

Φθάσαμε σε ένα γραμμικό σύστημα 2 εξισώσεων με 2 αγνώστους, το οποίο επιλύεται και δίνει:

$$\begin{aligned} A(x) &= \frac{1-2x}{1-4x+3x^2} = \frac{1}{2}\,\frac{1}{1-x} + \frac{1}{2}\,\frac{1}{1-3x} \\ B(x) &= \frac{x}{1-4x+3x^2} = -\frac{1}{2}\,\frac{1}{1-x} + \frac{1}{2}\,\frac{1}{1-3x} \end{aligned}$$

Επομένως, από αυτές τις γεννήτριες συναρτήσεις προκύπτει ότι οι σειρές δυνάμεων είναι:

$$A(x) = \sum_{n=0}^{\infty} \frac{1+3^n}{2}\, x^n \quad \text{και} \quad B(x) = \sum_{n=0}^{\infty} \frac{3^n-1}{2}\, x^n$$

4.6 Βιβλιογραφική Συζήτηση

Το αντικείμενο των γεννητριών συναρτήσεων είναι τεράστιο. Οι απαρχές του πηγαίνουν πίσω στον De Moivre. Στο κεφάλαιο αυτό εξετάσθηκαν οι κανονικές γεννήτριες συναρτήσεις. Στη βιβλιογραφία αναφέρεται ένα πλήθος άλλων τύπων γεννητριών συναρτήσεων, όπως οι εκθετικές, οι πιθανοτικές, οι σειρές Lambert, οι σειρές Bell, οι σειρές Dirichlet και άλλες. Το βιβλίο του Wilf διαπραγματεύεται αποκλειστικά το αντικείμενο αυτό [171]. Μάλιστα το βιβλίο είναι διαθέσιμο στο Διαδίκτυο και περιέχει λυμένες και άλυτες ασκήσεις. Στα βιβλία των Govinda Rao [58], Liu [90], Bryant [18], Graham-Knuth-Patashnik [59] και Hofri [69] υπάρχουν εκτενή αντίστοιχα κεφάλαια (η σειρά των βιβλίων αντιστοιχεί σε αυξανόμενο βαθμό δυσκολίας). Η Άσκηση 6 προέρχεται από το άρθρο [50], ενώ οι Ασκήσεις 9-10 από το άρθρο [52].

4.7 Ασκήσεις

1. Να βρεθούν οι γεννήτριες συναρτήσεις των ακολουθιών:
   - $< 1, 2, 4, 8, \ldots >$, και
   - $< 1, 8, 27, 64, \ldots >$.
2. Δίνεται η ακολουθία: $< 2, 4, 6, 8, \ldots >$. Θα ανέμενονταν οι επόμενες τιμές της ακολουθίας να είναι: 10, 12 κοκ. Ποιά είναι η γεννήτρια συνάρτηση που παράγει αυτή την ακολουθία; Να βρεθούν άλλες γεννήτριες συναρτήσεις που να δίνουν ακολουθίες με τους ίδιους 4 πρώτους όρους αλλά η συνέχεια να είναι διαφορετική.

3. Να βρεθεί η συνάρτηση για τους συντελεστές του x^n των γεννητριών συναρτήσεων:

   - $f(x) = \frac{x^3}{1-x^2}$, και
   - $f(x) = \frac{x^2-1}{1+3x^3}$.

4. Ποιοί είναι οι συντελεστές των δυνάμεων του x που αντιστοιχούν στη γεννήτρια συνάρτηση:

   - $f(x) = \frac{1}{1-2x^3}$, και
   - $f(x) = \frac{1}{1-2x}$.

5. Κατά πόσους τρόπους μπορεί:

   - να τοποθετηθούν n πράσινα αντικείμενα σε ένα δοχείο;
   - να επιλεγεί ένα αντικείμενο από ένα δοχείο με n διαφορετικά χρωματιστά αντικείμενα;

6. Δύο παίκτες Π1 και Π2 παίζουν τάβλι και το τελικό αποτέλεσμα είναι Σ1:Σ2. Να υπολογισθεί ο αριθμός των διαφορετικών τρόπων που μπορούμε να φθάσουμε στο αποτέλεσμα αυτό. Για παράδειγμα, αν το τελικό αποτέλεσμα είναι 2:1, τότε υπάρχουν 3 διαφορετικοί τρόποι: (α) 0:1, 1:1, 2:1, (β) 1:0, 1:1, 2:1 και (γ) 1:0, 2:0, 2:1.

7. Χρησιμοποιώντας τον κανόνα της συνέλιξης να βρεθεί με πόσους τρόπους μπορούν να τοποθετηθούν n φρούτα σε ένα δοχείο υπό τους εξής περιορισμούς:

   - το πλήθος των μήλων να είναι άρτιο,
   - το πλήθος των μπανανών να είναι πολλαπλάσιο του 3,
   - το πλήθος των πορτοκαλιών να είναι τουλάχιστον 4, και
   - το πλήθος των αχλαδιών να είναι το μέγιστο 1.

8. Κατά πόσους τρόπους μπορούμε να δώσουμε $n \geq 3$ ευρώ σε 3 άτομα:

   - με τον περιορισμό ότι κάθε άτομο θα πάρει τουλάχιστον 1 ευρώ;
   - χωρίς κάποιον περιορισμό;

   Πόσες λύσεις έχει η εξίσωση $x + y + z = n$;

9. Με πόσους τρόπους μπορεί να καλυφθεί ένα ορθογώνιο 2xn με κομμάτια ντόμινο 2x1; Η άσκηση να επιλυθεί δοκιμάζοντας διαδοχικά για τιμές $n = 0, 1, 2, \ldots$. Τι παρατηρείτε;

10. Με πόσους τρόπους μπορεί να καλυφθεί ένα ορθογώνιο 3xn με κομμάτια ντόμινο 2x1; Επίσης, με πόσους τρόπους μπορεί να καλυφθεί ένα παραλληλεπίπεδο 2x2xn με τούβλα 2x1x1;

11. Να βρεθεί η γεννήτρια συνάρτηση που παράγει την ίδια ακολουθία με την έξοδο των επομένων ψευδικωδικών:

```
x <-- 3; i <--1;
while i<=5 do
   print(x);
   x <-- 2x-2;
   i <-- 1+1;

x <-- 1; i <--1;
while i<=5 do
   print(x);
   x <-- 5x-6;
   i <-- 1+1;

x <-- 6; i <--1;
while i<=5 do
   print(x);
   x <-- trunc(x/3)+6;
   i <-- 1+1;
```

12. Δίνεται η αναδρομική εξίσωση:

$$t_k \;=\; t_{k-1} + t_{k-2} - t_{k-3}$$

όπου $k \geq 3$, ενώ επίσης ισχύει $t_0 = t_1 = t_2 = 1$. Με τη βοήθεια γεννήτριας συνάρτησης να βρεθεί κλειστός τύπος για το t_k.

13. Δίνεται η αναδρομική εξίσωση:

$$t_k \;=\; 2t_{k-1} - 1$$

όπου $k \geq 1$, ενώ επίσης ισχύει $t_0 = 1$. Με τη βοήθεια γεννήτριας συνάρτησης να βρεθεί κλειστός τύπος για το t_k.

14. Δίνεται η αναδρομική εξίσωση:

$$3t_k = -4t_{k-1} + t_{k-2}$$

όπου $k \geq 2$, ενώ επίσης ισχύει $t_0 = t_1 = 1$. Με τη βοήθεια γεννήτριας συνάρτησης να βρεθεί κλειστός τύπος για το t_k.

5

Βασικοί Αλγόριθμοι

Περιεχόμενα Κεφαλαίου

Στο κεφάλαιο αυτό θα εξετάσουμε μερικούς γνωστούς αλγορίθμους και δομές δεδομένων και θα χρησιμοποιήσουμε τα εργαλεία που παρουσιάσθηκαν στα προηγούμενα κεφάλαια, αλλά θα γνωρίσουμε και αλγορίθμους των οποίων η ανάλυση είναι ad hoc και δεν στηρίζεται στις προηγούμενες τεχνικές. Πιο συγκεκριμένα θα εξετασθούν εναλλακτικοί αλγόριθμοι για τον υπολογισμό του μέγιστου κοινού διαιρέτη και των αριθμών Fibonacci, για το πρόβλημα των πύργων του Hanoi, καθώς και για τον υπολογισμό της δύναμης, των συνδυασμών και του πολλαπλασιασμού πινάκων.

5.1 Εύρεση Μέγιστου Κοινού Διαιρέτη

Ας ξεκινήσουμε από τα εύκολα. Στη συνέχεια δίνεται ένας πολύ απλός αλγόριθμος για την εύρεση του Μέγιστου Κοινού Διαιρέτη (ΜΚΔ).

```
     function gcd(m,n)
1.   if m<n then i <-- m else i <-- n
2.   while (m mod i <> 0) or (n mod i <> 0) do
3.      i <-- i-1
4.   return i
```

Ο αλγόριθμος δέχεται στην είσοδο δύο ακέραιους `m` και `n`, βρίσκει το μικρότερο και τον τοποθετεί στη μεταβλητή `i` (εντολή 1). Στη συνέχεια δοκιμάζουμε διαδοχικά τιμές μία-μία, ξεκινώντας από το `i` και ελαττώνοντας προς μικρότερους αριθμούς. Η εντολή 2 είναι ένας βρόχος με δύο ελέγχους που εκτελούν διαιρέσεις και μπορεί να θεωρηθεί ότι το κόστος των ελέγχων αυτών είναι φραγμένο από επάνω (δηλαδή, χωρίς να υπάρχουν άλλα κρυφά κόστη). Επομένως, δεχόμαστε την εντολή 3 ως βαρόμετρο, οπότε πλέον τίθεται το ερώτημα ποιός είναι ο αριθμός των επαναλήψεων. Ο αριθμός αυτός εξαρτάται από τη διαφορά μεταξύ του μικρότερου αριθμού μεταξύ των `m` και `n` αφ'ενός και του ΜΚΔ αφ'ετέρου. Στη χειρότερη περίπτωση ο ΜΚΔ ισούται με τη μονάδα, οπότε ισχύει ότι η πολυπλοκότητα του αλγορίθμου αυτού είναι γραμμική $O(\min(m,n))$.

Ο προηγούμενος αλγόριθμος δόθηκε απλώς για να τον απορρίψουμε στη συνέχεια, καθώς είναι γνωστό ότι ο αλγόριθμος του Ευκλείδη είναι αποτελεσματικότερος.

```
     function euclid(m,n)
1.   while m>0 do
2.      t <-- n mod m
3.      n <-- m
4.      m <-- t
5.   return n
```

Πρόταση.
Η πολυπλοκότητα του αλγορίθμου του Ευκλείδη είναι λογαριθμική.

Απόδειξη.
Πρώτα θα αποδείξουμε ότι για δύο ακεραίους m, n (όπου $n \geq m$) ισχύει: $n \bmod m < n/2$. Διακρίνουμε δύο περιπτώσεις:

- αν $m > n/2$, τότε $1 \leq n/m < 2 \Rightarrow n \bmod m = n - m < n - n/2 = n/2$.
- αν $m \leq n/2$, τότε $n \bmod m < m \leq n/2$.

Ξαναγυρίζουμε στον Ευκλείδη. Έστω ότι k είναι ο συνολικός αριθμός των επαναλήψεων στο βρόχο `while`. Έστωσαν n_i, m_i οι τιμές των μεταβλητών n, m στην έξοδο από την i-οστή επανάληψη του βρόχου (όπου $i \leq k$). Είναι ευνόητο ότι $m_i \geq 1$ αλλά $m_k = 0$. Αν n_0 και m_0 είναι οι αρχικές τιμές των μεταβλητών n και m, τότε στη συνέχεια οι τιμές των μεταβλητών n, m έχουν ως εξής:

$$n_i \;=\; m_{i-1}$$

$$m_i \;=\; n_{i-1} \bmod m_{i-1}$$

Προφανώς ισχύει $n_i > m_i$ για κάθε i. Έτσι για κάθε $i \geq 2$ προκύπτει:

$$m_i \;=\; n_{i-1} \bmod m_{i-1} \;<\; n_{i-1}/2 \;=\; m_{i-2}/2$$

Έστω ότι το k είναι περιττός αριθμός και ισχύει $k = 2d + 1$. Επομένως:

$$m_{k-1} \;<\; m_{k-3}/2 \;<\; m_{k-5}/4 \;<\; \ldots \;<\; m_0/2^d$$

Όμως ισχύει $m_{k-1} \geq 1$, οπότε

$$m_0 \geq 2^d \;\Rightarrow\; \log m_0 \geq d \;\Rightarrow\; \log m_0 \geq (k-1)/2 \;\Rightarrow\; k \leq 1 + 2\log m_0$$

Φθάσαμε στη μισή απόδειξη. Η άλλη μισή αφορά στην περίπτωση όπου το k είναι άρτιος αριθμός. Η απόδειξη αυτή είναι παρόμοια και στηρίζεται στη σχέση $m_1 = n_0 \bmod m_0 < m_0$. Έτσι καταλήγουμε ότι η πολυπλοκότητα του Ευκλείδειου αλγορίθμου για την εύρεση του ΜΚΔ είναι λογαριθμική $\mathrm{O}(\log m)$. □

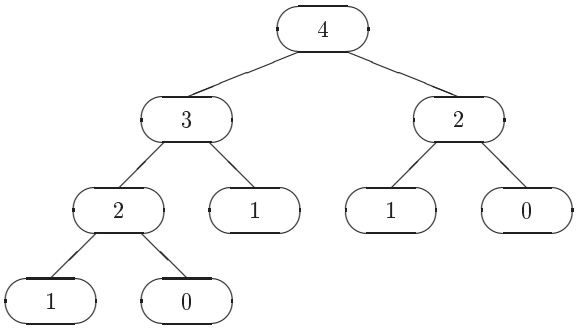
5.2 Υπολογισμός Αριθμών Fibonacci

Στο Κεφάλαιο 1 δόθηκαν μερικά βασικά στοιχεία για τους αριθμούς Fibonacci δεύτερης τάξης. Με βάση αυτόν τον αναδρομικό ορισμό μπορούμε να δώσουμε έναν αναδρομικό αλγόριθμο, όπως ο επόμενος `fib1` που δέχεται μη αρνητικούς ακέραιους αριθμούς n.

```
     function fib1(n)
1.   if n<=1 then return n
2.   else return fib1(n-1)+fib1(n-2)
```

Συχνά η χρήση αναδρομής διευκολύνει τον προγραμματιστή στην υλοποίηση και τον έλεγχο του προγράμματος. Αν και πολλές φορές η αναδρομή φαίνεται ως πιο φυσικός τρόπος προγραμματισμού, ωστόσο πρέπει να χρησιμοποιείται με μέτρο. Μεταξύ ενός απλού επαναληπτικού προγράμματος και ενός αναδρομικού προγράμματος προτιμάται το πρώτο. Ο λόγος είναι ότι κάθε κλήση ενός υποπρογράμματος (δηλαδή, διαδικασίας ή συνάρτησης) έχει χρονικό κόστος μη αμελητέο. Έτσι το κέρδος σε χρόνο προγραμματισμού δημιουργεί απώλεια σε χρόνο εκτέλεσης. Επομένως, αν ο χρόνος απόκρισης είναι κρίσιμος τότε είναι βέβαιο ότι θα πρέπει να προτιμηθεί η επαναληπτική μέθοδος.

Η καλύτερη απόδειξη για την ισχύ των ανωτέρω προκύπτει μελετώντας βαθύτερα τη συνάρτηση `fib1`. Η συνάρτηση αυτή υπολογίζει τις ίδιες τιμές πολλές φορές. Για παράδειγμα, κατά τον υπολογισμό του `fib1(4)` γίνονται οι κλήσεις `fib1(3)` και `fib1(2)`. Ομοίως κατά τον υπολογισμό του `fib1(3)` καλείται για δεύτερη φορά η `fib1(2)`. Με το σκεπτικό αυτό η `fib1(2)` θα κληθεί 2 φορές, η `fib1(1)` θα κληθεί 3 φορές, ενώ η `fib1(0)` θα κληθεί 2 φορές. Οι κόμβοι του δένδρου του επόμενου σχήματος δείχνουν τα ορίσματα όλων των κλήσεων που θα εκτελεσθούν κατά την κλήση της `fib1(4)`.



Σχήμα 5.1: Σειρά κλήσεων για την `fib1(4)`.

Από το σχήμα αυτό φαίνεται ότι τα φύλλα αντιστοιχούν στους δύο τύπους των βασικών κλήσεων με ορίσματα 1 και 0, αντίστοιχα. Καθώς $F_4 = 3$, έπεται ότι το δένδρο θα έχει 3 φύλλα που αντιστοιχούν στις 3 βασικές κλήσεις `fib1(1)` και μερικά ακόμη φύλλα (αυτή τη στιγμή άγνωστος ο αριθμός τους) που θα αντιστοιχούν στις βασικές κλήσεις `fib1(0)`. Γενικεύοντας, είναι ευνόητο ότι

η κλήση του `fib1(n)` μπορεί να παρασταθεί από ένα δυαδικό δένδρο που στα φύλλα θα έχει F_n κλήσεις της `fib1(1)`. Προφανώς, το πλήθος των κόμβων του δένδρου παριστά το συνολικό αριθμό κλήσεων της `fib1`. Στη συνέχεια δίνονται δύο εναλλακτικοί τρόποι υπολογισμού αυτού του συνολικού αριθμού κλήσεων.

Πρόταση.
Ο αριθμός των κλήσεων για την εύρεση του n-οστού αριθμού Fibonacci, F_n, ισούται με $2F_{n+1} - 1$.

Απόδειξη.
Η απόδειξη θα γίνει με διπλή επαγωγή. Σχεδιάζοντας απλά σχήματα όπως το προηγούμενο, παρατηρούμε ότι για τον υπολογισμό του F_1, ο αριθμός των κλήσεων είναι $1 = 2F_2 - 1$. Επίσης, για τον υπολογισμό του F_2, ο αριθμός των κλήσεων είναι $3 = 2F_3 - 1$. Υποθέτουμε ότι η πρόταση ισχύει για $n = k - 1$ και $n = k$, οπότε θα αποδείξουμε ότι ο αριθμός των κλήσεων για τον υπολογισμό του F_{k+1} ισούται με $2F_{k+2} - 1$. Το δένδρο των κλήσεων που παριστά τις κλήσεις για τον υπολογισμό αυτό, θα αποτελείται από μία ρίζα και δύο υποδένδρα, που θα αντιστοιχούν στις κλήσεις για τον υπολογισμό των αριθμών F_k και F_{k-1}. Με βάση τις προηγούμενες υποθέσεις έπεται ότι το πλήθος των κόμβων του τελευταίου δένδρου ισούται με $(2F_{k+1} - 1) + (2F_k - 1) + 1$. Από αυτή τη σχέση προκύπτει το ζητούμενο με απλή άλγεβρα. □

Πρόταση.
Κατά την εύρεση του αριθμού Fibonacci F_n εκτελούνται οι εξής κλήσεις: F_1 φορές η κλήση F_n, F_2 φορές η κλήση F_{n-1}, F_3 φορές η κλήση F_{n-2}, ..., F_{n-1} φορές η κλήση F_2, F_n φορές η κλήση F_1, και F_{n-1} φορές η κλήση F_0.

Απόδειξη.
Η απόδειξη θα γίνει με διπλή επαγωγή. Σχεδιάζοντας απλά σχήματα όπως το προηγούμενο, παρατηρούμε ότι για $n = 1$, δηλαδή για τον υπολογισμό του F_1, ο αριθμός των κλήσεων είναι $F_1 = 1$. Επίσης, για $n = 2$, δηλαδή για τον υπολογισμό του F_2, η κατανομή των κλήσεων έχει ως εξής: η F_2 θα κληθεί $F_1 = 1$ φορά, η F_1 θα κληθεί $F_2 = 1$ φορά και η F_0 θα κληθεί $F_1 = 1$ φορά. Υποθέτουμε ότι η πρόταση ισχύει για $n = k - 1$ και $n = k$. Πρέπει να αποδειχθεί για $n = k + 1$. Κατά τον υπολογισμό του F_{k+1} θα εκτελεσθούν όλες οι κλήσεις που θα εκτελούνταν για τον υπολογισμό του F_k συν όλες οι κλήσεις που θα εκτελούνταν για τον υπολογισμό του F_{k-1} συν ένα. Με απλή παράθεση των επιμέρους κλήσεων και κατάλληλες προσθέσεις προκύπτει η αλήθεια της

πρότασης. □

Πρόταση.
Ο αριθμός των κλήσεων για την εύρεση του αριθμού Fibonacci F_n ισούται με $2F_{n+1} - 1$.

Απόδειξη.
Με βάση την προηγούμενη πρόταση αρκεί να αποδειχθεί ότι:

$$\sum_{i=1}^{n} F_i + F_{n-1} = 2F_{n+1} - 1$$

Με κατάλληλη απλοποίηση προκύπτει ότι ισοδύναμα αρκεί να αποδειχθεί ότι:

$$\sum_{i=1}^{n} F_i = F_{n+2} - 1$$

Η σχέση αυτή θα αποδειχθεί επαγωγικά. Προφανώς η σχέση ισχύει για $n = 1$. Υποθέτουμε ότι ισχύει για $n = k$, οπότε θα αποδείξουμε ότι ισχύει για $n = k+1$, δηλαδή ότι ισχύει η σχέση:

$$\sum_{i=1}^{k+1} F_i = F_{k+3} - 1$$

Ξεκινώντας από το αριστερό σκέλος έχουμε:

$$\sum_{i=1}^{k+1} F_i = \sum_{i=1}^{k} F_i + F_{k+1} = F_{k+2} - 1 + F_{k+1} = F_{k+3} - 1$$

Επομένως, φθάσαμε μέσα από δύο επαγωγικούς δρόμους στο ίδιο συμπέρασμα, δηλαδή ότι η πολυπλοκότητα του αλγορίθμου αυτού είναι $\Theta(F_n)$. □

Το ίδιο συμπέρασμα ως προς την πολυπλοκότητα μπορεί να προκύψει και με μία ακόμη απόδειξη που δίνουμε στη συνέχεια. Ο λόγος της έμφασης στο σημείο αυτό είναι ότι παρόμοια προβλήματα θα αντιμετωπίσουμε στη συνέχεια του μαθήματος αυτού αλλά και σε επόμενα μαθήματα, και επομένως θα ισχύουν τα ίδια συμπεράσματα.

Έστω ότι με $G(n)$ συμβολίζουμε τον αριθμό των κλήσεων που θα εκτελεσθούν για τον υπολογισμό του `fib1(n)`. Προφανώς ισχύει:

$$G(n) \;=\; G(n-1) + G(n-2) + 1$$

με αρχικές συνθήκες $G(0) = G(1) = 1$. Η εξίσωση αυτή είναι μία περίπτωση μη ομογενούς γραμμικής αναδρομικής εξίσωσης και θα μπορούσε να επιλυθεί κατά τα γνωστά. Εδώ θα ακολουθήσουμε ένα διαφορετικό δρόμο αξιοποιώντας την ομοιότητα των συναρτήσεων F_n και $G(n)$. Ας υποθέσουμε ότι η $G(n)$ εξαρτάται γραμμικά από τη συνάρτηση F_n. Δηλαδή ισχύει:

$$G(n) \;=\; aF_n + b$$

όπου οι a, b είναι σταθερές που πρέπει να υπολογισθούν. Επειδή $G(0) = G(1) = 1$ και $F_0 = F_1 = 1$ έπεται ότι $a + b = 1$. Από τη σχέση αυτή και τις αναδρομικές εξισώσεις $G(n)$ και F_n έπεται ότι $b = -1$ άρα $a = 2$. Συνεπώς:

$$G(n) \;=\; 2F_n - 1$$

Επομένως, φθάνουμε στο πρόβλημα της εύρεσης της πολυπλοκότητας της ίδιας της συνάρτησης F_n. Για να βρούμε την πολυπλοκότητα του αλγορίθμου αρκεί να επιλύσουμε την επόμενη αναδρομική εξίσωση που βασίζεται στην ίδια τη φιλοσοφία του ορισμού των αριθμών Fibonacci:

$$T(n) \;=\; T(n-1) + T(n-2) + 2$$

όπου το 2 παριστά το σταθερό κόστος της εντολής 1 και το κόστος της πρόσθεσης στην εντολή 2. Από την ίδια τη μορφή της αναδρομής δημιουργείται η βεβαιότητα ότι η λύση της αναδρομικής αυτής εξίσωσης είναι $\Theta(F_n) = \Theta(\phi^n)$. Είναι δυνατόν με δημιουργική επαγωγή να βρεθούν 3 σταθερές a, b και c έτσι ώστε για κάθε ακέραιο n, να ισχύει: $aF_n \leq T(n) \leq bF_n - c$, σχέση που πιστοποιεί την ανωτέρω πολυπλοκότητα.

Στη συνέχεια παρουσιάζεται η επόμενη συνάρτηση `fib2` για τον υπολογισμό του n-οστού αριθμού Fibonacci, η οποία σε αντίθεση με την `fib1`, είναι επαναληπτική και δεν εκτελεί περιττούς επανα-υπολογισμούς. Η ανάλυσή του είναι σχετικά προφανής. Μέσα στο βρόχο `for` (εντολή 2) εκτελούνται 2 καταχωρίσεις (εντολή 3) με κόστος φραγμένο από επάνω. Έτσι, για λόγους απλοποίησης, θεωρούμε μοναδιαίο κόστος εντός του βρόχου. Ο βρόχος είναι απλός (δηλαδή, όχι φωλιασμένος), οπότε η πολυπλοκότητά του είναι γραμμική $\Theta(n)$. Για να φθάσουμε στο τελευταίο συμπέρασμα, και πάλι για λόγους απλοποίησης, δεν εξετάσαμε το μέγεθος των τελεσταίων, αν δηλαδή αρκούν 4 χαρακτήρες για την αποθήκευσή τους ή όχι.

```
     function fib2(n)
1.   i <-- 1; j <-- 0;
2.   for k <-- 1 to n do
3.      j <-- i+j; i <-- j-i
4.   return j
```

Στη συνέχεια δίνεται μία τρίτη μέθοδος, `fib3`, για τον υπολογισμό των αριθμών Fibonacci. Και στην περίπτωση αυτή αγνοούμε το θέμα του μεγέθους των τελεσταίων για να προχωρήσουμε ευκολότερα στην ανάλυση του αλγορίθμου. Η εντολή 1 έχει κόστος φραγμένο από επάνω, όπως επίσης και το εσωτερικό του βρόχου `while` αποτελείται από εντολές με συνολικό κόστος φραγμένο από επάνω. Επομένως, θεωρώντας ως βαρόμετρο τις εντολές εντός του βρόχου αρκεί να προσδιορίσουμε το πλήθος m των επαναλήψεων του βρόχου.

```
     function fib3(n)
1.   i <-- 1; j <-- 0; k <-- 0; h <-- 1;
2.   while n>0 do
3.      if n mod 2 = 1 then
4.         t <-- j*h; j <-- i*h+j*k+t; i <-- i*k+t
5.      t <-- h*h; h <-- 2*k*h+t;
6.      k <-- k*k+t; n <-- n div 2
7.   return j
```

Πρόταση.
Η πολυπλοκότητα της συνάρτησης `fib3` είναι $\mathrm{O}(\log n)$.

Απόδειξη.
Έστω ότι n_i είναι η τιμή της μεταβλητής n κατά την έξοδο από τον i-οστό βρόχο. Προφανώς για τις οριακές συνθήκες ισχύει $n_1 = \lfloor n/2 \rfloor$ και $n_m = 0$, αλλά γενικότερα για $2 \leq t \leq m$ ισχύει:

$$n_t \;=\; \lfloor \frac{n_{t-1}}{2} \rfloor \;\leq\; \frac{n_{t-1}}{2}$$

Από τα ανωτέρω προκύπτει:

$$n_t \;\leq\; \frac{n_{t-1}}{2} \;\leq\; \frac{n_{t-2}}{4} \;\leq\; \ldots \;\leq\; \frac{n_1}{2^{t-1}} \leq \frac{n}{2^t}$$

Από τη σχέση αυτή απομονώνουμε τους όρους:

$$n_m \;\leq\; \frac{n_{m-1}}{2} \;\leq\; \frac{n}{2^m}$$

Καθώς ισχύει $n_m = 0$, ενώ οι τιμές n_t είναι ακέραιες, έπεται ότι $n_{m-1} = 2$ ή 1. Θέτοντας $n_{m-1} = 2$ στη δεξιά ανισοϊσότητα της ανωτέρω σχέσης προκύπτει ότι: $m \leq \log n$. Επομένως, η πολυπλοκότητα του αλγορίθμου είναι λογαριθμική $O(\log n)$. □

5.3 Οι Πύργοι του Hanoi

Το puzzle των πύργων του Ανόι διατυπώθηκε από το Γάλλο μαθηματικό Francois-Edouard-Anatole Lucas (1842-1891), ο οποίος το "έντυσε" με το γνωστό μύθο, ότι ο θεός τοποθέτησε στη γη 3 διαμαντένιους πύργους και 64 χρυσά δακτυλίδια διαφορετικού μεγέθους μεταξύ τους. Αρχικά, λοιπόν, τα 64 δακτυλίδια ήταν τοποθετημένα σε έναν πύργο κατά σειρά μεγέθους: κάτω τα μεγάλα και επάνω τα μικρά δακτυλίδια. Ο σκοπός των καλόγερων ενός γειτονικού μοναστηριού ήταν να μεταφέρουν τα δακτυλίδια από το αρχικό πύργο σε ένα διπλανό με την ίδια σειρά μεγέθους, δηλαδή κάτω τα μεγάλα και επάνω τα μικρά δακτυλίδια. Η μόνη επιτρεπόμενη κίνηση για ένα δακτυλίδι είναι να μεταφερθεί από έναν πύργο σε έναν άλλο, χωρίς όμως ποτέ να τοποθετείται ένα μεγαλύτερο δακτυλίδι επάνω από ένα μικρότερο. Σύμφωνα με το μύθο, όταν οι καλόγεροι ολοκληρώσουν τη διαδικασία μεταφοράς, τότε θα έρθει το τέλος του κόσμου. Σκοπός του μύθου είναι να δηλώσει τη δυσκολία του εγχειρήματος.

Ο αλγόριθμος και η ανάλυσή του στηρίζεται στο εξής σκεπτικό. Για να μεταφέρουμε τα m ανώτερα (δηλαδή, τα m μικρότερα) δακτυλίδια από έναν πύργο i σε έναν πύργο j, πρώτα θα τοποθετήσουμε τα $m-1$ στον πύργο $6-i-j$, κατόπιν το m-οστό δακτυλίδι στον j-οστό πύργο, και τέλος τα $m-1$ δακτυλίδια από τον πύργο $6-i-j$ στον πύργο j. (Για τις μεταβλητές ισχύουν οι σχέσεις: $1 \leq i, j \leq 3, i \neq j$ και $m \geq 1$.) Η επόμενη διαδικασία περιγράφει αλγοριθμικά το σκεπτικό αυτό. Η λύση στο πρόβλημα δίνεται με την κλήση της διαδικασίας `hanoi(64,1,2)`.

```
     procedure hanoi(m,i,j)
1.   if m>0 then
2.      hanoi(m-1,i,6-i-j)
3.      write(i,"-->",j)
4.      hanoi(m-1,6-i-j,j)
```

Πρόταση.
Η πολυπλοκότητα της διαδικασίας `hanoi` είναι $\Theta(2^n)$.

Απόδειξη.
Θεωρούμε την εντολή 3 (`write`) ως το βαρόμετρο, οπότε η πολυπλοκότητα του προηγούμενου αναδρομικού αλγορίθμου εκφράζεται από την αναδρομική εξίσωση:

$$T(m) \;=\; \begin{cases} 1 & \text{αν} \;\; m = 1 \\ 2T(m-1)+1 & \text{αν} \;\; m > 1 \end{cases}$$

Η εξίσωση αυτή μπορεί να γραφεί ως μία γραμμική μη ομογενής αναδρομική εξίσωση:

$$t_n - 2t_{n-1} \;=\; 1$$

για $n \geq 1$ και $t_0 = 0$, από όπου κατά τα γνωστά προκύπτει η χαρακτηριστική εξίσωση:

$$(x-2)(x-1) \;=\; 0$$

Οι ρίζες της χαρακτηριστικής εξίσωσης είναι 1 και 2, οπότε η γενική λύση είναι της μορφής:

$$t_n \;=\; c_1 1^n + c_2 2^n$$

Χρησιμοποιούμε την τελευταία αναδρομική εξίσωση για να βρούμε μία δεύτερη αρχική συνθήκη:

$$t_1 \;=\; 2t_0 + 1 \;=\; 1$$

Επιλύοντας το σύστημα δύο εξισώσεων με δύο αγνώστους, που προκύπτει από τις αρχικές συνθήκες, έχουμε:

$$\begin{aligned} 0 &= c_1 + c_2 \\ 1 &= c_1 + 2c_2 \end{aligned}$$

από όπου προκύπτει ότι $c_1 = -1, c_2 = 1$ και επομένως η λύση της αναδρομής είναι $t_n = 2^n - 1$. Έτσι καταλήγουμε ότι η πολυπλοκότητα της μεθόδου είναι εκθετική $\Theta(2^n)$. □

Πράγματι, έχει υπολογισθεί ότι για να επιτευχθεί ο σκοπός πρέπει να εκτελεσθούν τουλάχιστον 18.446.744.073.709.551.615 κινήσεις.

5.4 Υπολογισμός Δύναμης

Είναι γνωστό ότι μερικές γλώσσες προγραμματισμού δεν προσφέρουν τελεστή για τον υπολογισμό της δύναμης (για παράδειγμα, η Pascal). Εξ άλλου στην κρυπτογραφία η ύψωση μεγάλων αριθμών σε δυνάμεις είναι συχνή πράξη. Στις

περιπτώσεις αυτές, ο προγραμματιστής πρέπει να υλοποιήσει μία αντίστοιχη συνάρτηση.

Στη συνέχεια δίνεται ένα επιπρόσθετο παράδειγμα χρήσης αναδρομικής και επαναληπτικής μεθόδου για τον υπολογισμό της ακέραιας δύναμης ενός πραγματικού αριθμού a^b. Οι επόμενες συναρτήσεις `power1` και `power2` είναι δύο πρώτες απλές προσεγγίσεις στο θέμα. Σε σχέση με την πρώτη που είνα επαναληπτική, παρατηρούμε ότι βασικά αποτελείται από έναν απλό βρόχο `for` (εντολή 2) που περιέχει μία μόνο εντολή καταχώρισης, η οποία είναι και το βαρόμετρο. Εύκολα προκύπτει από τα όρια του βρόχου ότι η πολυπλοκότητα της μεθόδου είναι γραμμική $\Theta(b)$.

```
     function power1(a,b)
1.   power <-- 1;
2.   for i <-- 1 to b do power1 <-- power1*a
3.   return power1

     function power2(a,b)
1.   if b=0 then return 1
2.   else if b=1 then return a
3.   else return a*power2(a,b-1)
```

Σε σχέση με τη δεύτερη που είναι αναδρομική, ισχύουν τα ίδια συμπεράσματα, δηλαδή έχουμε και πάλι γραμμική επίδοση $\Theta(b)$. Αυτό μπορεί να αποδειχθεί επιλύοντας την αναδρομική εξίσωση:

$$T(n) = \begin{cases} 0 & \text{αν } b \leq 1 \\ T(n-1)+1 & \text{αν } b > 1 \end{cases}$$

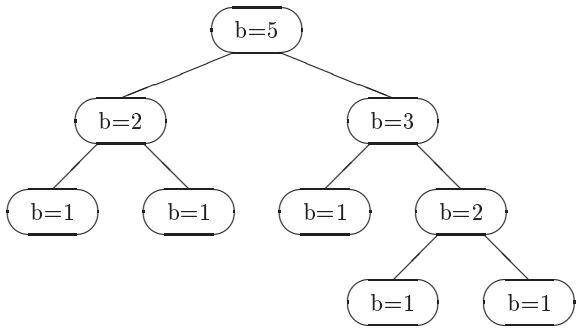
Είναι δυνατόν να επιτύχουμε κάτι καλύτερο;

Παρατηρούμε ότι αν υπολογισθεί το a^2 με έναν πολλαπλασιασμό, τότε το a^4 μπορεί να υπολογισθεί μόνο με έναν επιπλέον υπολογισμό (δηλαδή, $a^4 = a^2 * a^2$) αντί για άλλους δύο (όπως, $a^4 = a^2 * a * a$). Επομένως, γενικεύοντας προκύπτει ότι αν το b είναι άρτιος αριθμός, τότε $a^b = a^{b/2} * a^{b/2}$, αλλιώς $a^b = a^{(b-1)/2} * a^{(b+1)/2}$. Συνεχίζοντας την engineering προσπάθειά μας για βελτίωση της πολυπλοκότητας και με βάση την ιδέα αυτή μπορούμε να παρουσιάσουμε την επόμενη αναδρομική συνάρτηση `power3` που στηρίζεται στην αρχή του Διαίρει και Βασίλευε.

```
     function power3(a,b)
1.   if b=0 then return 1
```

```
2.   else if b=1 then return a
3.   else return power3(a,b/2)*power3(a,(b+1)/2)
```

Αν προσέξουμε καλύτερα αυτή τη συνάρτηση θα παρατηρήσουμε ότι δεν είναι ιδιαίτερα αποτελεσματική γιατί δεν αποφεύγει την εκτέλεση περιττών κλήσεων. Για παράδειγμα, για τον υπολογισμό του a^5, γίνονται οι εξής κλήσεις κατά σειρά: `power3(a,5)`, `power3(a,2)`, `power3(a,1)`, `power3(a,1)`, `power3(a,3)`, `power3(a,1)`, `power3(a,2)`, `power3(a,1)`, `power3(a,1)`. Γιατί γίνονται αυτές οι συγκεκριμένες κλήσεις και μάλιστα με αυτή τη σειρά μπορεί να γίνει κατανοητό αν εκτελέσουμε μία διάσχιση κατά βάθος (dfs) στο επόμενο δένδρο, όπου σε κάθε κόμβο εμφανίζεται η τιμή του εκθέτη b.



Σχήμα 5.2: Σειρά κλήσεων για την `power3(a,5)`.

Όπως και στην περίπτωση της συνάρτησης `fib1`, κατά παρόμοιο τρόπο και στο δένδρο αυτό, λοιπόν, παρατηρούμε ότι ο αριθμός των κλήσεων είναι υπερβολικός. Μάλιστα, στη βάση της αναδρομής για εκθέτη ίσο με 1, γίνονται $b = 5$ κλήσεις. Η επόμενη πρόταση δίνει το συνολικό αριθμό κλήσεων που εκτελούνται για τον υπολογισμό της δύναμης μέσω της `power3`.

Πρόταση.
Για κάθε μη κενό δυαδικό δένδρο, αν n_0 είναι ο αριθμός των φύλλων και n_2 είναι ο αριθμός των κόμβων βαθμού 2, τότε ισχύει:

$$n_0 \;=\; n_2 + 1$$

Απόδειξη.
Αν n_1 είναι το πλήθος των κόμβων βαθμού 1 και n ο συνολικός αριθμός κόμβων, τότε ισχύει:

$$n \;=\; n_0 + n_1 + n_2$$

Εκτός από τη ρίζα για κάθε κόμβο υπάρχει μία σύνδεση που καταλήγει στον κόμβο αυτό. Άρα αν k είναι ο συνολικός αριθμός συνδέσεων, τότε ισχύει:

$$n \;=\; k+1$$

Κάθε σύνδεση όμως ξεκινά είτε από κόμβο βαθμού 1, είτε από κόμβο βαθμού 2. Συνεπώς ισχύει:

$$k \;=\; n_1 + 2n_2$$

Από τις τρεις αυτές σχέσεις προκύπτει η αλήθεια της πρότασης. Βέβαια, στην περίπτωση του δένδρου μας γνωρίζουμε ότι ισχύει $n_1 = 0$ αλλά η απόδειξη έχει μείνει σκοπίμως γενική. □

Συνεπώς, από την πρόταση αυτή προκύπτει ότι εφ'όσον ο αριθμός των φύλλων ισούται με b, έπεται ότι ο συνολικός αριθμός των κόμβων/κλήσεων ισούται με $2b-1$ (διαπιστώνεται με απλή παρατήρηση στο σχήμα αλλά αποδεικνύεται ακόμη και επαγωγικά). Άρα, η μέθοδος Διαίρει και Βασίλευε δεν κατάφερε να εκμεταλλευθεί την ιδέα που είπαμε προηγουμένως, καθώς και πάλι η πολυπλοκότητα είναι $\Theta(b)$, ενώ δεν πρέπει να μας διαφεύγει από την προσοχή και η κρυμμένη σταθερά που ισούται με 2. Ο λόγος είναι ότι ενώ υπολόγισε το a^2 και το a^1, δεν τα αποθήκευσε ώστε να τα χρησιμοποιήσει αργότερα. Η επόμενη αναδρομική συνάρτηση `power4` αξίζει να προσεχθεί γιατί εφαρμόζει ακριβώς αυτό το σκεπτικό.

```
     function power4(a,b)
1.   if b=0 then return 1
2.   else if (b mod 2)=0 then return power4(a*a,b div 2)
3.   else return power4(a*a,b div 2)*a
```

Πρόταση.
Η πολυπλοκότητα της `power4` είναι λογαριθμική $\mathrm{O}(\log b)$.

Απόδειξη.
Αν θεωρήσουμε ότι βαρόμετρο είναι οι εκτελούμενοι πολλαπλασιασμοί, τότε προκύπτει η επόμενη σχέση:

$$T(n) \;=\; \begin{cases} 0 & \text{αν } n = 0 \\ T(n/2)+1 & \text{αν } n \bmod 2 = 0 \\ T((n-1)/2)+2 & \text{αν } n \bmod 2 = 1 \end{cases}$$

όπου θεωρούμε το χειρότερο ενδεχόμενο (δηλαδή, να χρειάζονται δύο πολλαπλασιασμοί για να υποδιπλασιάσουμε την τάξη του προβλήματος) και καταλήγουμε στην

επόμενη μορφή της:

$$T(n) \;=\; T(n/2) + 2$$

Η επίλυση αυτής της αναδρομικής εξίσωσης είναι πολύ εύκολη υπόθεση όπως εξετάσθηκε σε προηγούμενο κεφάλαιο. Θα μπορούσαμε να χρησιμοποιήσουμε τη μέθοδο της χαρακτηριστικής εξίσωσης ή το γενικό θεώρημα. Αντ'αυτών θα χρησιμοποιήσουμε τη μέθοδο της αντικατάστασης. Έτσι διαδοχικά έχουμε ότι:

$$\begin{aligned} T(n) &= T(n/2) + 2 \\ T(n/2) &= T(n/4) + 2 \\ &\vdots \\ T(1) &= T(0) + 2 \end{aligned}$$

Συνεπώς αντικαθιστώντας προκύπτει ότι:

$$T(n) \;=\; 2\log n$$

οπότε η πολυπλοκότητα της συνάρτησης `power4` είναι λογαριθμική $\mathrm{O}(\log b)$. □

Το πρόβλημα αυτό για τον υπολογισμό της δύναμης ήταν μία καλή ευκαιρία για να μελετήσουμε αρκετές εναλλακτικές λύσεις και να διαπιστώσουμε τα πλεονεκτήματα και τα μειονεκτήματα της κάθε μίας από αυτές. Στη συνέχεια θα εξετάσουμε έναν ακόμη αλγόριθμο `power5` που βασίζεται στο δυναμικό προγραμματισμό, ο οποίος υποθέτει ότι υπάρχει μία δομή πίνακα `store`.

```
     function power5(a,b);
1.   store[1] <-- a; i <-- 1; exponent <-- 1;
2.   while (exponent<b) do
3.      i <-- i+1; exponent <-- 2*exponent;
4.      store[i] <-- store[i-1]*store[i-1]
5.   index <-- 0; power5 <-- 1
6.   while (index<b) do
7.      if (index+exponent<=b) then
8.         power5 <--  power5 * store[i];
9.         index <--  index + exponent
10.     exponent <--  exponent/2; i <-- i-1
11.  return power5
```

Η συνάρτηση αυτή κατ'αρχήν υπολογίζει όλες τις δυνάμεις του `a` με εκθέτη 1, 2, 4, 8 κοκ και τις αποθηκεύει σε ένα πίνακα `store` μεγέθους $\lceil \log b \rceil$ θέσεων.

Στη συνέχεια επιλέγει αυτές που χρειάζεται για να υπολογίσει το a^b. Για παράδειγμα, για τον υπολογισμό του 3^{19} θα προκύψει ο Πίνακας 5.4, όπου στην επάνω γραμμή εμφανίζεται η τιμή του `exponent`, δηλαδή του εκθέτη όπου θα πρέπει να υψώσουμε τη βάση 3 για να μας δώσει το αντίστοιχο περιεχόμενο του πίνακα. Επομένως, τελικά, για τον υπολογισμό του 3^{19} θα πολλαπλασιάσουμε το περιεχόμενο της 1ης, τη 2ης και της 5ης θέσης του πίνακα επειδή $3^{19} = 3^{16+2+1}$.

1	2	4	8	16
3	9	81	6561	43046721

Πίνακας 5.1: Υπολογισμός δύναμης

Ποιά είναι η πολυπλοκότητα της μεθόδου; Για να απαντήσουμε αυτή την ερώτηση αρκεί να προσέξουμε τον ψευδοκώδικα της συνάρτησης. Στο πρώτο βρόχο `while` (εντολή 2) εκτελούμε $\lceil \log b \rceil$ επαναλήψεις, ενώ το εσωτερικό του δεύτερου βρόχου `while` (εντολή 6) επαναλαμβάνεται λιγότερο από $\lceil \log b \rceil$ φορές. Επομένως, η πολυπλοκότητα είναι της τάξης $\Theta(\log b)$.

Είναι δυνατόν να βελτιώσουμε ακόμη περισσότερο την υλοποίηση της προηγούμενης ιδέας έτσι ώστε να μην υπολογίζουμε δυνάμεις που δεν θα χρειασθούμε στη συνέχεια. Κλείνουμε, λοιπόν, την παράγραφο αυτή με την επόμενη συνάρτηση `power6` που είναι επαναληπτική, στηρίζεται στην ίδια φιλοσοφία με τη συνάρτηση `power5` και έχει την ίδια πολυπλοκότητα. Ωστόσο, είναι αναμενόμενο να είναι ταχύτερη από την προηγούμενη, πρώτον γιατί είναι επαναληπτική και όχι αναδρομική, δεύτερον γιατί δεν χρησιμοποιεί τον επιπλέον πίνακα, και τρίτον γιατί δεν θα περιέχει την κρυμμένη σταθερά 2.

```
     function power6(a,b)
1.   i <-- b div 2; j <-- b mod 2;
2.   if j<>0 then power6 <-- a
3.   while i<>0 do
5.      a <-- a*a; j <-- i mod 2; i <-- i div 2;
7.      if j<>0 then power6 <-- power6 * a
8.   return power6
```

5.5 Υπολογισμός Συνδυασμού

Στην παράγραφο αυτή θα εξετάσουμε ένα απλό υπολογιστικό πρόβλημα που δίνει τη δυνατότητα να αναπτύξουμε σταδιακά μία engineering προσέγγιση βελτιώνοντας

διαδοχικά αναδρομικούς και επαναληπτικούς αλγορίθμους.

Από το Κεφάλαιο 2.1 γνωρίζουμε τις εξής σχέσεις για το συνδυασμό των n αντικειμένων ανά k :

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!} = \binom{n}{n-k}$$

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1} = \binom{n-1}{k-1}\frac{n}{k}$$

και επομένως ο υπολογισμός του συνδυασμού ανάγεται στον υπολογισμό των παραγοντικών. Ας αρχίσουμε, λοιπόν, από την εξέταση του παραγοντικού με σκοπό να φθάσουμε στην αποτελεσματικότερη υλοποίηση. Στη συνέχεια δίνονται δύο εκδοχές, μία αναδρομική και μία επαναληπτική.

```
     function factorial1(n);
1.   if n=0 then return 1
2.   else return n*factorial1(n-1)

     function factorial2(n);
1.   product <-- 1;
2.   for i <-- 2 to n do product <-- i*product;
3.   return product
```

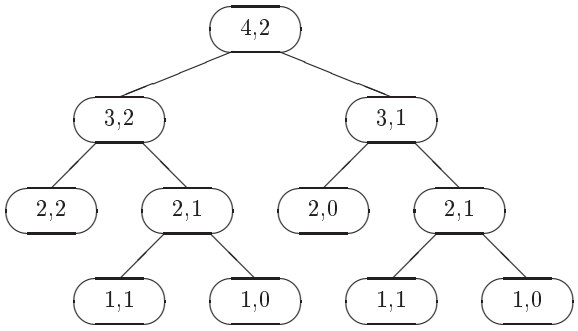
Είναι εύκολο να διαπιστώσουμε ότι θεωρητικά οι δύο εκδοχές είναι ισοδύναμες καθώς διακρίνονται από γραμμική πολυπλοκότητα $\Theta(n)$. Ωστόσο, εξ ίσου ευνόητο είναι ότι πρακτικά η επαναληπτική εκδοχή είναι ταχύτερη σύμφωνα με όσα έχουμε αναφέρει προηγουμένως. Στη συνέχεια ερχόμαστε στο ζητούμενο, στη μελέτη του υπολογισμού των συνδυασμών.

Η επόμενη διαδικασία comb1 είναι αναδρομική και στηρίζεται στην ιδιότητα των συνδυασμών, με την οποία ο υπολογισμός ενός συγκεκριμένου συνδυασμού ανάγεται στον υπολογισμό απλούστερων. Η διαδικασία αυτή είναι αναποτελεσματική καθώς πραγματοποιούνται επανα-υπολογισμοί πολλών συνδυασμών.

```
     function Comb1(n,k);
1.   if (k=0) or (k=n) then return 1
2.   else return Comb1(n-1,k-1)+Comb1(n-1,k)
```

Για παράδειγμα, κατά τον υπολογισμό του συνδυασμού $\binom{4}{2}$ θα προκύψουν οι κλήσεις για τον υπολογισμό των συνδυασμών $\binom{3}{2}$ και $\binom{3}{1}$. Κατά τον υπολογισμό

των δύο τελευταίων συνδυασμών θα προκύψουν δύο κλήσεις για τον υπολογισμό του συνδυασμού $\binom{2}{1}$. Το Σχήμα 5.3 παρουσιάζει μία δενδρική μορφή με κόμβους που εσωκλείουν τα ορίσματα όλων των κλήσεων που θα πραγματοποιηθούν για τον υπολογισμό του συνδυασμού των 4 αντικειμένων ανά 2. Ανάλογο σχήμα είχαμε κατασκευάσει και κατά την εξέταση των συναρτήσεων `fib1` και `power2`. Η πολυπλοκότητα της εκδοχής `comb1` είναι $\Theta\left(\binom{n}{k}\right)$. Για την ακρίβεια, ο αριθμός των κόμβων κάθε αντίστοιχου δένδρου είναι $2\binom{n}{k} - 1$.



Σχήμα 5.3: Κλήσεις για τον υπολογισμό του `comb1(4,2)`.

Από αυτό το σημείο εκκίνησης θα προχωρήσουμε σε κομψότερες λύσεις. Το πρώτο μέλημά μας είναι να απαλλαγούμε από την αναδρομή. Αυτό μπορούμε να το επιτύχουμε βασιζόμενοι στις προηγούμενες συναρτήσεις για τον υπολογισμό του παραγοντικού, όπως φαίνεται στην επόμενη συνάρτηση. Εφ'όσον ο υπολογισμός του παραγοντικού με την `factorial2` είναι μία γραμμική διαδικασία, έπεται ότι και η διαδικασία `comb2` είναι επίσης γραμμική ως προς n, δηλαδή είναι $\Theta(n)$. Βέβαια αξίζει να σημειωθεί ότι υπεισέρχεται μία σταθερά ίση με 2, που πρακτικά είναι ένα πρόβλημα που πρέπει να απαλείψουμε.

```
     function comb2(n,k);
1.   t1 <-- factorial2(n);
2.   t2 <-- factorial2(k);
3.   t3 <-- factorial2(n-k);
4.   return t1/(t2*t3)
```

Αν και η συνάρτηση `comb2` ήταν τεράστια πρόοδος σε σχέση με τη διαδικασία `comb1`, εν τούτοις υπάρχουν περιθώρια βελτίωσης αν παρατηρήσουμε ότι η `comb2` εκτελεί περιττούς πολλαπλασιασμούς στον αριθμητή που απλοποιούνται στη συνέχεια με αντίστοιχες πράξεις στον παρονομαστή. Η επόμενη διαδικασία `comb3` δεν

εκτελεί επανα-υπολογισμούς αν και είναι αναδρομική. Αν θεωρήσουμε την εντολή 2 και εκτελέσουμε τις απλοποιήσεις, τότε παρατηρούμε ότι τόσο ο αριθμητής όσο και ο παρονομαστής είναι γινόμενα ακριβώς k όρων. Από αυτήν την παρατήρηση συνάγεται ότι η πολυπλοκότητα της διαδικασίας αυτής είναι γραμμική ως προς k (όπου προφανώς $k \leq n$), δηλαδή είναι $\Theta(k)$, με μία κρυμμένη σταθερά ίση με 2. Στο ίδιο συμπέρασμα καταλήγουμε και τυπικά επιλύοντας την αναδρομική σχέση:

$$T(k) \ = \ T(k-1) + 2$$

όπου η σταθερά 2 προκύπτει λόγω του κόστους του ενός πολλαπλασιασμού και της μίας διαίρεσης στην εντολή 2 (εξ ου και η κρυμμένη σταθερά). Για να γίνει περισσότερο κατανοητή αυτή η αναδρομική εξίσωση σημειώνουμε ότι καθώς $n > k$, έπεται ότι θα γίνουν τόσες κλήσεις, όσες χρειάζονται μέχρι να μηδενισθεί το k (και όχι το n).

```
     function comb3(n,k);
1.   if (k=0) or (k=n) then return 1
2.   else return comb3(n-1,k-1)*n/k
```

Η μέθοδος αυτή μπορεί να βελτιωθεί περαιτέρω αν κάνουμε χρήση της ανωτέρω ιδιότητας των συνδυασμών, δηλαδή ότι $\binom{n}{k} = \binom{n}{n-k}$. Η επόμενη διαδικασία `comb4` πρώτα υπολογίζει το ελάχιστο μεταξύ των k και $n-k$, ώστε να εξοικονομήσει έναν αριθμό πράξεων. Αν και η τεχνική αυτή βελτιώνει την κατάσταση, εντούτοις προκύπτει επίσης γραμμική πολυπλοκότητα της τάξης $\Theta(\min(k, n-k))$, και πάλι με μία κρυμμένη σταθερά ίση με 2.

```
     function comb4(n,k);
1.   if k>n-k then k <-- n-k
2.   t1 <-- 1;
3.   for i <-- n downto n-k+1 do t1 <-- t1*i;
4.   t2 <-- factorial2(k);
5.   return t1/t2
```

Οι συναρτήσεις `comb3` και `comb4`, αν και γραμμικές, έχουν ένα πρακτικό μειονέκτημα. Κατά τον υπολογισμό του συνδυασμού προκύπτουν ενδιάμεσα αποτελέσματα που είναι μεγαλύτερα από το τελικό αποτέλεσμα. Έτσι μπορεί να προκύψει πρόβλημα υπερχείλισης αναλόγως με τις τιμές των n, k και τους χρησιμοποιούμενους τύπους δεδομένων. Για το λόγο αυτό, προχωρούμε σε μία ακόμη λύση, που τυχαίνει να είναι αναδρομική, σύμφωνα με την οποία εκτελούνται διαιρέσεις σε πρώιμο στάδιο πριν τους πολλαπλασιασμούς ώστε να

αποφευχθεί η υπερχείλιση. Αυτό επιτυγχάνεται θεωρώντας προηγουμένως το ΜΚΔ των n, k, όπως παρουσιάζεται στην επόμενη διαδικασία comb5.

```
     function comb5(n,k);
1.   d <-- Gcd(n,k); q <-- k/d;
2.   if (k=0) or (k=n) then return 1
3.   else return(Comb5(n-1,k-1)/q)*n/d
```

Είναι γνωστό από το Κεφάλαιο 5.1 ότι η εύρεση του ΜΚΔ έχει λογαριθμική πολυπλοκότητα $\Theta(\log k)$. Έτσι θα προκύψει προς επίλυση η αναδρομική εξίσωση:

$$T(k) \;=\; T(k-1) + 4 + \log k$$

όπου η σταθερά 4 συνάγεται καταμετρώντας τους πολλαπλασιασμούς και διαιρέσεις των εντολών 1 και 3. Με τη μέθοδο της αντικατάστασης προκύπτει τελικά ότι:

$$T(k) \;=\; 4k + \log k!$$

από όπου με τη βοήθεια του τύπου του Stirling προκύπτει ότι η πολυπλοκότητα της συνάρτησης comb5 είναι $\Theta(k \log k)$. Με λίγα λόγια έχουμε καταλήξει σε μία βραδύτερη συνάρτηση αλλά το πλεονέκτημά της είναι η σταθερότητά της (robustness) για διάφορες τιμές των n και k. Δηλαδή, θα έχουμε πρόβλημα υπερχείλισης για μεγαλύτερες τιμές των n και k.

Στη συνέχεια παρουσιάζουμε την τελευταία εκδοχή comb6 που αποτελεί ό,τι αποτελεσματικότερο μπορούμε να επιτύχουμε από θεωρητική και πρακτική άποψη. Η μέθοδος αυτή έχει τα εξής τρία πλεονεκτήματα:

1. είναι βέλτιστη με πολυπλοκότητα $\Theta(\min(k, n-k))$,
2. είναι σταθερή καθώς εκτελεί εναλλάξ πολλαπλασιασμούς και διαιρέσεις αποφεύγοντας την υπερχείλιση των χρησιμοποιούμενων τύπων.
3. είναι επαναληπτική και αποφεύγει τη χρονική επιβάρυνση των αναδρομικών και του αντίστοιχου κόστους σε χώρο αποθήκευσης,

```
     function comb6(n,k);
1.   t <-- 1;
2.   if k>n-k then k <-- n-k
3.   for i <-- n downto n-k+1 do t <-- t*i/(n-i+1)
4.   return t
```

5.6 Πολλαπλασιασμός Πινάκων

Έστω ότι δίνονται δύο τετραγωνικοί πίνακες A και B μεγέθους $n \times n$ και ζητείται το γινόμενό τους, ο πίνακας C μεγέθους επίσης $n \times n$. Ο πρώτος τρόπος που έρχεται κατά νου είναι να χρησιμοποιήσουμε έναν αλγόριθμο που θα υπολογίζεται κάθε θέση (i, j) του πίνακα C με τη βοήθεια του τύπου:

$$C(i,j) \;=\; \sum_{k=1}^{n} A(i,k)\; B(k,j)$$

που θα υλοποιηθεί με ένα εσωτερικό βρόχο ως προς k μέσα σε έναν άλλο διπλό βρόχο ως προς i και j. Συνεπώς, θεωρώντας ως βαρόμετρο τους εκτελούμενους πολλαπλασιασμούς πραγματικών, από τον τριπλό βρόχο θα προκύψει μία κυβική πολυπλοκότητα $\mathrm{O}(n^3)$. Στο ίδιο συμπέρασμα θα καταλήξουμε θεωρώντας ότι ο πίνακας C έχει n^2 θέσεις, ενώ κάθε θέση απαιτεί την εκτέλεση n πολλαπλασιασμών. Στη συνέχεια θα προσπαθήσουμε να βελτιώσουμε την πολυπλοκότητα της εκτέλεσης του πολλαπλασιασμού πινάκων βασιζόμενοι στη μέθοδο Διαίρει και Βασίλευε.

Έστω ότι ισχύει $n = 2^k$. Μπορούμε να θεωρήσουμε ότι οι πίνακες A και B αποτελούνται ο καθένας από τέσσερις υποπίνακες μεγέθους $\frac{n}{2} \times \frac{n}{2}$. Έτσι, θα μπορούσε να προκύψει η σχέση:

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \;=\; \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

από όπου κάθε υποπίνακας του πίνακα C θα υπολογίζονταν με βάση τις σχέσεις:

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22} \end{aligned}$$

Καθώς έχουμε θεωρήσει ότι το n είναι δύναμη του δύο, θα μπορούσαμε αναδρομικά να υποδιαιρέσουμε τους υποπίνακες των A και B σε ακόμη μικρότερους υποπίνακες μέχρι του σημείου $n = 2$, οπότε θα εκτελούσαμε εύκολα τους πολλαπλασιασμούς των πραγματικών αριθμών. Για να διατυπώσουμε την αναδρομική εξίσωση που εκφράζει την πολυπλοκότητα του πολλαπλασιασμού αρκεί να παρατηρήσουμε ότι σε κάθε επίπεδο της αναδρομής απαιτούνται 8 πολλαπλασιασμοί και 4 προσθέσεις υποπινάκων. Έτσι προκύπτει η αναδρομική εξίσωση:

$$T(n) \;=\; \begin{cases} b & \text{αν} \;\; n \leq 2 \\ 8T(n/2) + cn^2 & \text{αν} \;\; n > 2 \end{cases}$$

όπου το b εκφράζει το σταθερό κόστος του πολλαπλασιασμού στη βάση της αναδρομής, ενώ το c εκφράζει μία σταθερά που υπεισέρχεται στην πολυπλοκότητα που αφορά στην πρόσθεση δύο υποπινάκων. Αυτή η αναδρομική εξίσωση μπορεί να επιλυθεί με πολλούς τρόπους (ως μη ομογενής, με αντικατάσταση, με το γενικό θεώρημα κλπ.). Σε κάθε περίπτωση το αποτέλεσμα είναι να προκύψει κυβική πολυπλοκότητα $\mathrm{O}(n^3)$. Δηλαδή, αν και χρησιμοποιήσαμε έναν αλγόριθμο βασιζόμενο στη μεθοδολογία Διαίρει και Βασίλευε, εντούτοις δεν καταφέραμε να βελτιώσουμε την πολυπλοκότητα του αλγορίθμου. Αυτό θα επιτευχθεί με τη μέθοδο του Strassen, η οποία προτάθηκε το 1960 και βασίζεται επίσης στη μεθοδολογία Διαίρει και Βασίλευε.

Έστω κατ'αρχήν ότι οι πίνακες A και B είναι μεγέθους 2×2. Αν υπολογίσουμε τις επόμενες ποσότητες:

$$\begin{aligned} m_1 &= (a_{11}+a_{22})(b_{11}+b_{22}) \\ m_2 &= (a_{11}+a_{22})b_{11} \\ m_3 &= a_{11}(b_{12}-b_{22}) \\ m_4 &= a_{22}(b_{21}-b_{11}) \\ m_5 &= (a_{11}+a_{12})b_{22} \\ m_6 &= (a_{21}-a_{11})(b_{11}+b_{12}) \\ m_7 &= (a_{12}-a_{22})(b_{21}+b_{22}) \end{aligned}$$

τότε μπορεί εύκολα να επαληθευθεί ότι ο πίνακας C ισούται με:

$$\begin{pmatrix} m_1+m_4-m_5+m_7 & m_3+m_5 \\ m_2+m_4 & m_1+m_3-m_2+m_6 \end{pmatrix}$$

Εδώ πλέον παρατηρούμε ότι για την εκτέλεση αυτού του απλού πολλαπλασιασμού πινάκων απαιτούνται 7 πολλαπλασιασμοί και 18 προσθέσεις και αφαιρέσεις. Αν και ο αριθμός των προσθέσεων/αφαιρέσεων είναι τώρα μεγαλύτερος από ότι με την προηγούμενη μέθοδο Διαίρει και Βασίλευε, τώρα εξοικονομούμε έναν πολλαπλασιασμό, γεγονός που είναι σημαντικό καθώς ο αριθμός των πολλαπλασιασμών είναι το βαρόμετρό μας. Αν, λοιπόν, γενικεύσουμε αυτήν την τεχνική για μεγαλύτερους τετραγωνικούς πίνακες, προκύπτει αντίστοιχα η εξής αναδρομική εξίσωση:

$$T(n) = \begin{cases} b & \text{αν } n \le 2 \\ 7T(n/2)+cn^2 & \text{αν } n > 2 \end{cases}$$

Μεταξύ των εναλλακτικών τρόπων για την επίλυση αυτής της αναδρομικής εξίσωσης ας επιλέξουμε να χρησιμοποιήσουμε το γενικό θεώρημα. Ισχύει ότι

$n^{\log_b a} = n^{\log_2 7} = n^{2,81} > f(n) = n^2$ για $\epsilon = 0,81$. Επομένως αναγόμαστε στο πρώτο σκέλος του γενικού θεωρήματος και προκύπτει ότι η πολυπλοκότητα της μεθόδου του Strassen για τον πολλαπλασιασμό πινάκων είναι πολυπλοκότητας $O(n^{2,81})$. Προφανώς, στο ίδιο συμπέρασμα θα καταλήγαμε χρησιμοποιώντας τη μέθοδο της αντικατάστασης.

Ως τελική σημείωση σημειώνεται ότι στη βιβλιογραφία αναφέρονται παρόμοιες λύσεις προς την προηγούμενη. Από αυτές άλλες είναι χειρότερες (δηλαδή, εκτελούν 7 πολλαπλασιασμούς και 24 προσθαφαιρέσεις), άλλες είναι ίδιες (εκτελούν 7 πολλαπλασιασμούς και 18 προσθαφαιρέσεις), ενώ τέλος άλλες είναι καλύτερες καθώς με έξυπνο τρόπο εκτελούν 7 πολλαπλασιασμούς και 15 προσθαφαιρέσεις (αποθηκεύοντας και μη επανα-υπολογίζοντας μερικά στοιχεία). Βέβαια σε κάθε περίπτωση όλες αυτές οι λύσεις είναι ασυμπτωτικά ισοδύναμες καθώς το πλήθος εκτέλεσης της πράξης βαρόμετρου του πολλαπλασιασμού είναι το ίδιο. Επίσης, σημειώνεται ότι έχουν προταθεί και άλλες μέθοδοι με χαμηλότερη πολυπλοκότητα, όπως $O(n^{2.376})$, αλλά με πολύ μεγάλη κρυμμένη σταθερά που τις καθιστά μη πρακτικές. Τέλος, σημειώνεται ότι τα ανωτέρω έχουν θεωρητική σημασία. Για παράδειγμα, έχει αποδειχθεί ότι η μέθοος Strassen είναι αποτελεσματικότερη από τη συμβατική μέθοδο για πυκνούς πίνακες με $n > 100$.

5.7 Βιβλιογραφική Συζήτηση

Τα προβλήματα της εύρεσης μέγιστου κοινού διαιρέτη (αλγόριθμος Ευκλείδη) και του πολλαπλασιασμού πινάκων είναι κλασικά αντικείμενα που βρίσκονται σε όλα σχεδόν τα διδακτικά εγχειρίδια αλγορίθμων. Εξ ίσου δημοφιλής με τα προηγούμενα προβλήματα είναι και η ακολουθία των αριθμών Fibonacci, για την οποία ο αναγνώστης μπορεί να ανατρέξει στο άρθρο [102], που αναφέρεται αποκλειστικά στο θέμα, καθώς επίσης και στα άρθρα [39, 97, 131]. Η συνάρτηση `fib3` αναφέρεται στο άρθρο [62]. Με αφορμή το πρόβλημα των πύργων του Ανόι έχουν υπάρξει πολλές παραλλαγές. Για παράδειγμα, 300 τέτοιες παραλλαγές έχουν καταγραφεί στην ιστοσελίδα `http://www.cs.wm.edu/~pkstoc/toh.html`. Το βιβλίο [17] αναφέρεται στον υπολογισμό των δυνάμεων. Σχετικά με τον υπολογισμό συνδυασμών είναι τα άρθρα [96, 133]. Ο αλγόριθμος του Strassen έχει δημοσιευθεί στο άρθρο [154], το άρθρο [25] αναφέρεται στο πρόβλημα της αποτελεσματικής υλοποίησης πολλαπλασιασμού πινάκων κατά Strassen, ενώ το άρθρο [121] βελτιώνει την πολυπλοκότητα του αλγορίθμου Strassen. Η Άσκηση 7 σχετικά με τους 4 πύργους του Ανόι προέρχεται από το άρθρο [23], οι Ασκήσεις 8-9 από το άρθρο [113],ενώ η Άσκηση 14 από το άρθρο [121].

5.8 Ασκήσεις

1. Δίνεται το επόμενο τμήμα ψευδοκώδικα. Ποιά είναι η πολυπλοκότητά του. Τί υπολογίζει;

```
rel <-- 0; tot <-- 0;
for i <-- 1 to n do
   for j <-- i+1 to n do
      tot <-- tot+1;
      if GCD(i,j)=1 then rel <-- rel+1;
```

2. Να σχεδιασθούν και να αναλυθούν εναλλακτικοί αλγόριθμοι εύρεσης του ΜΚΔ τριών ακεραίων αριθμών.

3. Να αποδειχθεί ότι ένας αριθμός d είναι ο ΜΚΔ δύο αριθμών a και b, αν και μόνο αν υπάρχουν δύο ακέραιοι x και y, έτσι ώστε να ισχύει: $d = ax + by$.

4. Ο επόμενος ψευδοκώδικας επεκτείνει τον αλγόριθμο του Ευκλείδη για την εύρεση του ΜΚΔ δύο αριθμών a και b. Πέραν του ΜΚΔ (που συμβολίζεται με d), βρίσκει και δύο αριθμούς x και y ώστε να ισχύει $d = ax + by$.

```
function extended_euclid(a,b)
if b=0 then return (1,0,a);
(x',y',d) <-- extended_euclid(b,a mod b);
return (y',x'- trunc(a/b)*y',d)
```

   Ποιά είναι η πολυπλοκότητα του αλγορίθμου; Να σχεδιασθούν εναλλακτικές εκδοχές του. Σημειώνεται ότι η συνάρτηση `trunc` υλοποιεί τη συνάρτηση πάτωμα.

5. Να αποδειχθεί ότι:

   - $\mathrm{GCD}(F_{n+1}, F_n) = 1$
   - $\mathrm{GCD}(F_m, F_n) = F_{\mathrm{GCD}(m,n)}$.

6. Να θεωρηθεί το puzzle των πύργων του Ανόι με 3 πύργους και $2n$ δίσκους που αποτελούνται από n ζεύγη όμοιων δίσκων. Να αναλυθεί η περίπτωση.

7. Να θεωρηθεί το puzzle των πύργων του Ανόι με 4 πύργους αντί 3. Η στρατηγική που θα ακολουθηθεί είναι η εξής. Αρχικά οι n δίσκοι βρίσκονται στον πρώτο πύργο και πρέπει να μεταφερθούν στον τέταρτο. Λαμβάνονται,

λοιπόν, οι $n-k$ δίσκοι και τοποθετούνται στο δεύτερο πύργο μέσω του τρίτου πύργου ακολουθώντας τη βέλτιστη τακτική του κλασικού προβλήματος για τρεις πύργους. Στη συνέχεια οι k δίσκοι με τον ίδιο κλασικό τρόπο τοποθετούνται στον τέταρτο πύργο μέσω του τρίτου. Τελικά μένει να μεταφερθούν οι $n-k$ δίσκοι από το δεύτερο πύργο στον τέταρτο. Ποιά τιμή του k βελτιστοποιεί τη διαδικασία; Να σχεδιασθεί αλγόριθμος υπολογισμού του βέλτιστου k για διάφορες τιμές του n, αρχίζοντας με $n = 3$. Να βρεθεί το πρότυπο αύξησης του k αυξανομένου του n.

8. Το γραμμικό (linear) πρόβλημα των πύργων του Ανόι απαιτεί τη μεταφορά των δίσκων από τον αριστερότερο στο δεξιότερο πύργο μόνο δια μέσου του μεσαίου πύργου (και ποτέ απευθείας). Να σχεδιασθεί και να αναλυθεί αλγόριθμος επίλυσης της παραλλαγής.

9. Το γραμμικό δίδυμο (linear twin) πρόβλημα των πύργων του Ανόι θεωρεί ότι στον αριστερότερο δίσκο υπάρχουν n άσπροι δίσκοι, ενώ στο δεξιότερο πύργο υπάρχουν n μαύροι δίσκοι. Να σχεδιασθεί και να αναλυθεί αλγόριθμος που να επιτυγχάνει τη μεταφορά των αριστερών δίσκων στο δεξιότερο πύργο και τον μαύρων δίσκων στον αριστερότερο δίσκο χρησιμοποιώντας απαραιτήτως τον ενδιάμεσο δίσκο για κάθε μετακίνηση από το ένα άκρο στο άλλο.

10. Οι αριθμοί Fibonacci μπορούν να υπολογισθούν με πολλαπλασιασμό πινάκων. Για παράδειγμα, ισχύει:

$$\begin{pmatrix} F_1 \\ F_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

ενώ γενικώς ισχύει:

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}^n \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

Ζητείται: (α) η έκφραση αυτή να αποδειχθεί, και (β) να βρεθεί η πολυπλοκότητα για τον υπολογισμό του F_n.

11. Κατά σύμπτωση ο αριθμός $\phi = 1,61803$ ισούται περίπου με την αναλογία χιλιομέτρου προς μίλι (1 km = 1,609344 mi). Έτσι μία απόσταση F_{n+1} χιλιομέτρων ισούται με F_n μίλια περίπου. Να σχεδιασθεί και να αναλυθεί αλγόριθμος μετατροπής χιλιομετρικών αποστάσεων σε μίλια χρησιμοποιώντας τους αριθμούς Fibonacci. Μπορεί, επίσης, να σχεδιασθεί και η αντίστροφη διαδικασία για τη μετατροπή αποστάσεων από μίλια σε χιλιόμετρα.

12. Να χρησιμοποιηθεί το σκεπτικό της Άσκησης 11 για τη σχεδίαση και ανάλυση ενός αλγορίθμου για τη μετατροπή μίας επιφάνειας σε τετραγωνικά χιλιόμετρα σε τετραγωνικά μίλια και το αντίστροφο.

13. Να πολλαπλασιασθούν δύο σύνθετοι αριθμοί $x = a + bi$ και $y = c + di$ με 3 πολλαπλασιασμούς.

14. Έστω ένας αλγόριθμος που πολλαπλασιάζει δύο πίνακες 70x70 με 143.640 πολλαπλασιασμούς. Είναι λιγότερο ή περισσότερο αποτελεσματικός από τη μέθοδο Strassen;

6

Αλγόριθμοι Χαμηλού Επιπέδου

Περιεχόμενα Κεφαλαίου

Στο Κεφάλαιο 1.4 αναφέραμε το περιβάλλον εργασίας μας. Δηλαδή, ότι υποθέτουμε ένα μοντέλο μηχανής RAM, μίας μηχανής με έναν επεξεργαστή όπου οι εντολές εκτελούνται σειριακά. Αναφερθήκαμε επίσης στην ποικιλία των στοιχειωδών εντολών που για λόγους απλοποίησης θεωρούμε ότι έχουν μοναδιαίο κόστος. Έτσι η πολυπλοκότητα ενός αλγορίθμου εξαρτάται από το πλήθος των στοιχειωδών πράξεων που καταμετρούμε, ώστε να φθάσουμε σε μία έκφραση συναρτήσει του μεγέθους της εισόδου n. Στο σημείο αυτό θα ανοίξουμε μία παρένθεση για να διερευνήσουμε περαιτέρω τις υποθέσεις αυτές.

6.1 Αριθμητικοί Αλγόριθμοι

Σε περιπτώσεις αναζήτησης ή ταξινόμησης είναι λογικό να θεωρούμε ως μέγεθος του προβλήματος το αντίστοιχο πλήθος των στοιχείων εισόδου. Επίσης σε περιπτώσεις διάσχισης ενός δένδρου ή ενός γράφου είναι λογικό να θεωρούμε ως μέγεθος του προβλήματος το πλήθος των κορυφών ή/και το πλήθος των ακμών. Ωστόσο, για ένα μεγάλο σύνολο αλγορίθμων που αφορούν σε αριθμητικές πράξεις, όπως για παράδειγμα η παραγοντοποίηση ενός αριθμού, η εύρεση του μέγιστου κοινού διαιρέτη, οι πράξεις modulo, ο έλεγχος για πρώτους αριθμούς, ο πολλαπλασιασμός πινάκων ή ακόμη απλούστερα ο πολλαπλασιασμός δύο ακεραίων, προκύπτει ότι η καλύτερη μέθοδος για την εύρεση της πολυπλοκότητας είναι η θεώρηση του πλήθους των bits που είναι απαραίτητα για την αναπαράσταση των αντίστοιχων αριθμών σε δυαδική μορφή. Χωρίς την ανάλυση αυτών των αλγορίθμων σε χαμηλό επίπεδο, δηλαδή σε επίπεδο πράξεων bit, δεν θα μπορούσε να γίνει μία αντικειμενική ανάδειξη της πολυπλοκότητας του αντίστοιχου αλγορίθμου. Το σκεπτικό αυτό θα γίνει κατανοητό καλύτερα με το παράδειγμα.

Στο Κεφάλαιο 5.5 παρουσιάσαμε τον επόμενο αλγόριθμο `factorial2` για τον υπολογισμό του παραγοντικού για $n \geq 0$ και καταλήξαμε ότι η πολυπλοκότητά του είναι γραμμική $\Theta(n)$. Ωστόσο, στο σημείο εκείνο έγινε σιωπηρά η υπόθεση ότι κάθε πράξη πολλαπλασιασμού (εντολή 2) απαιτεί σταθερό χρόνο εκτέλεσης ανεξαρτήτως του μεγέθους των τελεστών. Όμως είναι αυτό ρεαλιστικό, δηλαδή να θεωρούμε σταθερό το κόστος πολλαπλασιασμού δύο αριθμών καθώς μας ενδιαφέρει το πλήθος n των δεδομένων εισόδου και όχι το μέγεθος εκάστου ενός των n δεδομένων εισόδου;

```
     function factorial2(n);
1.   product <-- 1;
2.   for i <-- 1 to n do product <-- i*product;
3.   return product
```

6.2 Κόστος Στοιχειωδών Πράξεων

Αρχικά θα εξετάσουμε και θα θυμηθούμε τη λειτουργία των τεσσάρων βασικών πράξεων (πρόσθεση, αφαίρεση, πολλαπλασιασμός και διαίρεση), ώστε να συναγάγουμε το κόστος τους θεωρώντας το μέγεθος των αριθμών που υπεισέρχονται σε κάθε πράξη. Στην επόμενη παράγραφο, θα χρησιμοποιήσουμε το υλικό αυτής της παραγράφου για να μελετήσουμε σύνθετους αριθμητικούς αλγορίθμους.

Δεδομένου ενός ακεραίου αριθμού n ισχύει ότι $2^{k-1} \leq n < 2^k$. Επομένως, για να αναπαραστήσουμε έναν ακέραιο σε δυαδική μορφή χρειαζόμαστε k bits, όπου $k = \lfloor \log n \rfloor + 1$.

Πρόσθεση Ακεραίων

Από όλα τα κλασικά διδακτικά βιβλία Αρχιτεκτονικής Υπολογιστών γνωρίζουμε τι είναι οι αθροιστές (adders). Για παράδειγμα, έστω ότι δίνονται δύο ακέραιοι αριθμοί υπό δυαδική μορφή $a_{k-1}a_{k-2}...a_0$ και $b_{k-1}b_{k-2}...b_0$ που αντιστοιχούν στους ακεραίους $a_{k-1}2^{k-1} + a_{k-2}2^{k-2} + \ldots + a_0 2^0$ και $b_{k-1}2^{k-1} + b_{k-2}2^{k-2} + \ldots + b_0 2^0$ υπό δεκαδική μορφή. Ο i-οστός αθροιστής που δέχεται εισόδους τα a_i, b_i και c_{i-1}, όπου c_{i-1} είναι το κρατούμενο από τον προηγούμενο αθροιστή. Στο Σχήμα 6.2 παρουσιάζεται η πρόσθεση δύο ακεραίων σε δεκαδικό και δυαδικό σύστημα. Είναι προφανής η σάρωση των δυαδικών αριθμών από τα δεξιά προς τα αριστερά. Συνεπώς, η πολυπλοκότητα της πρόσθεσης είναι γραμμική ως προς το μήκος σε bits των τελεσταίων, δηλαδή $\Theta(k)$, με μία κρυμμένη σταθερά ίση με 2 λόγω του κρατούμενου c_i.

```
 8          1000
+3         +0011
--          ----
11          1011
```

Σχήμα 6.1: Πρόσθεση αριθμών σε δεκαδικό και δυαδικό σύστημα.

Αφαίρεση Ακεραίων

Η πράξη της αφαίρεσης είναι παρόμοια προς την πράξη της πρόσθεσης και στηρίζεται στο συμπλήρωμα. Στο Σχήμα 6.2 παρουσιάζεται η αντίστοιχη διαδικασία. Το συμπλήρωμα, όπως και η διαδικασία της πρόσθεσης, ουσιαστικά είναι μία σάρωση, άρα είναι μία γραμμική διαδικασία, ενώ η τελική πρόσθεση του άσσου έχει σταθερό κόστος. Συνεπώς, τελικά η πολυπλοκότητα είναι και πάλι γραμμική, δηλαδή $\Theta(k)$.

	αφαίρεση αριθμών	πρόσθεση συμπληρώματος	πρόσθεση μονάδας
8	1000	1000	0100
-3	-0011	+1100	+1
--	----	----	----
5	1011	0100	0101

Σχήμα 6.2: Αφαίρεση αριθμών σε δεκαδικό και δυαδικό σύστημα.

Πολλαπλασιασμός Ακεραίων
Έστωσαν δύο ακέραιοι αριθμοί υπό δυαδική μορφή $a_{k-1}a_{k-2}...a_0$ και $b_{k-1}b_{k-2}...b_0$ που δίνονται προς πολλαπλασιασμό. Οι αριθμοί τοποθετούνται στους καταχωρητές A και B, αντίστοιχα, ενώ στον καταχωρητή P θα τοποθετηθεί το γινόμενό τους. Ο πολλαπλασιασμός θα γίνει με τα εξής δύο βήματα:

1. αν το λιγότερο σημαντικό ψηφίο του A είναι 1, τότε ο καταχωρητής B προστίθεται στο P, αλλιώς προστίθεται το 00...00. Το άθροισμα τίθεται πάλι στον καταχωρητή P.

2. οι καταχωρητές A και P ολισθαίνουν προς τα δεξιά, ενώ το κρατούμενο του αθροίσματος μετακινείται στο υψηλότερο (αριστερότερο) bit του καταχωρητή P και το χαμηλότερο bit του P μετακινείται στον καταχωρητή A, όπου το δεξιότερο bit διολισθαίνει προς τα έξω καθώς δεν θα χρησιμοποιηθεί στη συνέχεια.

Μετά από n βήματα, το γινόμενο εμφανίζεται στους καταχωρητές P και A, όπου ο A αποθηκεύει τα λιγότερο σημαντικά ψηφία του αποτελέσματος. Το Σχήμα 6.2 δείχνει τον πολλαπλασιασμό δύο ακεραίων, των A=6 και B=3, τους οποίους παριστούμε με τους αντίστοιχους δυαδικούς 110 και 011. Εφ'όσον το μήκος των αριθμών σε bits είναι 3, έπεται ότι θα υπάρξει ένας βρόχος τριών επαναλήψεων των δύο βημάτων. Το τελικό αποτέλεσμα είναι 010010, δηλαδή 18. Παρατηρούμε ότι για κάθε bit του A εκτελείται μία πρόσθεση όλων των bits του B. Συνεπώς, η πολυπλοκότητα του πολλαπλασιασμού δύο ακεραίων που αναπαριστώνται με k_1 και k_2 bits αντίστοιχα είναι $\Theta(k_1k_2)$.

Διαίρεση Ακεραίων
Στην πράξη της διαίρεσης δύο δυαδικών αριθμών, με μία διαδικασία παρόμοια προς την προηγούμενη, οι αριθμοί τοποθετούνται στους καταχωρητές A και B,

	A	B	P
Βήμα 1 (πρόσθεση B στο P)	110	011	000
Βήμα 2 (ολίσθηση δεξιά)	011	011	000
Βήμα 1 (πρόσθεση B στο P)	011	011	011
Βήμα 2 (ολίσθηση δεξιά)	101	011	001
Βήμα 1 (πρόσθεση B στο P)	101	011	100
Βήμα 2 (ολίσθηση δεξιά)	010	011	010

Σχήμα 6.3: Πολλαπλασιασμός αριθμών σε δυαδικό σύστημα.

ενώ το αποτέλεσμα τοποθετείται στον καταχωρητή P. Η διαίρεση θα γίνει με τα εξής τέσσερα βήματα:

1. ολισθαίνουμε το ζεύγος των καταχωρητών A και P μία θέση προς τα αριστερά.

2. αφαιρούμε το περιεχόμενο του καταχωρητή B από τον καταχωρητή P και τοποθετούμε το αποτέλεσμα στον καταχωρητή P.

3. αν το αποτέλεσμα του προηγούμενου βήματος είναι αρνητικό, τότε βάζουμε 0 στο χαμηλότερο ψηφίο του A, αλλιώς βάζουμε το 1.

4. αν το αποτέλεσμα του ιδίου βήματος είναι αρνητικό, τότε αποκαθιστούμε την παλιά τιμή του P προσθέτοντας το περιεχόμενο του B στο P.

Για παράδειγμα, έστω ότι θέλουμε να διαιρέσουμε τους αριθμούς A=14=1110_2 και B=3=11_2. Το Σχήμα 6.2 παρουσιάζει την αλληλουχία των βημάτων. Στο τέλος των τεσσάρων (όσο το μήκος του A) επαναλήψεων των τεσσάρων βημάτων στον καταχωρητή A είναι αποθηκευμένο το πηλίκο (δηλαδή 4), ενώ στον καταχωρητή P είναι αποθηκευμένο το υπόλοιπο (δηλαδή 2).

Στο παράδειγμα αυτό, σαρώνουμε k_1 φορές (δηλαδή, όσο το μέγεθος σε bits του καταχωρητή A) τον καταχωρητή P μεγέθους k_2 bits (δηλαδή, όσο το μέγεθος σε bits του καταχωρητή B). Συνεπώς, όπως και στην περίπτωση του πολλαπλασιασμού, η πολυπλοκότητα της διαίρεσης δύο ακεραίων που αναπαριστώνται με k_1 και k_2 bits αντίστοιχα είναι $\Theta(k_1 k_2)$.

	P	A	B
Αρχική κατάσταση	00	1110	11
Βήμα 1 (ολίσθηση αριστερά)	01	110	11
Βήμα 2 (αφαίρεση B από P)	-10	110	11
Βήμα 3 (χαμηλό bit του A)	-10	1100	11
Βήμα 4 (αποκατάσταση P)	01	1100	11
Βήμα 1 (ολίσθηση αριστερά)	11	100	11
Βήμα 2 (αφαίρεση B από P)	00	100	11
Βήμα 3 (χαμηλό bit του A)	00	1001	11
Βήμα 1 (ολίσθηση αριστερά)	01	001	11
Βήμα 2 (αφαίρεση B από P)	-10	001	11
Βήμα 3 (χαμηλό bit του A)	-10	0010	11
Βήμα 4 (αποκατάσταση P)	01	0010	11
Βήμα 1 (ολίσθηση αριστερά)	10	010	11
Βήμα 2 (αφαίρεση B από P)	-01	010	11
Βήμα 3 (χαμηλό bit του A)	-01	0100	11
Βήμα 4 (αποκατάσταση P)	10	0100	11

Σχήμα 6.4: Διαίρεση αριθμών σε δυαδικό σύστημα.

6.3 Κόστος Βασικών Αλγορίθμων

Στην παράγραφο αυτή θα επανεξετάσουμε αλγορίθμους που ήδη έχουμε μελετήσει στα προηγούμενα μαθήματα, αλλά τώρα η επανεξέταση θα γίνει κάτω από το νέο πρίσμα για την κοστολόγηση των πράξεων.

Υπολογισμός Δύναμης

Θέλουμε να υπολογίσουμε τη δύναμη a^b, όπου τα a, b είναι ακέραιοι αριθμοί. Ας υποθέσουμε ότι θα εφαρμόσουμε τη γνωστή συνάρτηση `power1`, που την υπενθυμίζουμε στη συνέχεια, ώστε να αντιληφθούμε το μηχανισμό του υπολογισμού αυτού.

```
     function power1(a,b)
1.   power <-- 1;
2.   for i <-- 1 to b do power1 <-- power1*a
3.   return power1
```

Με βάση όσα αναφέραμε προηγουμένως για το κόστος του πολλαπλασιασμού προκύπτει ότι για να υπολογίσουμε το τετράγωνο a^2 χρειαζόμαστε $\Theta((\log a)^2)$

πράξεις σε bit, ενώ το μέγεθος του αποτελέσματος έχει μέγεθος $2 \log a$ bits. Ομοίως, για τον υπολογισμό του a^3 θα εκτελέσουμε έναν πολλαπλασιασμό μεταξύ δύο αριθμών: ενός αριθμού μεγέθους $\log a$ bits και ενός αριθμού μεγέθους $2 \log a$ bits. Συνεπώς, το αντίστοιχο υπολογιστικό κόστος θα είναι $\Theta((\log a)^2)$ πράξεις με bit (με κρυμμένη σταθερά 2), ενώ το αποτέλεσμα της πράξης θα αποθηκευθεί σε χώρο $3 \log a$ bits. Συνεχίζοντας το σκεπτικό αυτό καταλήγουμε στο συμπέρασμα ότι για τον υπολογισμό του a^b θα εκτελέσουμε έναν πολλαπλασιασμό μεταξύ δύο αριθμών: ενός αριθμού μεγέθους $\log a$ bits και ενός αριθμού $(b-1) \log a$ bits. Συνεπώς η χρονική πολυπλοκότητα της ύψωσης σε δύναμη, a^b, είναι $\Theta(b\ (\log a)^2)$, ενώ η χωρική πολυπλοκότητα είναι $\Theta(b(\log a)^2)$.

Υπολογισμός Αθροίσματος
Θέλουμε να υπολογίσουμε το άθροισμα $\sum_{i=1}^{n} i^2 = n(n+1)(2n+1)/6$. Αρχικά θεωρούμε το αριστερό σκέλος. Με βάση τα προηγούμενα, προκύπτει ότι το κόστος υπολογισμού του ενός τετραγώνου είναι $\Theta((\log n)^2)$, και επομένως το συνολικό κόστος για τον υπολογισμό όλων των n τετραγώνων του αριστερού σκέλους θα είναι $\Theta(n\ (\log n)^2)$. Καθώς το κόστος των προσθέσεων $n-1$ προσθέσεων είναι μικρότερο από το κόστος των πολλαπλασιασμών, προκύπτει ότι το προηγούμενο κόστος είναι και το τελικό σε σχέση με το αριστερό σκέλος. Απεναντίας, το κόστος υπολογισμού του δεξιού σκέλους είναι $\Theta((\log n)^2)$, γεγονός που εξηγείται με βάση το γεγονός ότι απλώς εκτελούνται 3 πολλαπλασιασμοί και μία διαίρεση.

Πολλαπλασιασμός Πινάκων
Ας υποθέσουμε ότι πρέπει να πολλαπλασιάσουμε δύο τετραγωνικούς πίνακες $n \times n$, ενώ το μέγεθος των στοιχείων των πινάκων είναι μικρότερο από $m > 0$. Ο πίνακας που θα προκύψει από τον πολλαπλασιασμό θα έχει επίσης n^2 στοιχεία, κάθε ένα από τα οποία για να υπολογισθεί απαιτεί n πολλαπλασιασμούς και $n-1$ προσθέσεις. Επειδή κάθε πολλαπλασιασμός απαιτεί $(\log m)^2$ πράξεις με bit, ενώ κάθε πρόσθεση απαιτεί $2 \log m$ πράξεις με bit, έπεται ότι το συνολικό κόστος είναι $(n^2\ (n(\log m)^2 + 2(n-1) \log m))$ πράξεις με bits, και επομένως η πολυπλοκότητα είναι $\Theta(n^3((\log m)^2 + \log m))$.

Υπολογισμός Συνδυασμού
Θεωρούμε τη γνωστή την Εξίσωση 2.1 για τον υπολογισμό του συνδυασμού των n ανά m αντικειμένων:

$$\binom{n}{m} = \frac{n \times (n-1) \times (n-2) \times \ldots \times (n-m+1)}{1 \times 2 \times 3 \times \ldots \times m}$$

και απο το Κεφάλαιο 5.5 χρησιμοποιούμε τη γνωστή συνάρτηση comb4, που υπενθυμίζουμε στη συνέχεια.

```
     function comb4(n,m);
1.   if m>n-m then m <-- n-m
2.   t1 <-- 1;
3.   for i <-- n downto n-m+1 do t1 <-- t1*i;
4.   t2 <-- factorial2(m);
5.   return t1/t2
```

Σύμφωνα με τον αλγόριθμο αυτό αρχικά εκτελούνται $m-1$ πολλαπλασιασμοί για τον αριθμητή, στη συνέχεια άλλοι $m-1$ πολλαπλασιασμοί για τον παρονομαστή και τελικά μία διαίρεση. Αν υποθέσουμε ότι το μέγεθος του n είναι $k = \log n$ bits, τότε με τη λογική που αντιμετωπίσαμε το πρόβλημα του υπολογισμού της ύψωσης σε δύναμη θα έχουμε τον εξής μηχανισμό.

Για να εκτελέσουμε τον πρώτο πολλαπλασιασμό χρειαζόμαστε $\Theta(k^2)$ πράξεις σε bit, ενώ το μέγεθος του αποτελέσματος θα έχει μέγεθος $2k$ bits. Για το δεύτερο πολλαπλασιασμό θα θεωρήσουμε έναν αριθμό μεγέθους k bits και έναν αριθμό μεγέθους $2k$ bits. Συνεπώς το αντίστοιχο κόστος θα είναι $\Theta(k^2)$ πράξεις με bit (με κρυμμένη σταθερά 2), ενώ το αποτέλεσμα της πράξης θα αποθηκευθεί σε χώρο $3k$ bits. Συνεχίζοντας το σκεπτικό αυτό καταλήγουμε στο συμπέρασμα ότι για την εκτέλεση του $(m-1)$-οστού πολλαπλασιασμού θα θεωρήσουμε έναν αριθμό μεγέθους k bits και έναν αριθμό μεγέθους $(m-1)k$ bits. Συνεπώς η χρονική πολυπλοκότητα για τον υπολογισμό του αριθμητή θα είναι $k^2(m-1)^2$, δηλαδή ισχύει η πολυπλοκότητα $\Theta(k^2\ m^2)$. Το κόστος του υπολογισμού του παρονομαστή (εντολή 4) είναι ασφαλώς φραγμένο από επάνω από την ίδια πολυπλοκότητα του υπολογισμού του αριθμητή, δηλαδή $\Theta(k^2m^2)$. Επομένως αυτή είναι και η τελική απάντηση σε σχέση με τη ζητούμενη πολυπλοκότητα.

Μετατροπή Δυαδικού σε Δεκαδικό Αριθμό

Ας υποθέσουμε ότι δίνεται ένας ακέραιος n υπό τη δυαδική μορφή $b_{k-1}b_{k-2}...b_2b_1b_0$ και επιθυμούμε να βρούμε την ισοδύναμή του έκφραση στο δεκαδικό σύστημα. Θα μπορούσε να εφαρμοσθεί ένας αλγόριθμος της επόμενης μορφής.

```
     function bin2dec
1.   num <-- b_{k-1};
2.   for i <-- k-2 downto 0 do num <-- 2*num + b_i;
```

Καθώς κάθε μερικό αποτέλεσμα είναι μεγέθους το πολύ k bits και ο βρόχος εκτελείται $k-1$ φορές, έπεται ότι ο συνολικός αριθμός πράξεων με bit είναι $\Theta(k^2) = \Theta((\log n)^2)$.

Εύρεση Μέγιστου Κοινού Διαιρέτη
Στο Κεφάλαιο 5.1 μελετήσαμε διεξοδικά τον αλγόριθμο του Ευκλείδη για την εύρεση του μέγιστου κοινού διαιρέτη μεταξύ δύο ακεραίων $n > m$. Στο σημείο αυτό υπενθυμίζουμε τον αλγόριθμο.

```
     function euclid(m,n)
1.   while m>0 do
2.      t <-- n mod m
3.      n <-- m
4.      m <-- t
5.   return n
```

Για να προχωρήσουμε σε μία εκτίμηση κόστους, κατ'αρχήν πρέπει να παρατηρήσουμε ότι στις διαδοχικές διαιρέσεις παράγονται συνεχώς μικρότερα υπόλοιπα. Επίσης, στο Κεφάλαιο 5.1 παρατηρήσαμε ότι $m_i < m_{i-2}/2$. Με λίγα λόγια, μετά από κάθε δεύτερο βήμα το υπόλοιπο υποδιαιρείται και επομένως στη χειρότερη περίπτωση θα εκτελεσθούν $2\log m$ διαιρέσεις. Καθώς σε κάθε τέτοιο βήμα οι αριθμοί που υπεισέρχονται στις διαιρέσεις αυτές είναι μικρότεροι του m, έπεται ότι κάθε τέτοια πράξη απαιτεί στη χειρότερη περίπτωση $\mathrm{O}((\log m)^2)$. Επομένως, τελικά η πολυπλοκότητα είναι $\mathrm{O}((\log m)^3)$. Σημειώνουμε ότι με μία πιο σφικτή ανάλυση μπορεί να προκύψει ότι η πολυπλοκότητα είναι $\mathrm{O}((\log m)^2)$.

Εύρεση Πρώτων Αριθμών
Από την Άσκηση 1.12 γνωρίζουμε ότι για την εύρεση των πρώτων αριθμών από 1 ως n, μπορεί να χρησιμοποιηθεί το κόσκινο του Ερατοσθένη. Ο μικρότερος πρώτος αριθμός είναι 2. Έτσι, αγνοούμε όλα τα πολλαπλάσια του 2 μέχρι την τιμή n. Από τους αριθμούς που απομένουν ο μικρότερος είναι και πάλι πρώτος, δηλαδή το 3. Έτσι αγνοούμε όλα τα πολλαπλάσια του 3 μέχρι την τιμή n. Σύμφωνα με το επόμενο θεώρημα, η διαδικασία αυτή πρέπει να συνεχισθεί για όλους τους ακεραίους μέχρι το $\sqrt{n}$, καθώς αυτό το όριο αρκεί για τον Ερατοσθένη.

Θεώρημα.
Αν το n είναι θετικός σύνθετος ακέραιος, τότε ο n έχει έναν πρώτο αριθμό που δεν ξεπερνά το $\sqrt{n}$.

Απόδειξη.
Αν το n είναι θετικός σύνθετος ακέραιος, τότε ισχύει $n = a \times b$, όπου $0 < a \leq$

$b < n$. Συνεπώς $a \leq \sqrt{n}$ γιατί αλλιώς θα έπρεπε $a \times b > n$.

Ο έλεγχος ενός αριθμού για την εύρεση των πρώτων αριθμών του με τη μέθοδο του Ερατοσθένη είναι πολύ αναποτελεσματικός. Αν με $\Pi(x)$ συμβολίζουμε το πλήθος των πρώτων αριθμών που δεν υπερβαίνουν τον αριθμό x, τότε για να ελέγξουμε αν ο x είναι πρώτος αρκεί να εκτελέσουμε $\Pi(\sqrt{x})$ διαιρέσεις. Έχει αποδειχθεί ότι ασυμπτωτικά ισχύει $\Pi(x) = x/\log x$ και επομένως το πλήθος των πρώτων αριθμών που δεν υπερβαίνουν τον αριθμό $\sqrt{n}$ είναι $\sqrt{n}/\log\sqrt{n}$. Καθώς κάθε τέτοια διαίρεση απαιτεί $\Theta((\log n)^2)$ πράξεις με bit, είναι ευνόητο ότι συνολικά απαιτούνται $\Theta(\sqrt{n}\log n)$ πράξεις με bit, με την προϋπόθεση ότι διαθέτουμε μία λίστα με όλους τους πρώτους αριθμούς που δεν υπερβαίνουν την τιμή $\sqrt{n}$. Η πολυπλοκότητα αυτή είναι εκθετική ως προς $k = \lfloor \log n \rfloor + 1$, ενώ αν διεκπεραιώναμε το πρόβλημα θεωρώντας μία συμβατική προσέγγιση για την ανάλυση του αλγορίθμου, τότε θα καταλήγαμε σε μία γραμμική πολυπλοκότητα.

6.4 Βιβλιογραφική Συζήτηση

Η οπτική γωνία αυτού του κεφαλαίου δεν είναι η συνήθης στη διεθνή βιβλιογραφία. Η αντιμετώπιση της σχεδίασης και ανάλυσης αλγορίθμων γενικώς βασίζεται σε μία περισσότερο "υψηλή" προσέγγιση. Το υλικό του κεφαλαίου αυτού στηρίζεται σε μεγάλο βαθμό στα άρθρα [20, 21] και για την κατανόησή του απαιτούνται στοιχειώδεις γνώσεις αρχιτεκτονικής. Το βιβλίο των Dasgupta-Παπαδημητρίου-Vazirani αφιερώνει σημαντικό χώρο στο αντικείμενο [28]. Σημειώνεται ότι η παρουσίαση του σχετικού υλικού στο τελευταίο βιβλίο γίνεται με στόχο την ανάπτυξη των θεμελιώσεων της κρυπτογραφίας [153].

6.5 Ασκήσεις

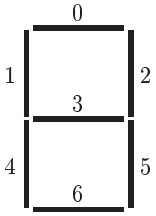
1. Να αποδειχθεί ότι σε κάθε αριθμητικό σύστημα με βάση $b \geq 2$, το άθροισμα 3 οποιωνδήποτε μονοψήφιων αριθμών είναι αριθμός διψήφιος το πολύ.

2. Να αποδειχθεί ότι κάθε αριθμός στο δυαδικό σύστημα έχει τετραπλάσιο το πολύ μήκος από τον αντίστοιχο δεκαδικό. Ποιό είναι το όριο αυτών των δύο μηκών για μεγάλους αριθμούς;

3. Έστω ότι ένας ακέραιος αριθμός των n bits λαμβάνει οποιαδήποτε τιμή στο διάστημα $[0..2^n - 1]$ με την ίδια πιθανότητα. Να υπολογισθούν τα εξής:

- Ποιά είναι η πιθανότητα για την i-οστή θέση (όπου $1 \leq i \leq n$) να είναι 0 και ποιά να είναι 1;
- Ποιά είναι η μέση τιμή του πλήθους των '1' στα n bits;
- Ποιά είναι η προσδοκητή τιμή της θέσης του αριστερότερου '1';

4. Στις ψηφιακές οθόνες κάθε ακέραιος παρίσταται με ένα συνδυασμό επτά φωτεινών γραμμών. Οι γραμμές αυτές αριθμούνται από 0 ως 6 όπως φαίνεται στο Σχήμα 6.5. Να σχεδιασθεί και να αναλυθεί ένας αλγόριθμος που να δέχεται ένα θετικό ακέραιο των 16 bits και να δίνει ένα πίνακα από 5 bytes. Το i-οστό bit του j-οστού byte θα πρέπει να είναι 1, αν και μόνο αν η i-οστή γραμμή της παράστασης είναι φωτεινή.



Σχήμα 6.5: Το πρόβλημα του πίνακα matrix.

5. Να σχεδιασθεί και να αναλυθεί αλγόριθμος υπολογισμού του συμπληρώματος ως προς 2 ενός ακεραίου που βρίσκεται στο διάστημα $[-2^{n-1}, 2^{n-1} - 1]$.

6. Δίνονται δύο αριθμοί $(a \mod N)$ και $(b \mod N)$. Ποιά είναι η πολυπλοκότητα της πρόσθεσης και του πολλαπλασιασμού τους αν $n = \lceil N \rceil$;

7. Σε κρυπτογραφικές εφαρμογές απαιτείται ο υπολογισμός του $a^b \mod N$, όπου συχνά τα a, b, N μπορεί να έχουν μήκος 100άδες bits. Να σχεδιασθεί και να αναλυθεί σχετικός αλγόριθμος εφαρμόζοντας διαδοχικούς b υπολογισμούς του $a \mod N$. Να σχολιασθεί και η χρονική αλλά και η χωρική πολυπλοκότητα.

8. Ο ΜΚΔ δύο ακεραίων μπορεί να υπολογισθεί επίσης σύμφωνα με τον τύπο:

$$gcd(a,b) = \begin{cases} 2\ gcd(a/2, b/2) & \text{αν τα } a, b \text{ ειναι αρτια} \\ gcd(a, b/2) & \text{αν το } a \text{ ειναι περιττο και το } b \text{ αρτιο} \\ gcd((a-b)/2, b) & \text{αν τα } a, b \text{ ειναι περιττα} \end{cases}$$

Να σχεδιασθεί ένας αλγόριθμος που να υλοποιεί αυτόν τον τύπο. Να συγκριθεί η επίδοση αυτού του αλγορίθμου σε σχέση με αυτήν του αλγορίθμου του Ευκλείδη, υπό την προϋπόθεση ότι οι αριθμοί a και b είναι ακέραιοι των n bits.

9. Με βάση τη λογική του ψευδοκώδικα `power4(a,b)` του Κεφαλαίου 5.4 διατυπώνουμε τη σχέση:

$$a^b = \begin{cases} \left(a^{\lceil b/2 \rceil}\right)^2 & \text{αν το } b \text{ ειναι αρτιο} \\ a \cdot \left(a^{\lceil b/2 \rceil}\right)^2 & \text{αν το } b \text{ ειναι περιττο} \end{cases}$$

Η σχέση αυτή να χρησιμοποιηθεί για τη σχεδίαση και την ανάλυση αλγορίθμου που να βασίζεται στη λογική Διαίρει και Βασίλευε και να υπολογίζει το $a^b \mod N$. Να γίνει σύγκριση με τον αλγόριθμο της Άσκησης 8 ως προς τη χρονική αλλά και τη χωρική πολυπλοκότητα.

10. Με βάση τη λογική του ψευδοκώδικα `factorial2(n)` του Κεφαλαίου 5.5 να διατυπωθεί η πολυπλοκότητα του υπολογισμού του $n!$, ως συνάρτηση του μήκους της δυαδικής αναπαράστασης του n. Πόσα bits απαιτούνται για την αναπαράσταση του $n!$.

11. Να αποδειχθεί ότι αν $N = a^b$ (όπου όλα τα a, b, N είναι θετικοί ακέραιοι), τότε είτε $b \leq \log N$ είτε $N = 1$. Να σχεδιασθεί και να αναλυθεί αλγόριθμος που να διαπιστώνει αν ένας θετικός ακέραιος είναι δύναμη.

12. Να σχεδιασθεί και να αναλυθεί αλγόριθμος εύρεσης του ΕΚΠ δύο θετικών ακεραίων μεγέθους n bits.

7

Αλγοριθμικές Τεχνικές

Περιεχόμενα Κεφαλαίου

Υπάρχουν πολλές οπτικές γωνίες κατά το σχεδιασμό σωστών προγραμμάτων. Ο σχεδιασμός καλών και σωστών αλγορίθμων είναι μία από αυτές τις οπτικές γωνίες. Είναι απαραίτητο κατά το σχεδιασμό αλγορίθμων να έχουμε πάντοτε υπόψη μερικές στρατηγικές για την επίλυση του κάθε φορά προβλήματος. Από το μάθημα της Σχεδίασης Αλγορίθμων ήδη γνωρίζουμε μία σειρά τεχνικών/οικογενειών που βοηθούν στο σκοπό αυτό. Για παράδειγμα, γνωρίζουμε τις οικογένειες και τα αντίστοιχα προβλήματα:

Ωμή βία (brute force). Εξαντλητική παραγωγή μεταθέσεων, εξαντλητική παραγωγή συνδυασμών κλπ.

Άπληστη Μέθοδος (greedy method). Ελάχιστα ζευγνύοντα μονοπάτια (Prim, Kruskal), Συντομότερα μονοπάτια (Dijkstra), Κωδικοποίηση Huffmann, το Πρόβλημα του σάκου, το Πρόβλημα της αποθήκευσης σε ταινίες, κλπ.

Διαίρει και Βασίλευε (divide and conquer). Δυαδική αναζήτηση, Ταξινόμηση με συγχώνευση, Γρήγορη ταξινόμηση, το Πρόβλημα της επιλογής, Πολλαπλασιασμός πινάκων (Strassen), κλπ.

Δυναμικός Προγραμματισμός (dynamic programming). Αλυσιδωτός πολλαπλασιασμός πινάκων, Συντομότερα μονοπάτια (Floyd), Βέλτιστα δυαδικά δένδρα, Περιοδεύων πωλητής (Hamilton), Μακρύτερη κοινή υποακολουθία, κλπ.

Οπισθοδρόμησης (backtracking). Διάσχιση γράφων, το πρόβλημα των βασιλισσών, Περιοδεύων πωλητής (Hamilton), κλπ.

Διακλάδωση και περιορισμός (branch and bound). Το πρόβλημα των βασιλισσών, Περιοδεύων πωλητής (Hamilton), κλπ.

Στο κεφάλαιο αυτό θα εξετάσουμε 3 προβλήματα, που θα προσπαθήσουμε να επιλύσουμε με εναλλακτικούς τρόπους, όπου κάθε φορά θα προχωρούμε σε εξυπνότερες επιλογές και αποδοτικότερους αλγορίθμους.

7.1 Μέγιστο Άθροισμα Υποακολουθίας

Δίνεται ένας πίνακας με ακέραιους αριθμούς και ζητείται να βρεθεί ο υποπίνακας εκείνος, του οποίου τα άθροισμα των τιμών των στοιχείων είναι μέγιστο σε σχέση με κάθε άλλο υποπίνακα που μπορεί να θεωρηθεί. Για παράδειγμα, στον Πίνακα 7.1 το μέγιστο άθροισμα των τιμών των στοιχείων δίνεται από τον υποπίνακα `A[3..5]` και είναι ίσο με 19. Για παράδειγμα, στον επόμενο πίνακα

6	-8	7	10	2	-4	5

Πίνακας 7.1: Μέγιστο άθροισμα υποακολουθίας.

`A` το μέγιστο άθροισμα των τιμών των στοιχείων δίνεται από τον υποπίνακα `A[3..5]` και είναι ίσο με 19.

Στο πρόβλημα αυτό υπάρχουν δύο προφανείς λύσεις. Αν ο πίνακας περιέχει μόνο θετικούς αριθμούς, τότε η λύση είναι ολόκληρος ο πίνακας. Όταν ο πίνακας περιέχει μόνο αρνητικούς αριθμούς τότε η λύση είναι ένας υποπίνακας με 0 στοιχεία. Σε κάθε άλλη περίπτωση, το πρόβλημα θέλει ιδιαίτερη προσοχή. Στη συνέχεια θα εξετάσουμε τέσσερις αλγορίθμους, όπου διαδοχικά θα βελτιώνουμε την πολυπλοκότητά τους.

Πρώτος Αλγόριθμος (ωμή βία)
Κατ' αρχήν παρουσιάζεται η πρώτη μας προσπάθεια: ένας προφανής τρόπος επίλυσης του προβλήματος, όπου εξετάζεται κάθε δυνατή περίπτωση υποπίνακα. Η συνάρτηση `max_subseq_sum1` που ακολουθεί, δέχεται τον πίνακα `A`, που αποτελείται από n στοιχεία, και επιστρέφει τη ζητούμενη μέγιστη τιμή. Με `left` και `right` συμβολίζονται τα δύο ακραία στοιχεία της υποακολουθίας, που κάθε φορά εξετάζεται. Για κάθε εξεταζόμενη υποακολουθία, το αντίστοιχο άθροισμα των τιμών των στοιχείων της υπολογίζεται με τη βοήθεια της μεταβλητής `current_sum` που συγκρίνεται με τη μεταβλητή `max_sum` και την αντικαθιστά αν έχει μεγαλύτερη τιμή.

```
     function max_subseq_sum1;
1.   max_sum <-- 0; left <-- 0; right <-- 0;
2.   for i <-- 1 to n do
3.      for j <-- i to n do
4.         current_sum <-- 0;
5.         for k <-- i to j do
6.            current_sum <-- current_sum+A[k];
7.         if (current_sum>max_sum) then
8.            max_sum <-- current_sum;
9.            left <-- i; right <-- j
10.  return max_sum
```

Η συνάρτηση αυτή αποτελείται από τρεις φωλιασμένες εντολές `for`. Η εύρεση της πολυπλοκότητας αναδεικνύεται εύκολα όπως φαίνεται στη συνέχεια

θεωρώντας ως βαρόμετρο την εντολή 6 (με μοναδιαίο κόστος), που βρίσκεται στο εσωτερικότερο σημείο των βρόχων.

$$\begin{aligned}
T(n) &= \sum_{i=1}^{n}\sum_{j=i}^{n}\sum_{k=i}^{j} 1 \;=\; \sum_{i=1}^{n}\sum_{j=i}^{n}(j-i+1) \\
&= \frac{1}{2}\sum_{i=1}^{n}(n-i+1)(n-i+2) \\
&= \frac{1}{2}\sum_{i=1}^{n} i^2 \;-\; \left(n+\frac{3}{2}\right)\sum_{i=1}^{n} i \;+\; \frac{1}{2}(n^2+3n+2)\sum_{j=1}^{n} 1 \\
&= \frac{1}{2}\frac{n(n+1)(2n+1)}{6} \;-\; \left(n+\frac{3}{2}\right)\frac{n(n+1)}{2} \;+\; \frac{n}{2}(n^2+3n+2) \\
&= \frac{n^3+3n^2+2n}{6}
\end{aligned}$$

Έτσι προκύπτει ότι η πολυπλοκότητα αυτού του αλγορίθμου είναι $\Theta(n^3)$.

Δεύτερος Αλγόριθμος (ωμή βία)
Παρατηρώντας τον προηγούμενο αλγόριθμο εύκολα βγαίνει το συμπέρασμα ότι γίνονται πολλοί περιττοί υπολογισμοί. Έτσι, προκειμένου να βελτιωθεί η πολυπλοκότητα του αλγορίθμου αυτού, μπορεί να παραληφθεί η μία από τις τρεις φωλιασμένες εντολές `for`. Αυτό προκύπτει από την παρατήρηση ότι το άθροισμα των τιμών των στοιχείων του υποπίνακα `A(left..right)` είναι ίσο με το άθροισμα των τιμών των στοιχείων του υποπίνακα `A(left..right-1)` και του στοιχείου `A(right)`, οπότε κατά τον υπολογισμό του `A(left..right)` υπολογίζεται και πάλι το άθροισμα των τιμών των στοιχείων `A(left..right-1)`. Αν γίνει η αλλαγή αυτή στη συνάρτηση `max_subseq_sum1` προκύπτει η συνάρτηση `max_subseq_sum2`.

```
     function max_subseq_sum2;
1.   max_sum <-- 0; left <-- 0; right <-- 0;
2.   for i <-- 1 to n do
3.      current_sum <-- 0;
4.      for j <-- i to n do
5.         current_sum <-- current_sum+A[j];
6.         if (current_sum>max_sum) then
7.            max_sum <-- current_sum;
8.            left <-- i; right <-- j
9.   return max_sum
```

Ευκολότερα από την προηγούμενη φορά είναι δυνατόν να αποδειχθεί ότι ο αλγόριθμος έχει τετραγωνική πολυπλοκότητα $\Theta(n^2)$.

Τρίτος Αλγόριθμος (διαίρει και βασίλευε)
Μία διαφορετική αντιμετώπιση του προβλήματος προκύπτει ακολουθώντας τη στρατηγική Διαίρει και Βασίλευε, δηλαδή διαιρώντας τον πίνακα σε δύο υποπίνακες (σχεδόν) ίσου μήκους. Για τα υποδιανύσματα αυτά υπολογίζονται αναδρομικά οι ακολουθίες μέγιστου αθροίσματος, διαιρώντας τα σε επί μέρους, αν και όσο χρειασθεί. Έτσι διακρίνουμε τρεις περιπτώσεις: η υποακολουθία του μέγιστου αθροίσματος είτε περιέχεται στο πρώτο μισό του αρχικού πίνακα, είτε περιέχεται στο δεύτερο μισό, είτε τέλος περιέχεται και στα δύο. Όταν ισχύει η τρίτη περίπτωση, τότε η λύση του προβλήματος βασίζεται στη σκέψη ότι πρέπει να υπολογισθεί αφ' ενός η υποακολουθία με μέγιστο άθροισμα του πρώτου μισού, η οποία περιέχει το τελευταίο στοιχείο του πρώτου υποπίνακα και αφ'ετέρου η υποακολουθία με μέγιστο άθροισμα του δεύτερου μισού, η οποία περιέχει το πρώτο στοιχείο του δεύτερου υποπίνακα. Η λύση του αρχικού προβλήματος είναι εκείνη από τις τρεις υποακολουθίες που έχει το μέγιστο άθροισμα των τιμών των στοιχείων της.

Για παράδειγμα, αν ο πίνακας `A` έχει την πρώτη μορφή του Πίνακα 7.2, τότε διαιρείται στο μέσο και προκύπτει η δεύτερη μορφή του σχήματος. Με βάση αυτή τη θεώρηση υπολογίζεται η υποακολουθία μέγιστου αθροίσματος του πρώτου μισού (είναι ο υποπίνακας `A(1..2)=(3,3)` με άθροισμα 6), η υποακολουθία του δεύτερου μισού (είναι ο υποπίνακας `A(7)=(7)` με ένα μόνο στοιχείο) και η υποακολουθία που περιέχει το τελευταίο στοιχείο του πρώτου μισού και το πρώτο του δεύτερου μισού (είναι ο υποπίνακας `A(4..7)=(1,4,-4,7)` με άθροισμα 8). Επομένως, η υποακολουθία με μέγιστο άθροισμα είναι ο υποπίνακας `A(4...7)` με άθροισμα των τιμών των στοιχείων ίσο με 8.

3	3	-7	1	4	-4	7	-6	2	1

3	3	-7	1	4	-4	7	-6	2	1

Πίνακας 7.2: Μέγιστο άθροισμα υποακολουθίας (Διαίρει και Βασίλευε).

Η αναδρομική συνάρτηση `max_subseq_sum3` υπολογίζει το μέγιστο άθροισμα των τιμών των στοιχείων μεταξύ όλων των υποακολουθιών του πίνακα `A`, διαιρώντας τον πίνακα σε επιμέρους τμήματα. Αυτό είναι το σκέλος Διαίρει. Η βασική συνθήκη (base case) θεωρεί την υποακολουθία με ένα μόνο στοιχείο, όπου

αν το στοιχείο αυτό έχει τιμή θετική, τότε το άθροισμα ισούται με την τιμή αυτή, αλλιώς το άθροισμα ισούται με μηδέν. Με άλλα λόγια, στο σημείο αυτό υλοποιείται το σκέλος Βασίλευε της γενικής μεθόδου.

```
     function max_subseq_sum3(left,right);
1.   if left=right then
2.      if A[left]>0 then return A[left]
3.      else return 0;
4.   else
5.      center <-- (left+right)/2;
6.      max_lsum <-- max_subseq_sum3(left,center);
7.      max_rsum <-- max_subseq_sum3(center+1,right);
8.      max_lborder_sum <-- 0; lborder_sum <-- 0;
9.      for i <-- center downto left do
10.        lborder_sum <-- lborder_sum+A[i];
11.        if (lborder_sum>max_lborder_sum) then
12.           max_lborder_sum <-- lborder_sum
13.     max_rborder_sum <-- 0; rborder_sum <-- 0;
14.     for i <-- center+1 to right do
15.        rborder_sum <-- rborder_sum+A[i];
16.        if (rborder_sum>max_rborder_sum) then
17.           max_rborder_sum <-- rborder_sum
18.     temp <-- max_lborder_sum+max_rborder_sum;
19.     return max(max_lsum,max_rsum,temp)
```

Οι χρησιμοποιούμενες μεταβλητές είναι ακέραιες και έχουν το εξής νόημα. Οι μεταβλητές `max_lsum` και `max_rsum` εξυπηρετούν τις δύο πρώτες περιπτώσεις που αναφέρθηκαν και δίνουν το μέγιστο άθροισμα για το αριστερό και δεξιό υποπίνακα, αντίστοιχα. Οι υπόλοιπες μεταβλητές αφορούν στην τρίτη περίπτωση. Οι μεταβλητές `lborder_sum` και `rborder_sum` δίνουν το κάθε φορά υπολογιζόμενο άθροισμα στα αριστερά και δεξιά του `center`, ενώ στις `max_lborder_sum` και `max_rborder_sum` αποθηκεύεται το μέχρι στιγμής μέγιστο άθροισμα για το αριστερό και δεξιό υποπίνακα αντίστοιχα, τα οποία αργότερα θα προστεθούν για να δώσουν το αποτέλεσμα της τρίτης περίπτωσης. Τέλος, θεωρείται ότι υπάρχει μία συνάρτηση `max`, που επιστρέφει το μεγαλύτερο από τα στοιχεία που είναι τα ορίσματά της.

Η πολυπλοκότητα της μεθόδου μπορεί να προκύψει με βάση την αντίστοιχη αναδρομική εξίσωση. Κατ'αρχήν αν το μήκος του πίνακα είναι 1, τότε η δραστηριότητα στις εντολές 1-3 αποτιμάται ως σταθερό μοναδιαίο κόστος, δηλαδή $T(1) = 1$. Αν ο πίνακας είναι μεγαλύτερος, τότε θα εκτελεσθούν δύο αναδρομικές

κλήσεις, ενώ στις εντολές 9-17 θα εκτελεσθούν δύο απλοί βρόχοι for γραμμικής πολυπλοκότητας. Επομένως ισχύει:

$$T(n) \;=\; T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n$$

Θεωρώντας ότι $n = 2^k$ για απλοποίηση, έπεται ότι:

$$\begin{aligned} T(n) \;&=\; 2T(n/2) + n \;=\; 2\,(2T(n/4) + n/2) + n \;=\; 4T(n/4) + 2n \\ &=\; \ldots \;=\; 2^k T(n/2^k) + kn \;=\; 2^k + kn \;=\; n \lg n + n \;=\; \Theta(n \lg n) \end{aligned}$$

Στο συμπέρασμα αυτό καταλήγουμε αν απλώς θεωρήσουμε ότι το βάθος των αναδρομικών κλήσεων είναι $\Theta(\lg n)$, ενώ κάθε φορά ο πίνακας υφίσταται επεξεργασία της τάξης $\Theta(n)$. Ακολουθεί ο τελευταίος αλγόριθμος, που είναι ο καλύτερος που μπορούμε να σχεδιάσουμε.

Τέταρτος Αλγόριθμος (βέλτιστος τρόπος)
Τέλος, λοιπόν, μία ακόμη λύση του προβλήματος που διαπραγματευόμαστε, δίνεται από τη συνάρτηση max_subseq_sum4 που φαίνεται στη συνέχεια. Η συνάρτηση αυτή, σαρώνει όλα τα στοιχεία του πίνακα και τα προσθέτει όσο το άθροισμα τους είναι θετικό. Αν κάποια στιγμή η τιμή του αθροίσματος αυτού γίνει αρνητική, τότε μηδενίζεται η τιμή αυτού του αθροίσματος και συνεχίζεται η άθροιση των τιμών των στοιχείων από το επόμενο στοιχείο.

```
     function max_subseq_sum4;
1.   i <-- 0; current_sum <-- 0; max_sum <-- 0;
2.   left <-- 0; right <-- 0;
3.   for j <-- 1 to n do
4.      current_sum <-- current_sum+A[j];
5.      if (current_sum>max_sum) then
6.         max_sum <-- current_sum;
7.         left <-- i; right <-- j
8.      else
9.         if (current_sum<0) then
10.           i <-- j+1; current_sum <-- 0
11.  return max_sum;
```

Ο αλγόριθμος αυτός ανήκει στην κατηγορία των λεγόμενων on-line αλγορίθμων. Οι αλγόριθμοι αυτοί επεξεργάζονται τα δεδομένα όπως έρχονται, δηλαδή δεν είναι ανάγκη τα δεδομένα να είναι γνωστά εκ των προτέρων. Επιπλέον, αυτός ο τρόπος επίλυσης παρουσιάζει το σημαντικό πλεονέκτημα ότι σαρώνει μόνο μία

φορά τα στοιχεία του πίνακα. Αυτό σημαίνει ότι αν εξετασθεί κάποιο στοιχείο από τον αλγόριθμο, τότε δεν υπάρχει λόγος να παραμένει κάπου αποθηκευμένο γιατί δεν πρόκειται να χρησιμοποιηθεί ξανά. Επίσης ένα τελικό πλεονέκτημα του αλγορίθμου αυτού είναι ότι σε κάθε χρονική στιγμή, κατά τη διάρκεια της εκτέλεσης του αλγορίθμου, ο αλγόριθμος έχει υπολογίσει το μέγιστο άθροισμα της υποακολουθίας των στοιχείων που έχει εξετάσει. Η ανάλυση αυτού του αλγορίθμου είναι πολύ εύκολη υπόθεση. Καθώς ο κορμός του αλγορίθμου είναι ένας απλός βρόχος `for` (εντολή 3), έπεται ότι η πολυπλοκότητα είναι γραμμική $\Theta(n)$.

7.2 Τοποθέτηση 8 Βασιλισσών

Το πρόβλημα αυτό είναι ιστορικό και έχει απασχολήσει τους μεγάλους μαθηματικούς (όπως οι Gausss, Nauck, Guenther, Glaisher) πριν από τα μέσα του 19ου αιώνα. Το πρόβλημα ζητά να βρεθούν όλοι οι δυνατοί τρόποι τοποθέτησης 8 βασιλισσών σε μία σκακιέρα διαστάσεων 8x8 με τέτοιο τρόπο ώστε να μην αλληλο-απειλούνται. Βέβαια, είναι γνωστό στους σκακιστές ότι δύο βασίλισσες αλληλο-απειλούνται όταν βρίσκονται στην ίδια γραμμή, στην ίδια στήλη ή στην ίδια διαγώνιο. Συνολικά στο πρόβλημα υπάρχουν 92 λύσεις, αλλά πολλές από αυτές θεωρούνται ισοδύναμες καθώς η μία μπορεί να προκύψει από την άλλη με βάση τη συμμετρία ή εκτελώντας περιστροφές κατά 90^o. Έτσι, από τις 92 λύσεις προκύπτουν μόνο 12 διακριτές. Στο Σχήμα 7.1 παρουσιάζεται μία λύση του προβλήματος.

			Q				
					Q		
							Q
	Q						
						Q	
Q							
		Q					
				Q			

Σχήμα 7.1: Λύση του προβλήματος της τοποθέτησης των 8 βασιλισσών.

Ένας προφανής τρόπος επίλυσης του προβλήματος είναι να ελεγχθούν εξαντλητικά όλοι οι δυνατοί τρόποι τοποθέτησης των 8 βασιλισσών στις συνολικά 8x8=64 θέσεις της σκακιέρας. Όμως ο αριθμός των δυνατών αυτών τοποθετήσεων

είναι τεράστιος, καθώς ισούται με το συνδυασμό των 64 θέσεων ανά 8. Άρα, σύμφωνα με τον τρόπο αυτό πρέπει να ελεγχθούν 4.426.165.368 διαφορετικοί τρόποι τοποθέτησης. Μεταξύ αυτών των τρόπων συμπεριλαμβάνονται περιπτώσεις όπου σε μία γραμμή ή σε μία στήλη ή σε μία διαγώνιο μπορεί να τοποθετηθούν όχι μόνο δύο βασίλισσες, αλλά ακόμη περισσότερες, όπως τρεις, τέσσερις, κλπ μέχρι και οκτώ βασίλισσες. Προφανώς ο τρόπος αυτός δεν είναι αποδοτικός και γι'αυτό δεν εξετάζεται στη συνέχεια.

Πρώτος Αλγόριθμος (ωμή βία)

Μία πρώτη βελτίωση σε σχέση με την προηγούμενη προφανή αλλά μη πρακτική μέθοδο, είναι να θεωρηθεί ότι δεν μπορούν να τοποθετηθούν περισσότερες από μία βασίλισσες στη ίδια γραμμή. Η νέα μέθοδος οδηγεί σε σημαντική βελτίωση καθώς το πλήθος των δυνατών διαφορετικών καταστάσεων μειώνεται σε 8^8=16.777.216. Ο αριθμός αυτός προκύπτει λαμβάνοντας υπόψη ότι κάθε βασίλισσα μπορεί να καταλάβει 1 θέση μεταξύ 8 υποψηφίων θέσεων, ενώ η τοποθέτηση μίας βασίλισσας στην αντίστοιχη γραμμή μπορεί κατ' αρχήν να θεωρηθεί ανεξάρτητο γεγονός από τις τοποθετήσεις των άλλων βασιλισσών.

Οι βασίλισσες αριθμούνται από 1 ως 8, όπως επίσης οι στήλες και οι γραμμές της σκακιέρας. Έστω ότι η i-οστή βασίλισσα τοποθετείται στην i-οστή γραμμή και στη στήλη που συμβολίζεται με $X(i)$, (όπου $1 \leq i \leq 8$). Άρα λύσεις του προβλήματος αποτελούν τα διανύσματα $(X(1), \ldots, X(8))$, με τέτοιες τιμές των $X(i)$, έτσι ώστε να μην υπάρχουν δύο οποιεσδήποτε βασίλισσες που να βρίσκονται στην ίδια στήλη ή στην ίδια διαγώνιο. Συνεπώς, δεδομένης μίας τοποθέτησης για κάθε ζεύγος βασιλισσών πρέπει να γίνουν δύο έλεγχοι: (α) για το αν βρίσκονται στην ίδια στήλη, και (β) για το αν βρίσκονται στην ίδια διαγώνιο. Ο έλεγχος για να διαπιστώσουμε αν δύο βασίλισσες βρίσκονται στην ίδια στήλη είναι εύκολος. Δηλαδή, αν για την i-οστή και την j-οστή βασίλισσα ισχύει $X(i) = X(j)$ (όπου $1 \leq i, j \leq 8$), τότε είναι προφανές ότι οι βασίλισσες αυτές βρίσκονται στην ίδια στήλη. Όμως πώς διαπιστώνουμε αν δύο βασίλισσες βρίσκονται στην ίδια διαγώνιο;

Ο έλεγχος για το αν δύο βασίλισσες ανήκουν στην ίδια διαγώνιο βασίζεται στην εξής διαπίστωση: Αν θεωρήσουμε οποιαδήποτε θέση μίας διαγωνίου με κατεύθυνση από επάνω αριστερά προς κάτω δεξιά, τότε η διαφορά ὀγραμμή-στήληὄ παραμένει σταθερή. Για παράδειγμα, για τις θέσεις (1,1), (2,2), ..., (8,8) της κύριας διαγωνίου που ακολουθεί αυτήν την κατεύθυνση, η διαφορά αυτή ισούται με 0. Επίσης για τις θέσεις (1,2), (2,3), ..., (7,8) (αντίστοιχα (2,1), (3,2), ... (8,7)) της διαγωνίου που είναι επάνω (αντίστοιχα κάτω) από την προηγούμενη κύρια διαγώνιο, η διαφορά αυτή είναι -1 (αντίστοιχα 1).

Για τις θέσεις των διαγωνίων που έχουν κατεύθυνση από επάνω δεξιά προς κάτω αριστερά παρατηρούμε ότι παραμένει σταθερό το άθροισμα γραμμή+στήλη. Για παράδειγμα, για τα στοιχεία (1,8), (2,7), ... (8,1) της κύριας διαγωνίου στη συγκεκριμένη κατεύθυνση ισχύει γραμμή+στήλη=9. Για τα στοιχεία των διαγωνίων επάνω και κάτω από τη κεντρική αυτή διαγώνιο ισχύει γραμμή+στήλη=8 και 10, αντίστοιχα. Έτσι, με βάση τις παρατηρήσεις αυτές, καταλήγουμε ότι αν ισχύει $i - X(i) = j - X(j)$ ή $i + X(i) = j + X(j)$, τότε η i-οστή και η j-οστή βασίλισσα ανήκουν στην ίδια διαγώνιο.

Η λογική συνάρτηση `Place1(k)` που ακολουθεί, λαμβάνει ως παράμετρο την τιμή `k` της βασίλισσας που πρόκειται να τοποθετηθεί στη σκακιέρα, ελέγχει τις θέσεις των βασιλισσών στις προηγούμενες γραμμές, και επιστρέφει `true` αν μπορεί να γίνει η τοποθέτησή της στη στήλη `X(k)`.

```
     function Place1(k);
1.   j <-- 1; flag <-- true;
2.   while (j<k-1) and (flag=true) do
3.      if (X[i]<>X[k]) or (abs(X[i]-X[k])<>abs(i-k))
4.         then j <-- j+1
5.      else flag <-- false;
6.   return flag
```

Για να καθορισθεί αν ένα διάνυσμα $(X(1), \ldots, X(8))$, με τιμές των $X(i)$ (για $1 \leq i \leq 8$) στο διάστημα [1,8], αποτελεί λύση του προβλήματος, πρέπει να ελεγχθούν όλα τα ζεύγη των βασιλισσών, οπότε αν βρεθεί έστω και ένα ζεύγος όπου οι βασίλισσες αλληλο-απειλούνται, τότε το διάνυσμα δεν αποτελεί λύση. Ο έλεγχος αυτός γίνεται με την κλήση της λογικής συνάρτησης `Solution(X)`, που έχει παράμετρο το διάνυσμα `X` και επιστρέφει `true` αν το `X` είναι λύση του προβλήματος.

```
     function Solution(X);
1.   i <-- 1; flag <-- true;
2.   while (i<=8) and (place1(i)) do i <-- i+1;
3.   if i=9 then return true else return false
```

Η μέθοδος αυτή επίλυσης του προβλήματος μπορεί να υλοποιηθεί με τη χρήση 8 φωλιασμένων εντολών `for`, όπως φαίνεται στη συνέχεια. Όπως αναφέρθηκε, ο αλγόριθμος αυτός εξετάζει συνολικά 16.777.216 διαφορετικές τοποθετήσεις. Την πρώτη από αυτές, την βρίσκει και την τυπώνει μετά από 1.299.852 δοκιμές.

```
     procedure Queens1;
```

```
1.   for X[1] <-- 1 to 8 do
2.      for X[2] <-- 1 to 8 do
3.         for X[3] <-- 1 to 8 do
4.            for X[4] <-- 1 to 8 do
5.               for X[5] <-- 1 to 8 do
6.                  for X[6] <-- 1 to 8 do
7.                     for X[7] <-- 1 to 8 do
8.                        for X[8] <-- 1 to 8 do
9.                           if Solution(X) then
10.                             for j <-- 1 to 8 do
11.                                write(X[j],' ')
```

Δεύτερος Αλγόριθμος (ωμή βία)
Όπως έχει ήδη αναφερθεί, λύσεις του προβλήματος αποτελούν τα διανύσματα $(X(1),\ldots,X(8))$ με στοιχεία που είναι διαφορετικά μεταξύ τους και μπορούν να πάρουν τιμές από 1 έως 8, δηλαδή αποτελούν διαφορετικές διατάξεις των ακεραίων 1..8. Αυτό οδηγεί σε μία νέα μέθοδο που στηρίζεται στην εξέταση πολύ λιγότερων καταστάσεων σε σχέση με τις μεθόδους που εξετάσθηκαν προηγουμένως. Το πλήθος των δυνατών καταστάσεων είναι $8!= 40.320$.

Μία άλλη προσέγγιση του προβλήματος, λοιπόν, είναι με την κλήση κάποιας διαδικασίας `Perm` να παραχθούν όλες οι διαφορετικές διατάξεις των αριθμών στο διάνυσμα (1,2,3,4,5,6,7,8). Υπάρχουν πολλές μέθοδοι εύρεσης των διαφορετικών αυτών διατάξεων, οι οποίες λέγονται γεννήτριες διατάξεων (permutation generation). Η επόμενη διαδικασία `Perm` στηρίζεται στην εξής λογική: Θέτει όλες τις τιμές από 1 ως 8 στην πρώτη θέση του διανύσματος και για κάθε τέτοια τοποθέτηση, αναδρομικά βρίσκει τις διαφορετικές διατάξεις των 7 στοιχείων του διανύσματος, τα οποία απομένουν. Όταν αναδιαταχθούν τα 8 στοιχεία, τότε τυπώνεται το διάνυσμα που προκύπτει αν είναι λύση του προβλήματος.

```
     procedure Perm(i);
1.   if i=n then
2.      if Solution(X) then
3.         for j <-- 1 to n do write(X[j],' ')
4.   else
5.      for j <-- i to n do
6.         Swap(X[i],X[j]); Perm(i+1);
7.         Swap(X[j],X[i])
```

Η επόμενη διαδικασία `Queens2` αρχικά δίνει την τιμή (1,2,3,4,5,6,7,8) στο

διάνυσμα X και καλώντας τη διαδικασία Perm(1) βρίσκει τις διαφορετικές διατάξεις του διανύσματος, τυπώνοντας αυτές που αποτελούν λύση στο πρόβλημα.

```
     procedure Queens2;
1.   for i <-- 1 to 8 do X[i] <-- i;
2.   Perm(1)
```

Όπως αναφέρθηκε, ο αριθμός των δυνατών καταστάσεων είναι 40.320, αλλά η πρώτη λύση θα δοθεί μετά από 2.830 δοκιμές. Όμως πέρα από το γεγονός ότι οι λύσεις θα βρεθούν με πολύ λιγότερες δοκιμές σε σύγκριση με τη διαδικασία Queens1, η μέθοδος είναι ταχύτερη γιατί δεν χρειάζεται η συνάρτηση Place1 να εκτελεί ελέγχους σε σχέση με τις στήλες αλλά μόνο ως προς τις διαγώνιους. Η νέα εκδοχή συνάρτησης αυτής, Place2, παρουσιάζεται στη συνέχεια.

```
     function Place2(k);
1.   j <-- 1; flag <-- true;
2.   while (j<k-1) and (flag=true) do
3.      if (abs(X[i]-X[k])<>abs(i-k)) then j <-- j+1
4.      else flag <-- false;
5.   return flag
```

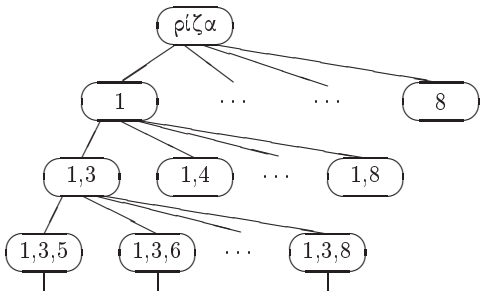
Τρίτος Αλγόριθμος (οπισθοδρόμηση)
Τέλος, το πρόβλημα των 8 βασιλισσών μπορεί να αντιμετωπιστεί με τη μέθοδο της Οπισθοδρόμησης, σύμφωνα με την οποία αφού τοποθετηθεί μία βασίλισσα, στη συνέχεια ελέγχεται αν υπάρχει επιτρεπτή θέση για να τοποθετηθεί η επόμενη. Αν βρεθεί τέτοια θέση, τότε ο αλγόριθμος διαχειρίζεται την τοποθέτηση της επόμενης βασίλισσας, διαφορετικά επιστρέφει (δηλαδή, οπισθοδρομεί) στην πιο πρόσφατα τοποθετημένη και την τοποθετεί στη επόμενη δυνατή θέση σε σχέση με αυτή όπου είχε τοποθετηθεί προηγουμένως. Αν τέτοια θέση δεν υπάρχει, τότε γίνεται οπισθοδρόμηση στην αμέσως προηγούμενη κοκ.

Η διαδικασία που υλοποιεί αυτόν τον τρόπο επίλυσης είναι η Queens3. Αυτή χρησιμοποιεί τη συνάρτηση Place1(k), που περιγράφηκε προηγουμένως, για να καθορίσει αν η k-οστή βασίλισσα μπορεί να τοποθετηθεί στη στήλη $X(k)$. Η κλήση της Queens3(8) δίνει τη λύση του προβλήματος.

```
     procedure Queens3 (n);
1.   X[1] <-- 0; k <-- 1;
2.   while k>0 do
3.      X[k] <-- X[k]+1;
```

```
4.      while (X[k]<=n) and (not place(k)) do
5.         X[k] <-- X[k]+1;
6.      if X[k]<=n then
7.         if k=n then
8.            for i <-- 1 to n do write(X[i],' ')
9.         else
10.           k <-- k+1; X[k] <-- 0
11.     else k <-- k-1 {* backtracking *}
```

Τέταρτος Αλγόριθμος (δικλάδωση με περιορισμό)
Ο τελευταίος τρόπος επίλυσης μπορεί να βελτιωθεί ακόμη περισσότερο, αν το πλήθος των καταστάσεων του προβλήματος οργανωθεί σε μορφή δένδρου. Σε κάθε επίπεδο του δένδρου από τη ρίζα μέχρι τα φύλλα, οι κόμβοι περιγράφουν μία νόμιμη κατάσταση με 0, 1, 2, κτλ. βασίλισσες τοποθετημένες ώστε να μην αλληλο-απειλούνται. Όλα τα κλαδιά (v, u) του δένδρου είναι τέτοια ώστε στον κόμβο v να περιγράφεται μία κατάσταση για k βασίλισσες, ενώ στο κόμβο u να περιγράφεται μία κατάσταση με $k+1$ βασίλισσες, η οποία προκύπτει από την προηγούμενη με την προσθήκη μίας επιπλέον βασίλισσας σε επιτρεπτή θέση. Τμήμα του δένδρου αυτού φαίνεται στο επόμενο σχήμα. Παρατηρούμε ότι η ρίζα απεικονίζει την κατάσταση ενός διανύσματος με 0 βασίλισσες και έχει οκτώ παιδιά. Ωστόσο, το πρώτο παιδί έχει μόνον έξι παιδιά καθώς δεν θα ήταν νόμιμη μία κατάσταση με μία βασίλισσα στις θέσεις 2,1 ή 2,2, δεδομένου ότι ήδη υπάρχει μία βασίλισσα στη θέση 1,1. Επεκτείνοντας αυτό το δένδρο προς τα φύλλα, είτε φθάνουμε σε καταστάσεις που οδηγούν σε αδιέξοδο και σταματούμε τις περαιτέρω τοποθετήσεις βασιλισσών, είτε οδηγούμαστε στη λύση.



Σχήμα 7.2: Λύση με διακλάδωση και περιορισμό.

Αυτή η μέθοδος έχει δύο πλεονεκτήματα σε σχέση με τις προηγούμενες. Το πρώτο είναι ότι μειώνει το πλήθος των καταστάσεων, όπου αναζητούνται

οι λύσεις του προβλήματος, αφού οι κόμβοι του δένδρου είναι λιγότεροι από 8!=40.320. Για την ακρίβεια, αν με τη βοήθεια του προγράμματος μετρήσουμε τους κόμβους του δένδρου, τότε θα βρούμε ότι είναι μόνο 2057, ενώ η πρώτη λύση θα βρεθεί μετά την εξέταση 114 κόμβων. Το δεύτερο είναι, ότι για να εξετασθεί αν ένας κόμβος του δένδρου, που εκφράζει ένα διάνυσμα μήκους k, αντιπροσωπεύει μία νόμιμη κατάσταση, δεν είναι απαραίτητο να ελεγχθούν όλα τα δυνατά ζεύγη μεταξύ των k βασιλισσών, αλλά αρκεί να ελεγχθεί αν η k-οστή βασίλισσα απειλεί τις προηγούμενες. Αυτός ο έλεγχος μπορεί να γίνει πολύ αποτελεσματικά αν σε κάθε κόμβο αποθηκεύουμε τις διαγωνίους (και των δύο κατευθύνσεων) που ελέγχονται από τις ήδη τοποθετημένες βασίλισσες. Τις διαγωνίους που έχουν κατεύθυνση από κάτω αριστερά προς επάνω δεξιά τις συμβολίζουμε με `d45`, ενώ με `d135` συμβολίζουμε τις διαγωνίους της άλλης κατεύθυνσης. Για να καταλάβουμε την αποτελεσματικότητα αυτής της τεχνικής, αρκεί να σκεφθούμε ότι στις προηγούμενες μεθόδους για να ελέγξουμε αν μία κατάσταση από 8 τοποθετημένες βασίλισσες είναι νόμιμη πρέπει να εκτελέσουμε 28 ελέγχους, στη χειρότερη περίπτωση.

Στη συνέχεια δίνεται η διαδικασία `Queens4` που κωδικοποιεί την τελευταία μέθοδο. Η λύση του προβλήματος λαμβάνεται δίνοντας την κλήση `Queens4 ((0,0,0,0,0,0,0,0),0,col,d45,d135)` . Προϋποτίθεται ότι οι μεταβλητές `col,d45,d135` είναι τύπου συνόλου (πχ. `SET of 1..15` στην Pascal).

```
     procedure Queens4(X; k; col,d45,d135);
1.   if k=8 then
2.      for j <-- 1 to 8 do write(X[j],' ')
2.   else
3.      for j <-- 1 to 8 do
5.         t1 <-- j-k; t2 <-- j+k;
6.         if not (j in col) and not (t1 in d45) and
7.               not (t2 in d135) then
8.            col <-- col+[j]; d135 <-- d135+[t2];
8.            d45 <-- d45+[t1]; X[k+1] <-- j;
9.            Queens4(X,k+1,col,d45,d135)
```

Στην προηγούμενη ανάπτυξη των λύσεων του προβλήματος των 8 βασιλισσών δεν έγινε αναφορά στην πολυπλοκότητα των αντίστοιχων μεθόδων. Αν γενικεύσουμε το πρόβλημα και θεωρήσουμε ότι πρέπει να τοποθετηθούν n βασίλισσες σε σκακιέρα $n \times n$ χωρίς να αλληλο-απειλούνται, τότε σε σχέση με την πολυπλοκότητα αναφέρονται συνοπτικά τα εξής. Η μη αναπτυχθείσα μέθοδος που αναφέρθηκε στην εισαγωγή έχει πολυπλοκότητα της τάξης $\binom{n^2}{n}$, άρα πολυπλοκότητα $\Theta(n^n)$.

Η πρώτη εξαντλητική μέθοδος έχει πολυπλοκότητα $\Theta(n^n)$, ενώ η δεύτερη εξαντλητική μέθοδος έχει πολυπλοκότητα $\Theta(n!) = \Theta(n^n)$. Με απλά λόγια, όλες έχουν την ίδια δραματική πολυπλοκότητα, με διαφορετικό βέβαια σταθερό συντελεστή. Οι δύο τελευταίες μέθοδοι διακρίνονται από εκθετική πολυπλοκότητα.

7.3 Περιοδεύων Πωλητής

Το πρόβλημα του περιοδεύοντος πωλητή (travelling salesperson problem, TSP) ζητά το συντομότερο κύκλο που πρέπει να ακολουθήσει ο πωλητής, ώστε αρχίζοντας από μία πόλη, να περάσει μία φορά από όλες τις πόλεις και να καταλήξει στην αφετηρία. Η έννοια του συντομότερου κύκλου μπορεί να θεωρηθεί ότι δηλώνει τον κύκλο του μικρότερου κόστους, όπου το κόστος μετράται είτε σε ώρες, είτε σε χιλιόμετρα, είτε σε χρήματα κτλ.). Θα ακολουθήσει μία λεπτομερής περιγραφή του τρόπου υλοποίησης των διαφόρων λύσεων που μπορούν να δοθούν στο πρόβλημα αυτό.

Θεωρώντας ότι κάθε πόλη είναι ένας κόμβος και ότι οι δρόμοι που ενώνουν δύο πόλεις είναι οι ακμές ενός ζυγισμένου συνδεδεμένου γράφου, το πρόβλημα μετασχηματίζεται στην εύρεση ενός κύκλου στο γράφο αυτό, όπου το άθροισμα των βαρών των ακμών να είναι ελάχιστο. Γενικά θεωρούμε ότι ο γράφος είναι πλήρης, δηλαδή όλες οι κορυφές ενώνονται μεταξύ τους. Ο γράφος μπορεί να αναπαρασταθεί από έναν πίνακα κόστους $C[1..n, 1..n]$, όπου n ο αριθμός των κόμβων. Η τιμή κάθε στοιχείου $C[i, j]$ του πίνακα είναι το κόστος της ακμής (i, j) του γράφου, όπου μάλιστα είναι δυνατόν να ισχύει $C[i, j] \neq C[j, i]$. Για τα διαγώνια στοιχεία του πίνακα ισχύει $C[i, i] = 0$, αλλά ακόμη και αν οι τιμές τους δεν είναι μηδενικές, δεν λαμβάνονται υπόψη, αφού το πρόβλημα απαιτεί επίσκεψη του κάθε κόμβου μόνο μία φορά. Αν δύο πόλεις δεν έχουν απ' ευθείας σύνδεση τότε ισχύει $C[i, j] = \infty$, οπότε δεν επηρεάζεται η λύση του προβλήματος. Στην περίπτωση αυτή, η ζητούμενη διαδρομή είναι ένας κύκλος Hamilton ελαχίστου κόστους.

Πρώτος Αλγόριθμος (άπληστη μέθοδος)
Ο άπληστος αλγόριθμος κάθε φορά επιλέγει από τον πίνακα κόστους C την ακμή εκείνη (i, j) που έχει την ελάχιστη τιμή, από όλες όσες δεν έχουν μέχρι στιγμής εξετασθεί και την προσθέτει στο ήδη σχηματισμένο μονοπάτι αν:

- δεν υπάρχει στο ήδη σχηματισμένο μονοπάτι άλλη ακμή που να ξεκινά από τον κόμβο i ή να καταλήγει στον κόμβο j, και
- με την προσθήκη της ακμής (i, j) δεν σχηματίζεται κύκλος (εκτός αν είναι

η ακμή εκείνη, που με την προσθήκη της σχηματίζεται ο κύκλος που περνά από όλους τους κόμβους του γράφου, δηλαδή ο κύκλος που είναι λύση του προβλήματος).

Με βάση τα προηγούμενα προκύπτει ότι αρχικά πρέπει να έχουμε τις ακμές οργανωμένες έτσι ώστε να βρίσκουμε εύκολα την ακμή με ελάχιστη τιμή κόστους. Αυτό σημαίνει ότι οι ακμές πρέπει να αποθηκευθούν σε ένα σωρό. Σε μία τέτοια περίπτωση, η εύρεση της κατάλληλης ακμής επιτυγχάνεται με τη βοήθεια της `DeleteHeap` (θα εξετασθεί στο Κεφάλαιο **??**), που επιστρέφει τα άκρα της ακμής (i, j) με το ελάχιστο κόστος, το οποίο ονομάζουμε min.

Οι ακμές που έχουν ήδη προστεθεί στο κύκλο, αποθηκεύονται σε μία λίστα που ορίζεται ως ολική μεταβλητή, την ονομάζουμε `Current_Path` και είναι τύπου δείκτη `Path_Edge` που δείχνει προς τη μεταβλητή `edge` τύπου εγγραφής. Σε κάθε κόμβο, δηλαδή, αποθηκεύεται μία ακμή (i, j), ενώ ο δείκτης `next` δείχνει στον επόμενο κόμβο της λίστας. Για να προστεθεί μία ακμή (i, j) στο μονοπάτι που έχει αναπτυχθεί μέχρι στιγμής, πρέπει να μην υπάρχει στο μονοπάτι αυτό (και επομένως στη λίστα `Current_Path`) άλλη ακμή της μορφής (i, k) ή (k, j). Αυτό συμβαίνει γιατί σε κάθε κύκλο υπάρχει για κάθε κόμβο μία ακμή που καταλήγει σ' αυτόν και μία που ξεκινά από αυτόν. Έτσι σαρώνονται όλα τα στοιχεία της λίστας και αν βρεθούν ακμές αυτής της μορφής, τότε η προσθήκη της νέας ακμής απορρίπτεται. Αυτή τη διαδικασία ακολουθεί η λογική συνάρτηση `Third_Edge(i,j)` που επιστρέφει `true` αν η ακμή (i, j) αποτελεί την τρίτη ακμή που προσπίπτει στον κόμβο i ή j, οπότε και απορρίπτεται η εισαγωγή της στο μονοπάτι.

```
     function Third_Edge(u,v);
1.   new(p); p <-- Current_Path;
2.   counter1 <-- 0; counter2 <-- 0;
3.   while (p<>nil) and (count1<=1) and (count2<=1) do
4.      if p.i=u then count1 <-- count1+1;
5.      if p.j=v then count2 <-- count2+1;
6.      p <-- p.next
7.   if (count1>=1) or (count2>=1) then return true
8.   else return false
```

Αν η συνάρτηση `Third_Edge(i,j)` επιστρέψει `false`, τότε πρέπει να γίνει έλεγχος για το σχηματισμό κύκλου, πριν προστεθεί η ακμή (i, j) στο μονοπάτι. Κύκλος σχηματίζεται όταν υπάρχουν ακμές $(j, k_1), (k_1, k_2), \ldots, (k_p, i)$ που να ανήκουν όλες στο μονοπάτι. Αν ο κύκλος αυτός περιέχει όλους τους κόμβους

του γράφου, τότε αποτελεί λύση του προβλήματος, αλλιώς απορρίπτεται η εισαγωγή της ακμής (i, j).

Η συνάρτηση `Find_Next_Node(v)` επιστρέφει έναν κόμβο k, αν η ακμή (v, k) ανήκει στο μονοπάτι. Αν δεν υπάρχει τέτοια ακμή, τότε επιστρέφει μηδέν, που σημαίνει ότι στην περίπτωση αυτή δεν σχηματίζεται κύκλος και επομένως η ακμή (i, j) μπορεί να εισαχθεί στο μονοπάτι.

```
     function Find_Next_Node(u);
1.   new(p); p <-- Current_Path;
2.   while (p.next<>nil) and (p.i<>u) do p <-- p.next;
3.   if p.i=u then return p.j else return 0
```

Η λογική συνάρτηση `Cycle(i,j)` χρησιμοποιεί τη συνάρτηση `Find_Next_Node` και από τον κόμβο j αναπτύσσει το μεγαλύτερο μονοπάτι που μπορεί να αναπτυχθεί, έτσι ώστε όλες οι ακμές του να περιέχονται στο `Current_Path`. Έστω ότι το μονοπάτι αυτό είναι το $(j, k_1, k_2, \ldots, k_p)$. Αν $k_p = i$, τότε σχηματίζεται κύκλος, οπότε αν οι κόμβοι $j, k_1, k_2, \ldots, k_p$ είναι ακριβώς οι κόμβοι του γράφου, τότε το μονοπάτι αυτό είναι η λύση του προβλήματος. Αν όμως οι κόμβοι $j, k_1, k_2, \ldots, k_p$ είναι λιγότεροι από τους κόμβους του γράφου, τότε αυτό σημαίνει ότι σχηματίζεται κύκλος που δεν αποτελεί λύση. Έτσι η συνάρτηση `Cycle` επιστρέφει `true` απορρίπτοντας την εισαγωγή της ακμής (i, j) στο `Current_Path`. Αν δεν ισχύει $k_p = i$ και δεν υπάρχει κόμβος k_{p+1} τέτοιος ώστε η ακμή (k_p, k_{p+1}) να περιέχεται στο `Current_Path`, δηλαδή η κλήση της συνάρτησης `Find_Next_Node(kp)` επιστρέφει μηδέν, τότε δεν σχηματίζεται κύκλος με την εισαγωγή της (i, j) και η `Cycle(i,j)` επιστρέφει `false`.

```
     function Cycle(u,v);
1.   s <-- [u,v]; a <-- Find_Next_Node(v);
2.   while (a<>0) and (a<>u) do
4.      s <-- s+[a]; a <-- Find_Next_Node(a)
6.   if (a=0) then return false
7.   else if (a=u) and (Include_all(s)) then
9.      cycle <-- false; solution <-- true
11.  else return true;
```

Η συνάρτηση `Cycle` καλεί τη λογική συνάρτηση `Include_all`, που σκοπό έχει να ελέγξει αν στο μονοπάτι περιλαμβάνονται όλες οι κορυφές του γράφου και δίνεται στη συνέχεια. Επίσης, η μεταβλητή `s` είναι τύπου `set`, ενώ η λογική μεταβλητή `solution` είναι καθολική και θα φανεί η χρήση της μέσα στο κύριο πρόγραμμα.

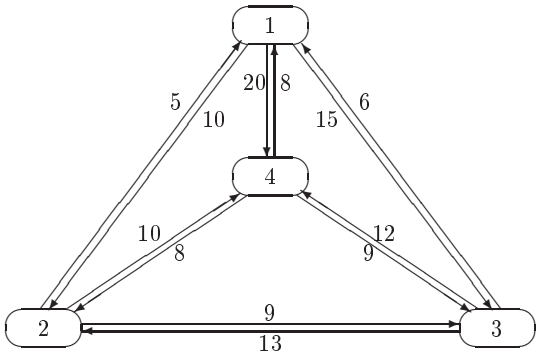
```
     function Include_all(s);
1.   Include_all <-- true;
2.   while (Include_all=true) and i<=n do
3.      if (not (i in s)) then Include_all <-- false;
4.      i <-- i+1
```

Επομένως η συνολική διαδικασία που ακολουθείται είναι: όσο δεν έχει βρεθεί λύση, επιλέγεται η ακμή με το μικρότερο κόστος μεταξύ αυτών που δεν έχουν εξετασθεί ως τώρα, γίνονται οι δύο έλεγχοι και ανάλογα, είτε γίνεται η εισαγωγή της ακμής στο Current_Path είτε απορρίπτεται. Στη συνέχεια ακολουθεί το κύριο πρόγραμμα που δένει όλες τις προηγούμενες συναρτήσεις. Getfile είναι η συνάρτηση που διαβάζει το γράφο εισόδου, ο οποίος αναπαρίσταται με τη μέθοδο του πίνακα γειτνίασης.

```
1.   Getfile; new(Current_Path); Current_Path <-- nil;
2.   lenght <-- 0; solution <-- false;
3.   while (not solution) do
4.      DeleteHeap(i,j,min);
5.      if not(Third_edge(i,j) and Cycle(i,j)) then
6.         new(p); p.i <-- i; p.j <-- j;
7.         p.next <-- Current_Path;
8.         Current_Path <-- p; length <-- length+min;
9.         if solution=true then
10.           new(p); p <-- Current_Path;
11.           while p<>nil do
12.              writeln(p.i,'  ',p.j); p <-- p.next
13.           writeln('Συνολικό κόστος ',length)
```

Ας θεωρήσουμε το γράφο του Σχήματος 7.3. Για την επίλυση του TSP για το γράφο αυτό, ο άπληστος αλγόριθμος ακολουθεί την εξής διαδικασία: Η ακμή με το ελάχιστο κόστος είναι η (2,1) με κόστος 5, που είναι η πρώτη ακμή που εισάγεται στο Current_Path. Από αυτές που απομένουν, η ακμή με ελάχιστο κόστος είναι η (3,1), που όμως δεν εισάγεται στο μονοπάτι, γιατί ήδη σε αυτό υπάρχει η (2,1). Για τον ίδιο λόγο απορρίπτεται και η (4,1). Στον Πίνακα 7.3 παρουσιάζονται διάφορα στοιχεία καθώς ο αλγόριθμος εξελίσσεται.

Ο κύκλος που σχηματίζεται αποτελείται από τις ακμές (2,1), (4,2), (3,4) και (1,3), είναι ο (1,3,4,2,1) και έχει μήκος 40. Παρατηρείται ότι ο κύκλος (1,2,4,3,1) έχει μήκος 35. Επομένως ο αλγόριθμος αυτός δεν δίνει πάντα τη βέλτιστη λύση. Για το λόγο κατατάσσεται στην κατηγορία των προσεγγιστικών ευριστικών



Σχήμα 7.3: Παράδειγμα γράφου.

άπληστων αλγορίθμων. Ωστόσο ο αλγόριθμος αυτός είναι χρήσιμος γιατί είναι σχετικά γρήγορος σε σχέση με αυτούς που θα εξετασθούν στη συνέχεια. Πιο συγκεκριμένα, όπως θα μελετήσουμε στο Κεφάλαιο **??**, η συνάρτηση `DeleteHeap` είναι τάξης $O(\lg n)$, οι συναρτήσεις `Third_Edge, Find_Next_Node` και `Include_All` είναι γραμμικές, ενώ η συνάρτηση `Cycle` είναι τετραγωνικής τάξης. Το τελευταίο προκύπτει από την πράξη βαρόμετρο `while p<>nil do`. Τελικά η πολυπλοκότητα της μεθόδου αυτής είναι τάξης $O(n^2)$. Η επόμενη λύση δίνει πράγματι τον κύκλο με το μικρότερο κόστος.

Ακμή	Κόστος	Ενέργεια
(2,1)	5	Εισάγεται
(3,1)	6	Απορρίπτεται λόγω του πρώτου ελέγχου
(4,1)	8	Απορρίπτεται λόγω του πρώτου ελέγχου
(4,2)	8	Εισάγεται
(2,3)	9	Απορρίπτεται λόγω του πρώτου ελέγχου
(4,3)	9	Απορρίπτεται λόγω του πρώτου ελέγχου
(1,2)	10	Απορρίπτεται λόγω του δεύτερου ελέγχου
(2,4)	10	Απορρίπτεται λόγω του δεύτερου ελέγχου
(3,4)	12	Εισάγεται
(3,2)	13	Απορρίπτεται λόγω του πρώτου ελέγχου
(1,3)	15	Εισάγεται, τέλος.
(1,4)	20	

Πίνακας 7.3: Εξεταζόμενες ακμές, κόστος, και ενέργεια

Δεύτερος Αλγόριθμος (δυναμικός προγραμματισμός)
Σύμφωνα με με τις αρχές του Δυναμικού Προγραμματισμού, τις οποίες γνωρίζουμε από τη Σχεδίασης Αλγορίθμων, το TSP μπορεί να λυθεί κάνοντας χρήση της συνάρτησης:

$$g(i,S) = \min_{j \in S}(C_{ij} + g(j, S - \{j\}))$$

όπου το S είναι σύνολο κόμβων. Η συνάρτηση $g(i,S)$ επιστρέφει το κόστος του συντομότερου μονοπατιού που ξεκινά από τον κόμβο i, περνά ακριβώς μία φορά από όλους τους κόμβους του συνόλου S και καταλήγει στον κόμβο 1, που χωρίς απώλεια της γενικότητας μπορεί να θεωρηθεί ότι ο κύκλος από αυτόν ξεκινά και σε αυτόν καταλήγει. Η συνάρτηση αυτή μπορεί να υλοποιηθεί αναδρομικά ως εξής.

```
     function g(i,s);
1.   cost <-- maxint;
2.   if s=[] then g <-- C[i,1]
3.   else
4.      for j <-- 2 to n do
5.         if j in s then
6.            k <-- C[i,j]+g(j,s-[j]);
7.            if k<cost then cost <-- k;
8.            g <-- cost
```

Αν δεν ζητείται μόνο το κόστος της συντομότερης διαδρομής, αλλά και αυτή η ίδια η διαδρομή, τότε πρέπει να ορισθεί μία συνάρτηση J, τέτοια ώστε το $J(i,S)$ να ισούται με τον κόμβο j που ελαχιστοποιεί το άθροισμα $C_{ij} + g(j, S - \{j\})$.

Ορίζεται μία λίστα `Next_Node` του τύπου `List_Next_Node`, που είναι ένας δείκτης προς μεταβλητές τύπου `element`, που με τη σειρά του είναι τύπου εγγραφής με τα πεδία `node, s, node_min_cost, next`. Σε κάθε στοιχείο της λίστας αυτής αποθηκεύονται ο κόμβος j (που ελαχιστοποιεί το κόστος $g(i,S)$) ως τιμή του πεδίου `node_min_cost`, ο κόμβος i ως τιμή του πεδίου `node` καθώς επίσης και το σύνολο `S`.

Για να μην υπολογίζεται μόνο το κόστος του μονοπατιού, αλλά να προσδιορίζεται και το ίδιο το μονοπάτι που έχει ελάχιστο κόστος, χρησιμοποιείται μία παραλλαγή της συνάρτησης `g` που περιγράφτηκε προηγουμένως. Σύμφωνα με τη νέα παραλλαγή κάθε φορά που βρίσκεται ένας κόμβος j που ελαχιστοποιεί το κόστος $C_{ij} + g(j, S-\{j\})$, προστίθεται στη λίστα `Next_Node` ένας κόμβος με πεδία που έχουν αντίστοιχα τις τιμές i, S, j. Αν όμως υπάρχει ήδη στη λίστα κόμβος με τιμές στα δύο πρώτα πεδία τις τιμές i και S, τότε ενημερώνεται η τιμή του τρίτου πεδίου και γίνεται j.

```
     function g(i,s);
1.   cost <-- maxint;
2.   if s=[] then g <-- C[i,1]
3.   else
4.      for j <-- 2 to n do if j in s then
5.         k <-- C[i,j]+g(j,s-[j]);
6.         if k<cost then
7.            cost <-- k;
8.            if not exist_in_path(i,s,j) then
9.               new(p); p.node <-- i; p.s <-- s;
10.              next_node <-- p; p.node_min_cost <-- j;
11.              p.next <-- next_node
12.        g <-- cost
```

Μετά την κλήση της συνάρτησης αυτής έχει υπολογισθεί το κόστος του συντομότερου κύκλου και έχουν βρεθεί οι τιμές των κόμβων j που ελαχιστοποιούν τα αθροίσματα $C_{ij} + g(j, S - \{j\})$ για κάθε κόμβο i του γράφου και για κάθε σύνολο κόμβων S. Ο συντομότερος κύκλος είναι $(1, J(1, \{2..N\}), J(J(1, \{2..N\}), \{2..N\} - J(1, \{2..N\})), \ldots, 1)$. Για να βρεθεί γίνεται αναζήτηση στους κόμβους της λίστας `Next_Node`, όπως φαίνεται από τον κώδικα της διαδικασίας `Search(i,S)` που ακολουθεί.

```
     procedure Search(i,s);
1.   new(p); p <-- next_node;
2.   while ((p.next<>nil) and ((p.node<>i) or (p.s<>s)))
3.      do p <-- p.next;
4.   if (p.node=i) and (p.s=s) then
5.      write(p.node_min_cost,' ');
6.      Search(p.node_min_cost,s-[p.node_min_cost]);
```

Για το γράφο του προηγούμενου σχήματος, οι τιμές της συνάρτησης J παρουσιάζονται στον Πίνακα 7.3, οπότε ο συντομότερος κύκλος που βρίσκει ο αλγόριθμος είναι (1,2,4,3,1) με κόστος 35.

J(2,[3])=3	J(3,[2])=2	J(4,[2])=2	J(2,[3,4])=4	J(4,[2,3])=2
J(2,[4])=4	J(3,[4])=4	J(4,[3])=2	J(3,[2,4])=4	J(1,[2,3,4])=2

Πίνακας 7.4: Τιμές συνάρτησης J για τον υπολογισμό του κύκλου

Επιλογικά, στη βιβλιογραφία αναφέρονται και άλλες επακριβείς λύσεις που στηρίζονται στη μέθοδο της οπισθοδρόμησης και στη μέθοδο της διακλάδωσης και του περιορισμού. Το γεγονός είναι ότι το TSP είναι ένα κλασικό NP-complete πρόβλημα και επομένως δεν μπορεί κανείς να είναι αισιόδοξος ότι θα βρει αποδοτικούς αλγορίθμους για μία γενική λύση του. Οι τεχνικές που περιγράψαμε εδώ δίνουν μία αρκετά ικανοποιητική προσεγγιστική λύση για μικρές (σχετικά) τιμές. Επειδή το πρόβλημα έχει πολλές πρακτικές εφαρμογές, έχει απασχολήσει πολλούς ερευνητές σε μία προσπάθεια να πετύχουν λύσεις σε όλο και μεγαλύτερα στιγμιότυπα του προβλήματος. Το 1954 οι Dantzig, Fulkerson και Johnson έλυσαν το πρόβλημα για 42 πόλεις των ΗΠΑ, ενώ το 1980 οι Padberg και Hong το έλυσαν για 318 πόλεις. Το 1992 οι Applegate, Bixby, Cook από το Πανεπιστήμιο Rice, και ο Chvatal από το Πανεπιστήμιο Rutgers βρήκαν τη βέλτιστη διαδρομή για 3.038 πόλεις των ΗΠΑ, χρησιμοποιώντας ένα σύστημα από 50 σταθμούς εργασίας. Το 1993 ανέβασαν τον αριθμό των πόλεων σε 4.461 πόλεις, ενώ το 1994 ανέβασαν τον αριθμό αυτό σε 7.397 πόλεις. Αυτό το ρεκόρ είχε μείνει ακατάρριπτο μέχρι το 1998. Με τη βοήθεια 3 ισχυρών μηχανών (Digital AlphaServer 4100 με 12 επεξεργαστές) και ένα σύνολο από 12 προσωπικούς υπολογιστές Pentium II, οι οποίοι έτρεχαν το πρόγραμμα για περίπου 3 μήνες, οι τέσσερις ερευνητές πέτυχαν να δώσουν λύση στο πρόβλημα για τις 13.509 πόλεις των ΗΠΑ με πληθυσμό περισσότερο από 500 κατοίκους.

7.4 Βιβλιογραφική Συζήτηση

Στο παρόν κεφάλαιο παρουσιάσθηκαν οι γνωστές τεχνικές σχεδίασης αλγορίθμων μέσα από μία engineering προσέγγιση για την επίλυση μερικών προβλημάτων, με σκοπό τη σταδιακή παραγωγή βελτιωμένης λύσης. Με παρόμοια προσέγγιση θα μπορούσε να επιλυθούν πλήθος άλλων προβλημάτων. Αντί άλλης βιβλιογραφικής παραπομπής συνιστάται την ανάγνωση του άρθρου [155], που συνεγράφη από τον Tarjan με την ευκαιρία της βράβευσής του με το Turing award, όπου συνοπτικά παρουσιάζεται όλο το πανόραμα του αντικειμένου.

Ειδικότερα για το πρόβλημα του μέγιστου αθροίσματος υποακολουθίας εξετάσθηκε εξαντλητικά στο βιβλίο του Bentley [7], όπου μπορούν να βρεθούν αποτελέσματα από τις δοκιμές των υλοποιήσεων των αλγορίθμων. Τονίζεται ότι στα βιβλία του Bentley [6, 7, 8] παρουσιάζονται ενδιαφέρουσες τεχνικές σχεδιασμού αλλά και κωδικοποίησης αλγορίθμων, σε μια βήμα προς βήμα βελτίωση του τελικού αποτελέσματος. Ακόμη, σχετικό υλικό υπάρχει στην προσωπική σελίδα του Bentley (`www.programmingpearls.com/teaching.html`). Επίσης το πρόβλημα μελετάται σε βάθος και στα βιβλία των Weiss [169] και Cohen [24], αλλά με

περισσότερο θεωρητικό και μεθοδολογικό τρόπο. Επίσης, το άρθρο [105] αναφέρεται στο ίδιο πρόβλημα.

Το πρόβλημα των 8 βασιλισσών είναι ένα κλασικό πρόβλημα για τη γνωστική περιοχή της Τεχνητής Νοημοσύνης (Artificial Intelligence) με μακρά ιστορία, καθώς ασχολήθηκε με αυτό ο Gauss το 1859. Στα πλαίσια του παρόντος βιβλίου εξετάζεται σε σχέση με τις γνωστές αλγοριθμικές τεχνικές. Το αντικείμενο εξετάζεται στο βιβλίο [173] καθώς και στα άρθρα [60, 152].

Το βιβλίο των Lawler-Lenstra-Rinnooy Kan-Shmoys [82] αναφέρεται αποκλειστικά στο πρόβλημα του περιοδεύοντος πωλητή. Επίσης, στο βιβλίο των Moret-Shapiro [114] υπάρχει εκτενής αναφορά στο ίδιο πρόβλημα (σελίδες 256-261). Ειδικότερα αναφέρεται η υλοποίηση μίας εξάδας άπληστων τεχνικών, ενώ τα αποτελέσματα τους συγκρίνονται με την επίδοση της βέλτιστης λύσης για ένα σύνολο 9 και 57 πόλεων των ΗΠΑ. Στο διαδίκτυο υπάρχουν πολλές σελίδες όπου υπάρχουν προγράμματα εμψύχωσης και οπτικοποίησης (όπως η σελίδα `http://www.cs.oswego.edu/~birgit/` όπου υπάρχει ένα κατάλογος με συνδέσμους σε ανάλογες σελίδες). Στη σελίδα `http://itp.nat.uni-magdeburg.de/~mertens/TSP/TSP.html` υπάρχει διαθέσιμος κώδικας επίλυσης του προβλήματος, όπου παρουσιάζονται αρκετές ευριστικές λύσεις.

Η Ασκήσεις 2-3 (το πρόβλημα της Ολλανδικής σημαίας) αναφέρονται στο βιβλίο [61], η Άσκηση 9 στο [17], η Άσκηση 10 για το πρόβλημα των 9 κερμάτων στα άρθρα [29, 65], η Άσκηση 13 στο άρθρο [51], ενώ η Άσκηση 14 στο βιβλίο [169]. Οι Ασκήσεις 6-7, που επεκτείνουν το πρόβλημα της εύρεσης του μέγιστου αθροίσματος υποακολουθίας, έχουν τεθεί αντιστοίχως στον Πανελλήνιο Διαγωνισμό Πληροφορικής του 1999 και στη Διεθνή Ολυμπιάδα Πληροφορικής του 1994, ενώ η Άσκηση 12 έχει τεθεί στη Βαλκανική Ολυμπιάδα Πληροφορικής του 1994.

7.5 Ασκήσεις

1. Να σχεδιασθούν και να αναλυθούν εναλλακτικοί αλγόριθμοι για την εύρεση των πενταψηφίων αριθμών, των οποίων το τετράγωνο αποτελείται από δέκα ανόμοια ψηφία. Οι αριθμοί αυτοί ονομάζονται καρκινικοί.

2. Δίνονται δύο ταξινομημένοι πίνακες `A[0..n-1]` και `B[0..m-1]`. Κάθε πίνακας αποτελείται από διακριτά στοιχεία. Να σχεδιασθούν και να αναλυθούν εναλλακτικοί αλγόριθμοι για την εύρεση του πλήθους των κοινών στοιχείων μεταξύ των δύο πινάκων.

3. Δίνεται ένας πίνακες `A[0..n-1]`, όπου κάθε στοιχείο είναι έχει τιμή ένα από τα χρώματα: άσπρο, μπλε και κόκκινο. Να σχεδιασθούν και να

αναλυθούν εναλλακτικοί αλγόριθμοι ώστε με τις κατάλληλες αντιμεταθέσεις, να έρθουν στην αρχή τα κόκκινα στοιχεία, και στο τέλος τα μπλε.

4. Δίνεται πίνακας A με n στοιχεία και ζητείται το δεύτερο μικρότερο ($k = 2$). Να σχεδιασθούν και να αναλυθούν δύο αλγόριθμοι για το πρόβλημα. Ο ένας να εκτελεί μια κατάλληλη σάρωση του πίνακα (Ωμή Βία), ενώ ο άλλος να στηρίζεται στη αρχή του Διαίρει και Βασίλευε.

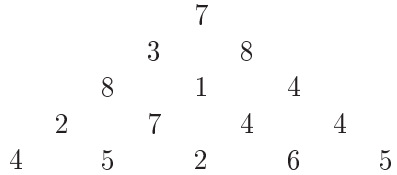
5. Να σχεδιασθούν και να αναλυθούν εναλλακτικοί αλγόριθμοι για τον υπολογισμό:

   - του ελάχιστου αθροίσματος υποακολουθίας,
   - του ελάχιστου θετικού αθροίσματος υποακολουθίας, και
   - του μέγιστου γινομένου υποακολουθίας.

6. Δίνεται ένας τετραγωνικός πίνακας `A[0..n-1,0..n-1]` με θετικούς και αρνητικούς ακέραιους. Να βρεθεί ο υποπίνακας (δηλαδή, με διάσταση από 1x1 μέχρι `nxn`) με το μεγαλύτερο άθροισμα στοιχείων. Να σχεδιασθούν και να αναλυθούν εναλλακτικές λύσεις.

7. Το Σχήμα 7.4 παρουσιάζει ένα αριθμητικό τρίγωνο, όπου η i-οστή γραμμή περιέχει i ακεραίους. Ζητείται να υπολογισθεί το μεγαλύτερο άθροισμα αριθμών επάνω σε μία διαδρομή, που ξεκινά από την κορυφή και τερματίζει κάπου στη βάση. Μία διαδρομή προσδιορίζεται σε κάθε βήμα από τη μετακίνηση διαγωνίως είτε κάτω αριστερά είτε κάτω δεξιά. Να σχεδιασθούν δύο αλγόριθμοι επίλυσης του προβλήματος που να βασίζονται στην Ωμή Βία και στο Δυναμικό Προγραμματισμό.

8. Δίνεται ένας πίνακας `A[0..n-1]` που περιέχει τους αριθμούς από 1 μέχρι `n`. Να βρεθεί ο αριθμός των αντιστροφών (inversions), δηλαδή των περιπτώσεων όπου `A[i]<A[j]` αλλά `i>j`. Να σχεδιασθούν και να αναλυθούν εναλλακτικές λύσεις που να στηρίζονται στη μέθοδο της Ωμής Βίας και τη μέθοδο Διαίρει και Βασίλευε.



Σχήμα 7.4: Το πρόβλημα του τριγώνου.

9. Μία αυτόματη μηχανή πωλήσεων δέχεται νομίσματα των 10, 20 και 50 λεπτών. Ποιός είναι το πλήθος των διαφορετικών τρόπων που μπορούμε να τροφοδοτήσουμε τη μηχανή με n λεπτά, όπου το n είναι πολλαπλάσιο του 10; Να σχεδιασθούν και να αναλυθούν δύο λύσεις, με Αναδρομή και με Δυναμικό Προγραμματισμό.

10. Ένας πίνακας 3x3 περιλαμβάνει 9 κέρματα. Αρχικά στην επάνω όψη άλλα κέρματα δείχνουν γράμματα και άλλα κορώνα. Ζητείται μία δεδομένη αρχική κατάσταση των κερμάτων να μετατραπεί σε μία τελική, όπου όλα τα κέρματα έχουν τα γράμματα στην επάνω όψη, κάνοντας τον ελάχιστο αριθμό αλλαγών όψεων. Με τον όρο "αλλαγή όψεων" (flip) εννοούμε ότι όταν γυρίζουμε ένα κέρμα, ταυτόχρονα γυρίζουμε και τα γειτονικά του που βρίσκονται προς τα επάνω, κάτω, αριστερά και δεξιά (όσα από αυτά υπάρχουν). Στον Πίνακα 7.5 παρουσιάζονται αριθμημένες οι θέσεις του πίνακα. Για παράδειγμα, γειτονικές της θέσης 5 είναι οι θέσεις 2, 4, 6 και 8, ενώ της θέσης 3 είναι οι 2 και 6, και της θέσης 2 είναι οι 1, 3 και 5. Να δοθούν λύσεις που να βασίζονται σε Δυναμικό Προγραμματισμό και Οπισθοδρόμηση.

1	2	3
4	5	6
7	8	9

Πίνακας 7.5: Το πρόβλημα με τα 9 κέρματα.

11. Για το πρόβλημα του υπολογισμού για ρέστα με τον ελάχιστο αριθμό κερμάτων αναφέρονται στη βιβλιογραφία δύο λύσεις: με Άπληστη Μέθοδο και με Δυναμικό Προγραμματισμό. Ποιά είναι καλύτερη σύμφωνα με αναλυτικά κριτήρια; Δίνουν και οι δύο μέθοδοι πάντοτε τη βέλτιστη λύση; Για το σχεδιασμό και την ανάλυση των δύο αλγορίθμων να θεωρηθούν οι εξής δύο σειρές νομισμάτων: (α) 1, 2, 5, 10, 20, 50 λεπτά, και (β) 1, 5, 25, 50 λεπτά.

12. Έστω ένα γραμμικό δίκτυο υπολογιστών, όπου ο κάθε υπολογιστής συνδέεται ακριβώς με δύο άλλους, εκτός από τους δύο ακραίους υπολογιστές που συνδέονται μόνο με έναν. Η απόσταση μεταξύ δύο υπολογιστών i και j δίνεται από το στοιχείο `A[i,j]` του πίνακα `A[0..n-1,0..n-1]`, όπου $0 \leq i, j \leq n-1$. Το πρόβλημα έγκειται στην εύρεση του τρόπου σύνδεσης των υπολογιστών, ώστε σε μία τέτοια ανοικτή αλυσίδα να ελαχιστοποιηθεί

το μήκος του απαιτούμενου καλωδίου. Υπόψη ότι το καλώδιο θα πρέπει να τοποθετηθεί κάτω από το πάτωμα, οπότε το συνολικό μήκος του καλωδίου που χρειάζεται για τη σύνδεση δύο διαδοχικών υπολογιστών ισούται με την απόσταση μεταξύ των υπολογιστών συν 10 μέτρα επιπλέον. Να σχεδιασθεί και να αναλυθεί αλγόριθμος για την επίλυση του προβλήματος.

13. Δίνεται το έτος γέννησης και το έτος θανάτου n κροκοδείλων. Σκοπός είναι ο υπολογισμός του μέγιστου πλήθους ζώντων κροκοδείλων σε οποιαδήποτε χρονική στιγμή. Να σχεδιασθούν και να αναλυθούν αλγόριθμοι για την επίλυση του προβλήματος. Να θεωρηθεί ότι: (α) όλες οι χρονολογίες είναι διακριτές και ανήκουν στο διάστημα `[-100000..2000]`, και (β) τα δεδομένα εισόδου αποθηκεύονται σε ένα πίνακα `A[1..2,0..n-1]`.

14. Έστω ότι πρέπει να παράξουμε όλες τις δυνατές διατάξεις n αριθμών με τη βοήθεια ενός πίνακα `A[0..n-1]`. Για το σκοπό αυτό θα χρησιμοποιηθεί η γεννήτρια τυχαίων αριθμών `rant_int(i,j)`, η οποία ισοπίθανα γεννά τους ακέραιους από i μέχρι j. Σχεδιάζουμε τους εξής 3 αλγορίθμους:

    - Γεμίζουμε από την αρχή μία-μία τις θέσεις του πίνακα `A`. Προκειμένου να γεμίσουμε τη θέση i, (όπου, $0 \leq i \leq n-1$) καλούμε τη γεννήτρια `rant_int` που δίνει έναν τυχαίο ακέραιο. Ελέγχουμε σειριακά τον πίνακα μέχρι τη συγκεκριμένη θέση μήπως ο ακέραιος αυτός έχει ήδη εισαχθεί. Αν δεν έχει εισαχθεί, τότε τον αποθηκεύουμε στη θέση i, αλλιώς καλούμε εκ νέου τη συνάρτηση.
    - Όπως προηγουμένως, αλλά διατηρούμε έναν επιπλέον πίνακα που ονομάζουμε `used`. Για κάθε εισαγόμενο τυχαίο αριθμό `ran`, θέτουμε `used[ran]=1`. Επομένως, ο έλεγχος μήπως έχει ήδη εισαχθεί κάποιος ακέραιος γίνεται άμεσα και όχι σειριακά.
    - Αρχικοποιούμε τον πίνακα ως εξής: `A[i]=i+1`. Κατόπιν εκτελούμε τις εντολές:

      ```
      for (i=1; i<n; i++)
         swap(&A[i], &A[rand_int(0,i)]);
      ```

    Να αναλυθούν οι αλγόριθμοι και να δοθούν οι σχετικοί συμβολισμοί Ο.

8

Αλγόριθμοι Αναζήτησης

Περιεχόμενα Κεφαλαίου

Από το αντικείμενο των Δομών Δεδομένων είναι γνωστή η σημαντικότητα του προβλήματος της αναζήτησης καθώς εμφανίζεται σε πλήθος θεωρητικών και πρακτικών προβλημάτων. Στο παρόν κεφάλαιο συνοψίζονται δεδομένα αποτελέσματα αλλά παρουσιάζονται κατά αναλυτικότερο τρόπο.

8.1 Σειριακή Αναζήτηση

Η απλούστερη μέθοδος αναζήτησης είναι η **σειριακή** (sequential) **γραμμική** (linear). Αν και η μέθοδος είναι απολύτως γνωστή από το αντικείμενο των Δομών Δεδομένων, επαναλαμβάνουμε ελάχιστα σημεία ως εισαγωγή στο παρόν κεφάλαιο. Η επόμενη διαδικασία `sequential1` υποθέτει ότι αναζητείται η τιμή `key` στον πίνακα `A` που περιέχει `n` αταξινόμητα στοιχεία, και επιστρέφει τη θέση του κλειδιού στον πίνακα ή την τιμή 0 αν το κλειδί δεν υπάρχει (περίπτωση ανεπιτυχούς αναζήτησης).

```
     procedure sequential1(key);
1.   i <-- 1;
2.   while (i<=n) do
3.      if A[i]=key then return i
4.      else i <-- i+1;
5.   return 0
```

Η διαδικασία αυτή μπορεί να βελτιωθεί υιοθετώντας την τεχνική του κόμβου φρουρού (sentinel), οπότε η μέθοδος θα υλοποιηθεί θεωρώντας μία ακόμη θέση στον πίνακα με τη μορφή `A[n+1] <-- key`. Αν και πρακτικά η διαδικασία θα βελτιωθεί, σε θεωρητικό επίπεδο σε κάθε περίπτωση θα ισχύουν οι επόμενες προτάσεις υποθέτοντας ότι η πιθανότητα αναζήτησης του κλειδιού `A[i]` είναι $p_i = 1/n$, για $1 \leq i \leq i$.

Πρόταση.
Η επιτυχής αναζήτηση σε πίνακα με n αταξινόμητα κλειδιά έχει πολυπλοκότητα $\Theta(1)$, $\Theta(n)$ και $\Theta(n)$ στην καλύτερη, στη χειρότερη και στη μέση περίπτωση, αντίστοιχα.

Απόδειξη.
Η καλύτερη και η χειρότερη περίπτωση συμβαίνουν όταν η αναζήτηση τερματίζεται με την εξέταση της πρώτης και της τελευταίας θέσης του πίνακα, αντίστοιχα.

Για τη μέση περίπτωση ισχύει ότι:

$$\mathrm{E} \;=\; \frac{1+2+\ldots+n}{n} \;=\; \frac{n+1}{2}$$

από όπου προκύπτει η αλήθεια της πρότασης. □

Πρόταση.
Η ανεπιτυχής αναζήτηση σε πίνακα με n αταξινόμητα κλειδιά έχει πολυπλοκότητα $\Theta(n)$ στην καλύτερη, στη χειρότερη και στη μέση περίπτωση.

Απόδειξη.
Σε κάθε περίπτωση θα σαρωθεί ολόκληρος ο πίνακας, δηλαδή $\mathrm{A} = n$, και άρα η πρόταση ισχύει. □

Αν υποθέσουμε ότι ο πίνακας περιέχει `n` ταξινομημένα στοιχεία, τότε για την επιτυχή αναζήτηση ισχύει η ανωτέρω πρόταση αλλά για να επιταχύνουμε την ανεπιτυχή αναζήτηση πρέπει να μετατρέψουμε τη διαδικασία `sequential1` ως εξής.

```
     procedure sequential2(key);
1.   done <-- false; i <-- 1;
2.   while (i<=n) do
3.      if A[i]>key then i <-- i+1
4.      else if A[i]=key then retun i
5.      else return 0
```

Πρόταση.
Η ανεπιτυχής αναζήτηση σε πίνακα με n ταξινομημένα κλειδιά έχει πολυπλοκότητα $\Theta(1)$, $\Theta(n)$ και $\Theta(n)$ στην καλύτερη, στη χειρότερη και στη μέση περίπτωση, αντίστοιχα.

Απόδειξη.
Όταν αναζητούμε ένα μη υπαρκτό κλειδί, τότε αυτό μπορεί να ανήκει σε ένα από $n+1$ μεσοδιαστήματα που (σχηματικά) δημιουργούνται από τις τιμές των στοιχείων του πίνακα επάνω στην ευθεία των ακεραίων. Η καλύτερη και η χειρότερη περίπτωση συμβαίνουν όταν το αναζητούμενο `key` είναι μικρότερο από το πρώτο στοιχείο και μεγαλύτερο από το τελευταίο στοιχείο του πίνακα, αντίστοιχα. Για τη μέση περίπτωση θα πρέπει να εξετάσουμε $n+1$ περιπτώσεις και να λάβουμε το μέσο όρο τους. Αν το `key` είναι μικρότερο από το πρώτο

στοιχείο, τότε αρκεί μία σύγκριση για να τερματισθεί η διαδικασία. Αν το key είναι μεγαλύτερο από το i-οστό και μικρότερο από το $(i+1)$-οστό στοιχείο (για $i < n$), τότε αρκούν $i+1$ συγκρίσεις. Αν το key είναι μεγαλύτερο από το n-οστό στοιχείο, τότε αρκούν n συγκρίσεις. Συνεπώς ισχύει:

$$\mathrm{A} = \frac{1+2+\ldots+n+n}{n+1} = \frac{1+2+\ldots+n+n+1-1}{n+1} = \frac{n+2}{2} - \frac{1}{n+1}$$

από όπου προκύπτει η αλήθεια της πρότασης. □

8.2 Δυαδική Αναζήτηση

Κλασικό παράδειγμα των αλγορίθμων της οικογενείας Διαίρει και Βασίλευε είναι η πανταχού παρούσα **δυαδική αναζήτηση** (binary search). Όπως γνωρίζουμε η αναζήτηση αυτή εφαρμόζεται σε πίνακες που περιέχουν ταξινομημένα στοιχεία. Υπενθυμίζοντας όσα τονίσθηκαν στο Κεφάλαιο 5.2 σχετικά με την αποτελεσματικότητα των αναδρομικών και των επαναληπτικών μεθόδων, στη συνέχεια παρουσιάζουμε την επαναληπτική διαδικασία binary_iterate και την αναδρομική διαδικασία binary_rec που δείχνουν την ίδια θεωρητική συμπεριφορά.

```
     procedure binary_iterate(key);
1.   bottom <-- 1; top <-- n;
2.   while (bottom<=top) do
3.      middle <-- (top+bottom) div 2;
4.      if A[middle]=key then return middle
5.      else if A[middle]>key then top <-- middle-1
6.      else bottom <-- middle+1
7.   return 0

     procedure binary_rec(key, left, right);
1.   if left>right then return 0;
2.   middle <-- (top+bottom) div 2;
3.   if A[middle]=key then return middle
4.   else if A[middle]>key then
5         binary_rec(key, left, middle-1)
5.   else binary_rec(key, middle+1, right);
```

Μπορούμε να περιγράψουμε τη λογική των προηγούμενων διαδικασιών ως εξής. Έστω ότι αναζητούμε το ακέραιο κλειδί key σε ένα πίνακα A[1..n] με ταξινομημένους ακεραίους αριθμούς. Συγκρίνουμε το κλειδί key με το

περιεχόμενο της μεσαίας θέσης του πίνακα `A`, που είναι η θέση `middle`. Στο σημείο αυτό τρία ενδεχόμενα μπορεί να συμβούν:

1. τα δύο στοιχεία είναι ίσα, οπότε ο σκοπός μας επιτεύχθηκε,
2. το `key` είναι μικρότερο από το `A[middle]`, οπότε είμαστε βέβαιοι ότι το `key` αποκλείεται να βρίσκεται στον υποπίνακα `A[middle..top]`. Έτσι συνεχίζουμε στον υποπίνακα `A[bottom..middle-1]` εξετάζοντας το μεσαίο στοιχείο του,
3. το `key` είναι μεγαλύτερο του `A[middle]`, οπότε το `key` σαφώς δεν βρίσκεται στον υποπίνακα `A[bottom..middle]`. Έτσι, συνεχίζουμε και πάλι εξετάζοντας το μεσαίο στοιχείο του υποπίνακα `A[middle+1..top]`.

Οι ανωτέρω διαδικασίες επιστρέφουν τη θέση του `key` μέσα στον πίνακα. Αν το `key` βρίσκεται πράγματι μέσα στον `A`, τότε επιστρέφει τη συγκεκριμένη θέση, ενώ στην αντίθετη περίπτωση επιστρέφει την τιμή 0.

Στη συνέχεια θα εξετάσουμε αναλυτικά τη δυαδική αναζήτηση. Μία βασική υπόθεση που γίνεται στο σημείο αυτό είναι ότι η σύγκριση είναι τριών δρόμων (3-way comparison). Δηλαδή, με μοναδιαίο κόστος αποφασίζουμε να επιλέξουμε ένα μεταξύ τριών δρόμων.

Πρόταση.
Η πολυπλοκότητα της δυαδικής αναζήτησης είναι λογαριθμική.

Απόδειξη.
Κάθε φορά που η σύγκριση του `key` με το μεσαίο στοιχείο του πίνακα δεν καταλήξει σε ισότητα, η σύγκριση επαναλαμβάνεται σε υποπίνακα μισού μεγέθους σε σχέση με το μέγεθος του αρχικού. Επομένως, εύκολα προκύπτει η αναδρομική εξίσωση:

$$\begin{aligned} T(0) &= 0 \\ T(n) &= 1 && \text{αν } \texttt{key} = \texttt{A[middle]} \\ &= 1 + T(\lfloor (n+1)/2 \rfloor - 1) && \text{αν } \texttt{key} < \texttt{A[middle]} \\ &= 1 + T(n - \lfloor (n+1)/2 \rfloor) && \text{αν } \texttt{key} > \texttt{A[middle]} \end{aligned}$$

Απλοποιούμε τη σχέση αυτή θεωρώντας τη χειρότερη περίπτωση (δηλαδή, αγνοούμε το δεύτερο σκέλος) και ότι $n = 2^k - 1$ για κάποιο ακέραιο αριθμό k. Έτσι προκύπτει:

$$T(2^k - 1) \;=\; 1 + T(2^{k-1} - 1)$$

με αρχική συνθήκη $T(0) = 0$. Έτσι διαδοχικά έχουμε:

$$\begin{aligned} T(n) &= 1 + \Big(1 + T(2^{k-2} - 1)\Big) \\ &= 1 + \Big(1 + (1 + T(2^{k-3} - 1))\Big) = \ldots \\ &= i + T(2^{k-i} - 1) \\ &= k + T(0) = k = \log(n+1) \end{aligned}$$

Έτσι, λοιπόν, για $n = 2^k - 1$ προκύπτει ότι η πολυπλοκότητα της δυαδικής αναζήτησης είναι λογαριθμική. Εύκολα μπορεί να αποδειχθεί ότι η πολυπλοκότητα είναι $\Theta(\lceil \log(n+1) \rceil)$ για τυχόν n. □

Εναλλακτικά μπορεί να διατυπωθεί η εξής πρόταση.

Πρόταση.
Η μέση τιμή του αριθμού των συγκρίσεων για μία επιτυχή ή μία ανεπιτυχή αναζήτηση δίνεται από τις σχέσεις:

$$\begin{aligned} \mathrm{E} &= k - \frac{2^k - k - 1}{n} \\ \mathrm{A} &= \lceil \log n \rceil \end{aligned}$$

όπου $k = \lfloor \log n \rfloor$.

Απόδειξη.
Έστω ότι $n = 2^k - 1 + m$, όπου $m < 2^k$. Τότε ισχύει:

$$\begin{aligned} \mathrm{E} &= \frac{1}{n} \sum_{i=1}^{k-1} i \times 2^{i-1} + k \times m \\ &= \frac{2^{k-1} \times (k-2) + 1 + k \times m}{n} = k - \frac{2^k - k - 1}{n} \end{aligned}$$

Η απόδειξη της περίπτωσης της ανεπιτυχούς αναζήτησης είναι προφανής. □

8.3 Αναζήτηση Παρεμβολής

Η αναζήτηση παρεμβολής προσομοιάζει με τον τρόπο αναζήτησης σε ένα λεξικό ή σε έναν τηλεφωνικό κατάλογο. Κατά την αναζήτηση μίας λέξης, δεν ανοίγουμε το λεξικό αρχικά στη μέση, και μετά στο 1/3 ή στα 3/4, κοκ, δηλαδή δεν

συνηθίζεται να κάνουμε δυαδική αναζήτηση. Για παράδειγμα, αν αναζητούμε μία λέξη που αρχίζει από Α, τότε ανοίγουμε το λεξικό προς την αρχή, αν αρχίζει από Ε πάλι ανοίγουμε προς την αρχή αλλά όχι τόσο κοντά στο Α, ενώ αν αρχίζει από Ω τότε ανοίγουμε προς το τέλος του λεξικού. Στη συνέχεια, αναλόγως αν η λέξη είναι λεξικογραφικά μεγαλύτερη ή μικρότερη, κινούμαστε προς το τέλος ή την αρχή του λεξικού παραλείποντας έναν αριθμό σελίδων, πριν λάβουμε την επόμενη απόφαση. Το πλήθος των παραλειπόμενων σελίδων είναι ανάλογο της αλφαβητικής απόστασης της σελίδας όπου βρισκόμαστε από τη ζητούμενη λέξη. Παραδείγματος χάριν, αν είμαστε στο Ε και η ζητούμενη λέξη αρχίζει από Η ή από Μ, τότε στην πρώτη περίπτωση θα μεταπηδήσουμε σε μικρότερη απόσταση σε σχέση με τη δεύτερη. Στην ουσία αυτή η τακτική αρχικά προσπαθεί να προσεγγίσει τη θέση όπου μπορεί να βρίσκεται το ζητούμενο στοιχείο και μετά να κινηθεί αναλόγως.

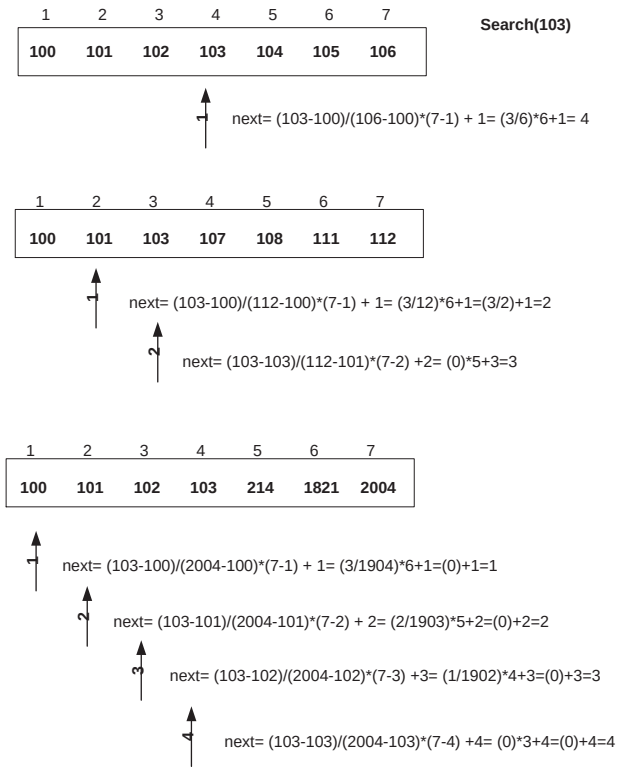
Έτσι λοιπόν, στην **αναζήτηση με παρεμβολή** (interpolation search) όταν αναζητούμε το στοιχείο x που βρίσκεται μεταξύ του $A[1]$ και του $A[n]$ ή γενικότερα μεταξύ του $A[left]$ και του $A[right]$, τότε το στοιχείο που επιλέγεται για εξέταση είναι το στοιχείο που βρίσκεται στη θέση

$$\left\lfloor \frac{x - A[left]}{A[right] - A[left]} \right\rfloor \tag{8.1}$$

Στην ουσία εξετάζοντας πόσο μεγαλύτερο είναι το ζητούμενο στοιχείο από το στοιχείου του αριστερού άκρου, καθώς και πόσο μεγαλύτερο είναι το στοιχείου του δεξιού από το στοιχείου του αριστερού άκρου, γίνεται εκτίμηση της θέσης του x λαμβάνοντας το λόγο των δύο διαφορών. Ο επόμενος ψευδοκώδικας υλοποιεί τη μέθοδο.

```
     procedure interpolation(key)
1.   low <-- 1; high <-- n;
2.   while (A[high]>=key) and (key>A[low]) do
3.      next <-- low+trunc((key-A[low])/(A[high]-A[low])*(high-low));
4.      if key>table[mid] then low <-- mid+1
5.      else if key<table[mid] then high <-- mid-1
6.      else low <-- mid;
7.   if key=A[low] then return low else return 0;
```

Στο παράδειγμα του Σχήματος 8.1 παρατηρούμε ότι όταν τα στοιχεία είναι ακέραιοι που διαφέρουν κατά μία μονάδα το ένα από το άλλο, τότε αυτή η εκτίμηση είναι ακριβής. Επίσης, αν τα στοιχεία διαφέρουν από τα γειτονικά τους



Σχήμα 8.1: Αναζήτηση παρεμβολής.

κατά μία μικρή ποσότητα, και πάλι η εκτίμηση θα είναι σχετικά ακριβής. Όμως, όταν οι διαφορές μεταξύ των στοιχείων είναι σημαντικές, τότε οι εκτιμήσεις δεν θα είναι επαρκώς ακριβείς και θα χρειασθούν περισσότερες επαναλήψεις.

Ο χρόνος χειρότερης περίπτωσης για την αναζήτηση παρεμβολής είναι $O(n)$, ενώ ο μέσος χρόνος είναι $O(\log \log n)$. Στη μέση περίπτωση η αναζήτηση παρεμβολής φαίνεται να υπερισχύει της δυαδικής αναζήτησης και αυτό είναι λογικό αφού τα προς εξέταση διαστήματα δεν υποδιπλασιάζονται σε κάθε βήμα αλλά μπορεί να γίνουν πολύ μικρότερα. Παρόλα αυτά, πειραματικά αποτελέσματα έδειξαν ότι η αναζήτηση παρεμβολής δεν εκτελεί πολύ λιγότερες συγκρίσεις έτσι ώστε να αποφεύγεται το αυξημένο υπολογιστικό κόστος εύρεσης του `next` και να έχει καλύτερους χρόνους, εκτός και αν οι πίνακες είναι πολύ μεγάλοι. Για το λόγο αυτό, συχνά είναι προτιμότερο τα πρώτα βήματα να εκτελούν αναζήτηση

παρεμβολής, έτσι ώστε το διάστημα να μειώνεται δραματικά, και στη συνέχεια να εκτελείται δυαδική αναζήτηση. Στη συνέχεια θα μελετήσουμε αυτήν ακριβώς την παραλλαγή.

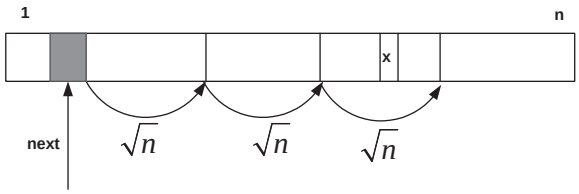
8.4 Δυαδική Αναζήτηση Παρεμβολής

Ακολουθεί ο ψευδοκώδικας της **δυαδικής αναζήτησης παρεμβολής** (binary interpolation search) ενώ στη συνέχεια θα σχολιασθεί και θα αναλυθεί η πολυπλοκότητα της μεθόδου.

```
     procedure interpolation_binary(key)
1.   left <-- 1; right <-- n; size <-- right-left+1;
2.   next <-- trunc(size * (key-A[left]) / (A[right]-A[left]));
3.   while (key<>A[next]) do
4.      i <-- 0; size <-- right-left+1;
5.      if (size <= 3)      {Απευθείας αναζήτηση}
6.      if (key => A[next])
7.         while (key>A[next+i*sqrt(size)-1] do
8.            i <-- i+1;
9.         right <-- next + i*sqrt(size);
10.        left <-- next + (i-1)*sqrt(size);
11.     else if (key<A[next])
12.        while (key<A[next-i*sqrt(size)+1] do
13.           i <-- i+1;
14.        right <-- next - (i-1)*sqrt(size);
15.        left <-- next - i*sqrt(size);
16.     next <-- left + trunc((right-left+1) * (key-A[left]) /
                      (A[right]-A[left]))-1;
17.  if (key=A[next]) then return next
18.  else return 0;
```

Σχετικά με τον ανωτέρω κώδικα, που είναι ενδεικτικός περισσότερο και σε καμία περίπτωση δεσμευτικός, παρατηρούμε τα εξής:

- Στην ουσία μετά το βήμα παρεμβολής (γραμμές 2 και 16, έκφραση 8.1) αναζητούμε γραμμικά με άλματα μεγέθους $\sqrt{\texttt{size}}$ για να προσδιορισθεί το υποδιάστημα που περιέχει το key. Στη συνέχεια επαναλαμβάνεται η ίδια διαδικασία για το νέο διάστημα, όπως φαίνεται στο Σχήμα 8.2.



Σχήμα 8.2: Αναζήτηση δυαδικής παρεμβολής.

- Για να γίνει απλούστερος ο ψευδοκώδικας, στη γραμμή 5 θεωρούμε ότι για σχετικά μικρά μεγέθη εκτελούμε γραμμική αναζήτηση, χωρίς αυτό να επηρεάζει την ασυμπτωτική συμπεριφορά του αλγορίθμου. Για παράδειγμα, κάλλιστα θα μπορούσε να εκτελεσθεί γραμμική αναζήτηση και για `size` μικρότερο του 5.
- Ο ανωτέρω ψευδοκώδικας δεν χειρίζεται κάποιες ακραίες περιπτώσεις, όπως όταν το αναζητούμενο στοιχείο `key` είναι μεγαλύτερο από κάθε στοιχείο του πίνακα `A` ή αν ενδεχομένως το βήμα μήκους $\sqrt{\texttt{size}}$ υπερβεί το δεξιό άκρο του υποπίνακα. Η διαχείριση των προβλημάτων αυτών είναι απλούστατη και αφήνεται ως άσκηση στον αναγνώστη.

Πρόταση.
Ο μέσος χρόνος της δυαδικής αναζήτησης παρεμβολής είναι $O(\log\log n)$.

Απόδειξη.
Έστω ότι όλα τα στοιχεία του πίνακα `A` επιλέγονται ισοπίθανα από το διάστημα [0,1]. Επίσης, έστω ότι C είναι το πλήθος των συγκρίσεων μέχρι να βρεθεί το κατάλληλο υποδιάστημα μεγέθους $\sqrt{\texttt{n}}$ που περιέχει το `key`. Τέλος, έστω P_i η πιθανότητα, ότι θα χρειασθούν τουλάχιστον i συγκρίσεις για να προσδιορισθεί το επίμαχο υποδιάστημα. Τότε ο αριθμός συγκρίσεων θα είναι:

$$C \;=\; \sum_{i\geq 1} i\,(P_i - P_{i+1}) \;=\; \sum_{i\geq 1} P_i$$

Είναι βέβαιο ότι θα χρειασθούν 2 συγκρίσεις τουλάχιστον και επομένως ισχύει: $P_1 = P_2 = 1$. Για το P_i ισχύει:

$$P_i \;\leq\; \text{Prob}\left[\; |\texttt{position\_of\_key} - \texttt{next}| \geq (i-2)\sqrt{n}\;\right], \qquad i \geq 3$$

Θα χρησιμοποιήσουμε την ανισο-ισότητα του Chebyschev:

$$\text{Prob}\left[|X-\mu| \geq \epsilon\right] \leq \frac{\sigma^2}{\epsilon^2}$$

με τυχαία μεταβλητή X, σταθερά ϵ, μέση τιμή μ και διασπορά σ.

Έστω λοιπόν X η τυχαία μεταβλητή, που λαμβάνει τιμές το πλήθος των $A_i < key$, όπου A_i είναι τα στοιχεία του πίνακα `A`. Αν q_j είναι η πιθανότητα το key να ισούται με j και τα $A_1, A_2, ..., A_n$ επιλέγονται ισοπίθανα, τότε η πιθανότητα p ώστε ένα A_i να είναι μικρότερο του key είναι:

$$p \;=\; \frac{key - A_0}{A_{n+1} - A_1}$$

και

$$q_j \;=\; \binom{n}{j}\, p^j\, (1-p)^{n-j}$$

Δηλαδή, είναι μία δυωνυμική κατανομή με $\mu = pn$ και $\sigma^2 = p(1-p)n \leq \frac{n}{4}$. Παρατηρούμε ότι $\mu = next$, δηλαδή ότι ο δείκτης παρεμβολής είναι ίσος με τη μέση τιμή της X. Άρα, ισχύει:

$$\begin{aligned} P_i &\leq \text{Prob}\left[\ |\texttt{position\_of\_key} - \texttt{next}| \geq (i-2)\sqrt{n}\ \right] \\ &\leq \frac{p(1-p)n}{((i-2)\sqrt{n})^2} \;=\; \frac{p(1-p)}{(i-2)^2} \;\leq\; \frac{1}{4(i-2)^2} \end{aligned}$$

Συνεπώς, ισχύει:

$$C \;\leq\; 2 + \sum_{i \geq 3} \frac{1}{4(i-2)^2} \;=\; 2 + \frac{\pi^2}{24} \;\approx\; 2.4$$

Αν $T(n)$ είναι το μέσο πλήθος των συγκρίσεων, τότε:

$$T(n) \;\leq\; C + T(\sqrt{n})$$

για $n \geq 3$ και $T(1) \leq 1, T(2) \leq 2$. Συνεπώς, τελικά ισχύει:

$$T(n) \;\leq\; 2 + 2.4 \log\log n$$

□

Πρόταση.
Στην χειρότερη περίπτωση ο χρόνος της δυαδικής αναζήτησης παρεμβολής είναι $O(\sqrt{n})$.

Απόδειξη.
Στην πρώτη εκτέλεση του βρόγχου `while` της εντολής 12 εκτελούνται το πολύ

$\sqrt{\texttt{n}}$ συγκρίσεις, στη δεύτερη το πολύ $\sqrt{\sqrt{\texttt{n}}}$, στην τρίτη $\sqrt{\sqrt{\sqrt{\texttt{n}}}}$ κοκ. Δηλαδή ισχύει:

$$n^{\frac{1}{2}} + n^{\frac{1}{4}} + n^{\frac{1}{8}} + \ldots \;=\; \mathrm{O}(\sqrt{n})$$

□

Ο χρόνος χειρότερης περίπτωσης μπορεί να βελτιωθεί από $\mathrm{O}(\sqrt{n})$ σε $\mathrm{O}(\log n)$ χωρίς να χειροτερεύει ο χρόνος μέσης περίπτωσης. Αντί το i να αυξάνεται γραμμικά (δηλαδή, με την εντολή 13: `i <-- i+1`) μέσα στο βρόγχο `while`, αυτό να αυξάνεται εκθετικά (δηλαδή, με μία εντολή: `i <-- 2*i`). Έτσι τα άλματα γίνονται συνεχώς μεγαλύτερα κρατώντας το αριστερό άκρο σταθερό (όπως στο Σχήμα 8.3), με αποτέλεσμα το τελευταίο υποδιάστημα να είναι πολύ μεγαλύτερο από $\sqrt{n}$. Μόλις βρεθεί αυτό το υποδιάστημα εφαρμόζουμε δυαδική αναζήτηση στα στοιχεία μέσα σε αυτό που απέχουν κατά $\sqrt{n}$ και έτσι προκύπτει το ζητούμενο υποδιάστημα μεγέθους $\sqrt{n}$. Πιο συγκεκριμένα:

1. Με μεγάλα εκθετικά βήματα εξέτασε τα διαστήματα: $[next, next + \sqrt{n}]$, $[next, next + 2\sqrt{n}], [next, next + 2^2\sqrt{n}], [next, next + 2^3\sqrt{n}], \ldots,$ $[next,$ $next + 2^i\sqrt{n}]$ μέχρι να βρεθεί j τέτοιο ώστε:

$$A[next + 2^{j-1}\sqrt{n}] \;<\; key \;\leq\; A[next + 2^j\sqrt{n}]$$

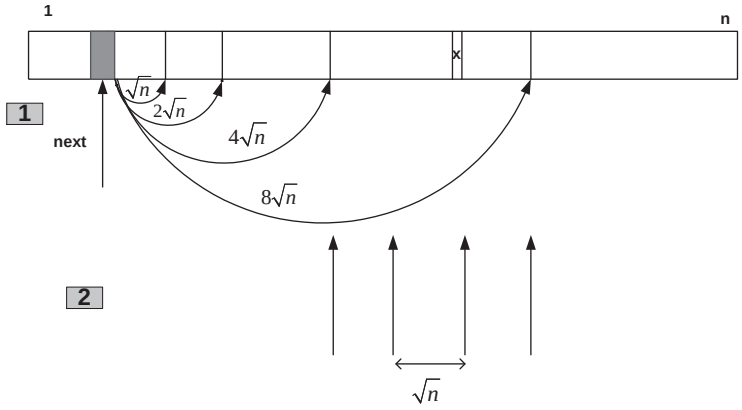
2. Αναζήτησε δυαδικά στα στοιχεία των θέσεων: $next + 2^{j-1}\sqrt{n}, next +$ $[2^{j-1}+1]\sqrt{n}, next+[2^{j-1}+2]\sqrt{n}, \ldots, next+2^j\sqrt{n}$. Δηλαδή, ψάξε δυαδικά στα στοιχεία που απέχουν απόσταση $\sqrt{n}$ στο διάστημα που βρέθηκε από το ερώτημα 1. Έτσι, προκύπτει τελικά ένα διάστημα μεγέθους $\sqrt{n}$.

Έστω ότι για το πρώτο βήμα εκτελούμε στη χειρότερη περίπτωση i συγκρίσεις. Αυτό σημαίνει ότι:

$$2^i\sqrt{n} \;=\; n \quad\Rightarrow\quad i \;=\; \log\sqrt{n}$$

Συνεπώς, για το πρώτο βήμα χρειάζεται χρόνο $\mathrm{O}(\log\sqrt{n})$, ενώ για το δεύτερο βήμα χρειάζεται χρόνος $\log 2^{i-1} = O(i-1) = O(\log\sqrt{n})$. Έτσι προκύπτει ότι κάθε επανάληψη του `while` θα χρειάζεται $\mathrm{O}(\log i)$ χρόνο στη χειρότερη περίπτωση. Άρα, συνολικά θα χρειασθεί χρόνος:

$$\begin{aligned}
\log\sqrt{n} + \log\sqrt{\sqrt{n}} + \log\sqrt{\sqrt{\sqrt{n}}} + \ldots \;&=\; \log n^{\frac{1}{2}} + \log n^{\frac{1}{4}} + \log n^{\frac{1}{8}} + \ldots \\
&=\; \frac{1}{2}\log n + \frac{1}{4}\log n + \frac{1}{8}\log n + \ldots \\
&=\; (\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots)\log n \\
&=\; \mathrm{O}(\log n)
\end{aligned}$$



Σχήμα 8.3: Βελτίωση με αναζήτηση άλματος.

Ο χρόνος μέσης περίπτωσης δεν αλλάζει, υπό την προϋπόθεση ότι όλα τα στοιχεία του πίνακα `A` επιλέχθηκαν ισοπίθανα από το διάστημα [0,1]. Αυτό συμβαίνει διότι σε κάθε επανάληψη της αναδρομικής σχέσης το κόστος είναι $O(\log i \leq n)$, συνεπώς συνολικά το αναμενόμενο κόστος θα είναι $O(\log \log n)$. Η περίπτωση τα στοιχεία του πίνακα `A` να μην επιλέχθηκαν ισοπίθανα από το διάστημα [0,1] εξετάζεται στη συνέχεια.

Κάθε μέθοδος αναζήτησης (δυαδική ή παρεμβολής) προσπαθεί να εντοπίσει το προς αναζήτηση στοιχείο περιορίζοντας σε κάθε επανάληψη το σχετικό διάστημα. Ο περιορισμός αυτός γίνεται με επιλογή ενός στοιχείου στο τρέχον διάστημα και με βάση αυτό το στοιχείο γίνεται ο διαχωρισμός του διαστήματος σε δύο υποδιαστήματα. Στη δυαδική αναζήτηση γίνεται διχοτόμηση επιλέγοντας το μεσαίο στοιχείο του διαστήματος σύμφωνα με τον τύπο:

$$next_{BIN} \leftarrow \left\lfloor \frac{left + right}{2} \right\rfloor$$

ενώ στην αναζήτηση παρεμβολής επιλέγοντας το στοιχείο σύμφωνα με τον τύπο:

$$next_{INT} \leftarrow \left\lfloor \frac{key - A[left]}{A[right] - A[left]} (right - left) \right\rfloor + left$$

Στη βιβλιογραφία έχουν προταθεί συνθετότεροι αλλά και αποτελεσματικότεροι τρόποι, με συνεχείς εναλλαγές στον τρόπο επιλογής του στοιχείου `next`.

Μέθοδος Alternate. Το στοιχείο διαχωρισμού υπολογίζεται στη μία επανάληψη με βάση τη δυαδική αναζήτηση, ενώ στην επόμενη με βάση την αναζήτηση παρεμβολής κοκ. Δηλαδή:

- Αν $i = 1 \mod 2$, τότε $next \leftarrow next_{INT}$, ενώ
- Αν $i = 0 \mod 2$, τότε $next \leftarrow next_{BIN}$

Μέθοδος Retrieve. Σε κάθε i-οστή επανάληψη της μεθόδου το στοιχείο διαχωρισμού επιλέγεται ως εξής:

- Αν $i = 1 \mod 3$, τότε $next \leftarrow min\{next^+, right - 1\}$
- Αν $i = 2 \mod 3$, τότε $next \leftarrow min\{next^-, left + 1\}$
- Αν $i = 0 \mod 3$, τότε $next \leftarrow next_{BIN}$

όπου

$$next^+ \leftarrow \lceil next_{INT} + \Delta \rceil$$

$$next^- \leftarrow \lceil next_{INT} - \Delta \rceil$$

$$\Delta \leftarrow \theta(right - left)(A[right] - A[left])^{\alpha} + \phi\sqrt{right - left}$$

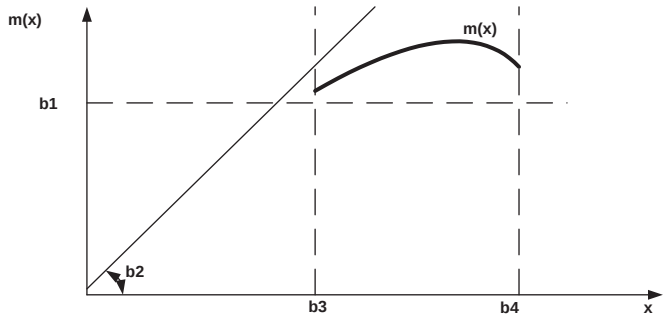
ενώ τα $0 < \alpha \leq 1$ και $\phi, \theta > 0$ είναι παράμετροι της μεθόδου.

Πρόταση.
Όταν τα στοιχεία του πίνακα A επιλέγονται από το διάστημα [0,1] με ομαλή κατανομή, τότε ο μέσος χρόνος για την Alternate είναι $O(\sqrt{\log n})$, ενώ για την Retrieve είναι $O(\log \log n)$. □

Ορισμός.
Μία κατανομή $m(x)$ καλείται **ομαλή** αν υπάρχουν b_1, b_2, b_3, b_4 για τα οποία ισχύει:

1. $m(x) \geq b_1 > 0$ όταν $b_3 < x < b_4$
2. $|m'(x)| \leq b_2$
3. $m(x) = 0$ όταν $x \leq b_3$ και $x \geq b_4$

όπως παρουσιάζεται στο παράδειγμα του Σχήματος 8.4. □



Σχήμα 8.4: Ομαλή κατανομή.

8.5 Κατακερματισμός

Με τον όρο **κατακερματισμός** (hashing) δηλώνεται ένα πολύ ευρύ αντικείμενο, τόσο σε θεωρητικό επίπεδο όσο και σε πρακτικό. Το πρόβλημα του σχεδιασμού αποτελεσματικών τεχνικών κατακερματισμού προκύπτει λόγω των πολλών διαφορετικών περιβαλλόντων και εφαρμογών όπου μπορεί να εφαρμοσθεί, όπως σε περιπτώσεις μεταγλωττιστών, βάσεων δεδομένων, ανάκτησης πληροφοριών κλπ. Υπό προϋποθέσεις οι μέθοδοι του κατακερματισμού έχουν πολύ καλή και σταθερή επίδοση, που υπερτερεί της δυαδικής αναζήτησης που εξετάσαμε προηγουμένως.

Εναλλακτική ονομασία του κατακερματισμού είναι **μετασχηματισμός κλειδιού σε διεύθυνση** (key to address transfrmation). Ο μετασχηματισμός αυτός επιτυγχάνεται με τη βοήθεια της **συνάρτησης κατακερματισμού** (hashing function), η οποία όμως δεν είναι αμφιμονοσήμαντη. Δηλαδή, είναι δυνατόν δύο διαφορετικά κλειδιά να αντιστοιχίζονται στην ίδια διεύθυνση του πίνακα. Το φαινόμενο αυτό ονομάζεται **σύγκρουση** (collision) και αποτελεί το μειονέκτημα της μεθόδου. Ουσιαστικά, κάθε μέθοδος κατακερματισμού προτείνει και μία διαφορετική τεχνική επίλυσης των συγκρούσεων. Το ερώτημα που τίθεται είναι πόσο συχνά μπορεί να εμφανισθεί περίπτωση σύγκρουσης. Στο ερώτημα αυτό απαντά το επόμενο παράδειγμα του παραδόξου των γενεθλίων (birthday paradox).

Το παράδοξο των γενεθλίων
Έστω ότι επιλέγονται τυχαία άτομα από το πλήθος. Κάθε φορά ελέγχουμε αν το επιλεγόμενο άτομο έχει γενέθλια την ίδια ημέρα με τους ήδη επιλεγέντες. Η ερώτηση είναι πόσα άτομα πρέπει να επιλεγούν ώστε η πιθανότητα δύο ή περισσότερα άτομα να έχουν την ίδια ημέρα γενέθλια να είναι περισσότερο από

50%. Κατ'αρχήν υποθέτουμε ότι όλες οι ημέρες του χρόνου είναι ισοπίθανο να είναι ημέρες γενεθλίων. Σε μία τέτοια περίπτωση, η πιθανότητα δύο άτομα να έχουν την ίδια ημέρα γενέθλια είναι $\frac{1}{365}$, οπότε η πιθανότητα δύο άτομα να μην έχουν την ίδια ημέρα γενέθλια είναι 1-$\frac{1}{365}$. Ομοίως, η πιθανότητα το τρίτο επιλεγόμενο άτομο να μην έχει γενέθλια με τους δύο προηγούμενους επιλεγέντες είναι 1-$\frac{2}{365}$. Με τη λογική αυτή καταλήγουμε ότι πρέπει να προσδιορίσουμε το i εκείνο για το οποίο ισχύει η σχέση:

$$\left(1-\frac{1}{365}\right)\ \left(1-\frac{2}{365}\right)\ \ldots\ \left(1-\frac{i}{365}\right)\ \geq\ 1/2$$

Αξιοποιώντας τη σχέση $1+x \leq e^x$ (που ισχύει για κάθε x όπως είδαμε στο Κεφάλαιο 2), προκύπτει η ισοδύναμη σχέση:

$$e^{-1/365}\ e^{-2/365}\ \ldots\ e^{-i/365}\ \geq\ 1/2\ \ \Rightarrow$$

$$e^{-1/365\sum_{j=1}^{i} j}\ \geq\ 1/2\ \ \Rightarrow$$

$$-\frac{1}{365}\sum_{j=1}^{i} j\ \geq\ \ln\frac{1}{2} = -\ln 2\ \ \Rightarrow$$

$$\frac{1}{365}\ \frac{i(i+1)}{2}\ \leq\ \ln 2$$

Με κατάλληλη άλγεβρα και λύνοντας την παραγόμενη τετραγωνική εξίσωση φθάνουμε στο συμπέρασμα ότι η ζητούμενη τιμή είναι $i = 23$. Το αποτέλεσμα αυτό θεωρείται παράδοξο, καθώς κάποιος θα ανέμενε ότι η συγκεκριμένη πιθανότητα (δηλαδή, δύο ή περισσότερα άτομα να έχουν την ίδια ημέρα γενέθλια να είναι περισσότερο από 50%) θα συνέβαινε για ένα μεγαλύτερο σύνολο ατόμων. Από το παράδοξο αυτό μπορούμε να σκεφτούμε την αναλογία σε σχέση με τον κατακερματισμό, δηλαδή ότι οι 365 ημέρες αντιστοιχούν σε θέσεις ενός πίνακα, ενώ τα άτομα αντιστοιχούν σε κλειδιά που εισάγονται στον πίνακα. Με βάση τη δυωνυμική κατανομή προκύπτει ότι η πιθανότητα κατά την εισαγωγή n κλειδιών, k από αυτά να συμπέσουν στην ίδια θέση είναι:

$$\binom{n}{k}\ \left(\frac{1}{n}\right)^k\ \left(1-\frac{1}{n}\right)^{n-k}$$

Για n αρκετά μεγάλο η προηγούμενη σχέση προσεγγίζεται από $1/(ek!)$. Από την προηγούμενη δυωνυμική κατανομή συνάγεται ότι αν σε πίνακα 1000 θέσεων εισαχθούν $n = 1000$ κλειδιά, τότε η πιθανότητα σε μία θέση του πίνακα να αντιστοιχηθούν k κλειδιά δίνεται από τον Πίνακα 8.1. Συνεπώς, η πιθανότητα να παρουσιασθούν συγκρούσεις είναι σημαντικότατη και για το λόγο αυτό έχουν προταθεί τόσες πολλές τεχνικές αντιμετώπισης του προβλήματος.

k	πιθανότητα k συγκρούσεων
0	37%
1	37%
2	18%
3	6%
4	1.5 %
5	0.3%

Πίνακας 8.1: Πιθανότητα σύγκρουσης σε πίνακα 100 θέσεων.

8.6 Γραμμική Αναζήτηση

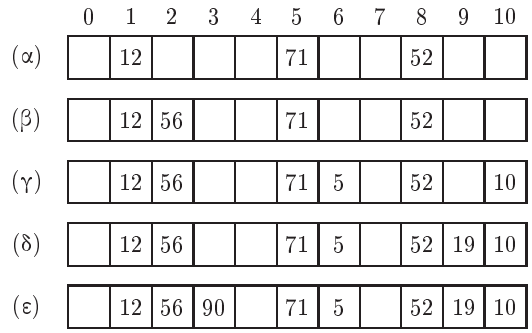
Ο **κλειστός κατακερματισμός** (closed hashing) είναι μία οικογένεια αλγορίθμων που δεν χρησιμοποιούν δείκτες για το χειρισμό των πινάκων. Στην οικογένεια αυτή ανήκουν αρκετές τεχνικές, μεταξύ των οποίων η **γραμμική αναζήτηση** (linear probing), η **τετραγωνική αναζήτηση** (quadratic probing), και ο **διπλός κατακερματισμός** (double hashing). Την απλούστερη τεχνική της γραμμικής αναζήτησης θα εξετάσουμε στη συνέχεια, ενώ στο Κεφάλαιο 8.7 θα εξετάσουμε την τετραγωνική αναζήτηση.

Η επόμενη διαδικασία `insert` παριστά τον αλγόριθμο εισαγωγής υποθέτοντας ότι ο πίνακας `A` έχει `m` θέσεις. Σύμφωνα με τη μέθοδο αυτή, κατά την εισαγωγή ενός κλειδιού καλείται η συνάρτηση κατακερματισμού που δίνει τη διεύθυνση του πίνακα όπου θα πρέπει να εισαχθεί το κλειδί. Αν η θέση δεν είναι κατειλημμένη, τότε το κλειδί εισάγεται και η διαδικασία τερματίζει. Αν η θέση είναι κατειλημμένη, τότε δοκιμάζεται η επόμενη θέση διαδοχικά μέχρι να τοποθετηθεί το κλειδί ή να γίνουν `m` προσπάθειες, γεγονός που δηλώνει ότι ο πίνακας είναι πλήρης.

```
     procedure insert(key);
1.   address <-- hash(key); count <-- 0;
2.   while (A[address]<>0) and (count<=m) do
3.      count <-- count+1;
4.      address <-- (address+1) mod m;
5.   if A[address]=0 then
6.      A[address] <-- key; return true
7.   else return false
```

Το επόμενο σχήμα δείχνει ένα παράδειγμα της δομής αυτής με $m = 11$ και $n = 8$. Πιο συγκεκριμένα, εισάγονται με τη σειρά τα κλειδιά 52, 12, 71, 56,

5, 10, 19 και 90, ενώ η χρησιμοποιούμενη συνάρτηση κατακερματισμού είναι: `hash(key) = key mod m`.



Σχήμα 8.5: Κατακερματισμός με γραμμική αναζήτηση.

Παρόμοιο είναι και το σκεπτικό του αλγορίθμου αναζήτησης, όπως φαίνεται στην επόμενη διαδικασία `search`, που επιστρέφει τη διεύθυνση του πίνακα όπου το κλειδί είναι αποθηκευμένο ή επιστρέφει την τιμή -1 αν το κλειδί δεν υπάρχει στον πίνακα.

```
     procedure search(key);
1.   address <-- hash(key); count <-- 0;
2.   while (A[address]<>key) and (count<=m) do
3.      count <-- count+1;
4.      address <-- (address+1) mod m
5.   if count=m then return -1 else return address
```

Θα αναλύσουμε την επίδοση της μεθόδου για την περίπτωση της επιτυχούς και της ανεπιτυχούς αναζήτησης, E και A αντίστοιχα, δηλαδή όταν αναζητούμε ένα υπαρκτό ή ανύπαρκτο κλειδί μέσα στον πίνακα. Υποθέτουμε ότι η συνάρτηση κατακερματισμού παράγει μία τυχαία διάταξη των κλειδιών, οπότε οι θέσεις στον πίνακα εξετάζονται με τυχαίο τρόπο. Αρχικά θα ασχοληθούμε με την ανεπιτυχή αναζήτηση. Οι παράμετροι που υπεισέρχονται στην ανάλυση της επίδοσης της μεθόδου είναι το μέγεθος m του πίνακα, το πλήθος των κλειδιών $n \leq m$ και ο **παράγοντας φόρτωσης** (load factor) $\alpha = n/m \leq 1$.

Πρόταση.
Η μέση τιμή του αριθμού των συγκρίσεων κατά τη γραμμική αναζήτηση, σε περίπτωση επιτυχούς και ανεπιτυχούς αναζήτησης είναι αντιστοίχως:

$$\mathrm{E} \approx \frac{1}{\alpha} \ln \frac{1}{1-\alpha}$$

και

$$\mathrm{A} \;\approx\; \frac{1}{1-\alpha}$$

Απόδειξη.
Ας υποθέσουμε ότι μία ανεπιτυχής αναζήτηση απαιτεί i προσπελάσεις ($1 \leq i \leq m$) στον πίνακα. Στην περίπτωση αυτή θα γίνουν $i-1$ προσπελάσεις σε κατειλημμένες θέσεις του πίνακα, ενώ η i-οστή θα προσπελάσει μία κενή θέση. Επομένως, αν εξαιρέσουμε αυτήν την ομάδα των i συνεχόμενων θέσεων, μένουν $m-i$ θέσεις εκ των οποίων κατειλημμένες είναι οι $n-i+1$ θέσεις. Το πλήθος των τρόπων που μπορεί να εμφανισθεί αυτή η περίπτωση είναι $\binom{m-i}{n-i+1}$, ενώ το σύνολο των περιπτώσεων είναι $\binom{m}{n}$. Συνεπώς, η πιθανότητα να απαιτηθούν i προσπελάσεις είναι:

$$P_i \;=\; \frac{\binom{m-i}{n-i+1}}{\binom{m}{n}}$$

Άρα ισχύει:

$$\mathrm{A} \;=\; \sum_{i=1}^{m} i\, P_i \;=\; (m+1) - \sum_{i=1}^{m} (m+1-i)\, P_i$$

Για το τελευταίο άθροισμα ισχύει:

$$(m+1-i)\, P_i \;=\; (m-i+1)\, \frac{\binom{m-i}{n-i+1}}{\binom{m}{n}} \;=\; \ldots \;=\; \frac{m-n}{\binom{m}{n}} \binom{m-i+1}{m-n}$$

Επομένως επιστρέφοντας στο A έχουμε:

$$\mathrm{A} \;=\; (m+1) - \frac{m-n}{\binom{m}{n}} \sum_{i=1}^{m} \binom{m-i+1}{m-n}$$

Με βάση την ταυτότητα των συνδυασμών, την οποία γνωρίζουμε από το Κεφάλαιο 2, έχουμε διαδοχικά:

$$\mathrm{A} \;=\; (m+1) - (m-n)\, \frac{\binom{m+1}{n}}{\binom{m}{n}} \;=\; \ldots \;=\; \frac{m+1}{m-n+1} \;\approx\; \frac{1}{1-\alpha}$$

Το αποτέλεσμα αυτό εξηγείται και διαισθητικά. Το α δηλώνει το ποσοστό των κατειλημμένων θέσεων, οπότε το $1-\alpha$ δηλώνει το ποσοστό των κενών θέσεων. Άρα αναμένουμε να εκτελέσουμε $\frac{1}{1-\alpha}$ προσπελάσεις πριν να εντοπίσουμε μία κενή θέση.

Για να φθάσουμε στην αντίστοιχη έκφραση για την επιτυχή αναζήτηση αρκεί να παρατηρήσουμε ότι το πλήθος των προσπελάσεων για την εύρεση ενός κλειδιού ισούται με το πλήθος των προσπελάσεων που είναι απαραίτητες για την εισαγωγή του (δηλαδή, $i-1$ προσπελάσεις σε κατειλημμένες θέσεις και μία προσπέλαση σε κενή θέση). Συνεπώς καταλήγουμε ότι:

$$\begin{aligned} \mathrm{E} &= \frac{1}{n}\sum_{i=0}^{n-1}\mathrm{A} = \frac{1}{n}\sum_{i=0}^{n-1}\frac{m+1}{m-i+1} \\ &= \frac{m+1}{n}\left(H_{m+1}-H_{m-n+1}\right) \approx \frac{m+1}{n}\left(\ln(m+1)-\ln(m-n+1)\right) \\ &= \frac{m+1}{n}\ln\frac{m+1}{m-n+1} \approx \frac{1}{\alpha}\ln\frac{1}{1-\alpha} \end{aligned}$$

□

Στις ίδιες εκφράσεις για τα E και A μπορούμε να καταλήξουμε στηριζόμενοι σε ένα διαφορετικό σκεπτικό. Έστω ότι με P_i συμβολίζουμε την πιθανότητα να απαιτηθούν ακριβώς i προσπελάσεις για μία ανεπιτυχή αναζήτηση, οπότε ισχύει:

$$\mathrm{A} = 1+\sum_{i=0}^{\infty} i\,P_i$$

Έστω επίσης ότι με Q_i συμβολίζουμε την πιθανότητα να απαιτηθούν τουλάχιστον i προσπελάσεις, οπότε ισχύει η σχέση:

$$\sum_{i=0}^{\infty} i\,P_i = \sum_{i=1}^{\infty} Q_i$$

Το πρόβλημα, λοιπόν, ανάγεται στον προσδιορισμό του Q_i. Προφανώς, ισχύει $Q_1=\alpha$, $Q_2 = Q_1\frac{n-1}{m-1}$, ενώ γενικώς ισχύει:

$$Q_i = \frac{n}{m}\,\frac{n-1}{m-1}\,\ldots\,\frac{n-i+1}{m-i+1} \le \left(\frac{n}{m}\right)^i = \alpha^i$$

Επανερχόμενοι στη σχέση για το $\mathrm{E}(n)$ έχουμε:

$$\mathrm{A} = \sum_{i=1}^{\infty} Q_i \le 1+\alpha+\alpha^2+\alpha^3+\ldots = \frac{1}{1-\alpha}$$

8.7 Τετραγωνική Αναζήτηση

Η επίδοση της γραμμικής αναζήτησης δεν είναι ιδιαίτερα ελκυστική και οφείλεται στο φαινόμενο της **πρωτεύουσας συγκέντρωσης** (primary clustering), δηλαδή της σύγκρουσης σε διάφορες περιοχές του πίνακα κλειδιών που προέρχονται από διαφορετικές αρχικές διευθύνσεις, δηλαδή δεν είναι συνώνυμα. Για τη μείωση της πρωτεύουσας συγκέντρωσης συχνά χρησιμοποιούνται μη γραμμικές μέθοδοι αναζήτησης της επόμενης διαθέσιμης θέσης. Μία τέτοια μέθοδος είναι η **τετραγωνική αναζήτηση** (quadratic search), που σκοπό έχει τη γρήγορη απομάκρυνση των κλειδιών από την αρχική περιοχή σύγκρουσης. Σύμφωνα με τη μέθοδο αυτή οι επόμενες θέσεις που θα εξετασθούν μπορούν να υπολογισθούν με βάση τη σχέση:

$$(c_1\ i^2 + c_2\ i + \texttt{key} \mod m) \mod m$$

όπου i είναι ο αύξων αριθμός της αναζήτησης (δηλαδή, $i = 0, 1, 2, \ldots$), ενώ τα c_1 και c_2 είναι αυθαίρετες σταθερές.

Στο Σχήμα 8.6 παρουσιάζεται η εισαγωγή της προηγούμενης ομάδας κλειδιών στον πίνακα των 11 θέσεων θεωρώντας ότι c_1=1 και c_2=2. Είναι εύκολη για τον αναγνώστη η κατανόηση του τρόπου που προκύπτουν οι διαδοχικές μορφές που παίρνει ο πίνακας μετά την αποθήκευση των κλειδιών 71, 56, 10 και 90 αντίστοιχα. Σημειώνεται ότι τα κλειδιά 56, 5 και 19 εισάγονται μετά από 2, 3, και 3 προσπάθειες, αντίστοιχα.

	0	1	2	3	4	5	6	7	8	9	10
(a)		12				71			52		
(b)		12			56	71			52		
(g)	5	12			56	71			52		10
(d)	5	12	90	19	56	71			52		10

Σχήμα 8.6: Τετραγωνικός κατακερματισμός.

Επίσης σημειώνεται ότι η προηγούμενη γενική σχέση μπορεί να απλοποιηθεί ως εξής:

$$(i^2 + \texttt{key} \mod m) \mod m$$

Για την περίπτωση αυτή έχει αποδειχθεί ότι δεν είναι απαραίτητο να υπολογίζεται το τετράγωνο που είναι χρονοβόρα πράξη, αλλά η θέση όπου θα γίνει η i-οστή σύγκριση μπορεί να προκύψει προσθέτοντας 2 μονάδες στο άλμα, που

χρησιμοποιήθηκε στην $(i-1)$-οστή σύγκριση. Αυτό φαίνεται από την επόμενη ταυτότητα, που μπορεί να αποδειχθεί με μαθηματική επαγωγή:

$$1+3+5+...+(2i-1) \;=\; i^2$$

Στη συνέχεια δίνεται η διαδικασία εισαγωγής με την τετραγωνική εξέταση, που στηρίζεται στην αντίστοιχη διαδικασία της γραμμικής εξέτασης. Είναι εύκολο για τον αναγνώστη να μετατρέψει τη διαδικασία `search` της γραμμικής εξέτασης ώστε να ισχύει για την τετραγωνική εξέταση.

```
     procedure quadratic_insert(key)
1.   location <-- Hash(key);
2.   if A[location]=0 then
3.      A[location] <-- key; return true;
4.   else
5.      count <-- 0; increment <-- 1;
6.      repeat
7.         count <-- count+1;
8.         location <-- (location+increment) mod m;
9.         increment <-- increment+2;
10.     until (A[location]=0) or (count=m div 2);
11.     if A[location]=0 then
12.        A[location] <-- key; return true;
13.     else return false;
```

Ένα μειονέκτημα της μεθόδου αυτής είναι ότι για μερικές τιμές του μεγέθους του πίνακα υπάρχει πιθανότητα να καταστεί αδύνατη η εισαγωγή ενός κλειδιού. Ειδικά αν $m = 2^k$, τότε μπορεί να προσπελασθούν πολύ λίγες θέσεις του πίνακα, ενώ διαφορετική είναι η περίπτωση όπου το m είναι πρώτος αριθμός. Στην περίπτωση αυτή μπορεί η i-οστή εξέταση να συμπίπτει με την $(m-i)$-οστή εξέταση.

Πρόταση.
Αν το m είναι πρώτος αριθμός, τότε κατά την τετραγωνική εξέταση μπορεί να προσπελασθούν μόνο οι μισές θέσεις του πίνακα.

Απόδειξη.
Έστω ότι η i-οστή και η j-οστή προσπάθεια, όπου το j είναι ο μικρότερος τέτοιος ακέραιος, συμπίπτουν στην ίδια θέση του πίνακα. Τότε ισχύει:

$$(h+i^2) \mod m \;=\; (h+j^2) \mod m$$

Συνεπώς προκύπτει:

$$(j-i) \times (j+i) \;=\; 0 \mod m$$

Επειδή τα j και i είναι άνισα πρέπει να ισχύει:

$$(j+i) \;=\; 0 \mod m$$

Άρα πρέπει ή το i ή το j είναι μεγαλύτερα από $m/2$. □

Επειδή ισχύει η προηγούμενη πρόταση ο πίνακας θεωρείται πλήρης όταν ισχύει $2n = m$. (όπως φαίνεται στα όρια της εντολής ελέγχου `until` της εντολής 10.) Το πρόβλημα αυτό της τετραγωνικής εξέτασης επιλύεται μερικά με την πρόσθεση διαδοχικά των όρων της ακολουθίας +1, –1, +4, –4, +9, –9 κλπ. επάνω στην αρχική ποσότητα της `location` (εντολή 1) και με κατάλληλη αναγωγή στο μέγεθος του πίνακα.

8.8 Διπλός Κατακερματισμός

Η τετραγωνική αναζήτηση αντιμετωπίζει το πρόβλημα της πρωτεύουσας συγκέντρωσης, υποφέρουν όμως από τη λεγόμενη **δευτερεύουσα συγκέντρωση** (secondary clustering). Ο λόγος είναι είτε ο τρόπος επιλογής της επόμενης θέσης, ο οποίος είναι κοινός για όλα τα κλειδιά, είτε η εξάρτηση από την αρχική διεύθυνση όπου κατευθύνεται ένα κλειδί προς αποθήκευση. Το πρόβλημα αυτό αντιμετωπίζεται με την τεχνική του **διπλού κατακερματισμού** (double hashing), που χρησιμοποιεί μία δεύτερη συνάρτηση μετασχηματισμού για να εντοπίσει την επόμενη διαθέσιμη θέση. Δηλαδή, έστω ότι χρησιμοποιείται η συνάρτηση της διαίρεσης:

$$h_1(\texttt{key}) \;=\; \texttt{key} \mod m$$

για την εύρεση της αρχικής θέσης αποθήκευσης του κλειδιού. Σε περίπτωση σύγκρουσης χρησιμοποιείται μία άλλη συνάρτηση για να δώσει την απόσταση σε θέσεις από την αρχική θέση. Σε περίπτωση νέας σύγκρουσης γίνεται δοκιμή σε θέση που απέχει την ίδια απόσταση από τη θέση της δεύτερης σύγκρουσης. Ένα συνηθισμένο παράδειγμα συνάρτησης επανα-κατακερματισμού είναι:

$$h_2(\texttt{key}) \;=\; (\texttt{key} \div m) \mod m$$

Δηλαδή, στη δεύτερη αυτή συνάρτηση πρώτα υπολογίζεται το πηλίκο του κλειδιού διαιρούμενου με το μήκος του πίνακα, κατόπιν υπολογίζεται το υπόλοιπο (κατά

τα γνωστά) που προστίθεται στο αποτέλεσμα της $h_1(\texttt{key})$. Σημειώνεται ότι αν το υπολογιζόμενο πηλίκο είναι ίσο με 0, τότε αυθαίρετα εξισώνεται με 1. Έτσι με τον τρόπο αυτό τα συνώνυμα που συγκρούονται σε μία αρχική θέση δεν ακολουθούν την ίδια διαδρομή στον πίνακα για την εύρεση μίας διαθέσιμης θέσης. Πρέπει επίσης να προσεχθεί ώστε το μέγεθος του πίνακα να είναι πρώτος αριθμός, γιατί αλλιώς για μερικά κλειδιά υπάρχει η δυνατότητα να μην προσπελασθούν όλες οι θέσεις του πίνακα για εξέταση. Από τα προηγούμενα γίνεται αντιληπτό γιατί η μέθοδος αυτή λέγεται και μέθοδος **γραμμικού πηλίκου** (linear quotient).

Ας εξετασθεί και πάλι το παράδειγμα εισαγωγής με τις γνωστές τιμές κλειδιών. Στο Σχήμα 8.7 φαίνεται διαδοχικά η μορφή του πίνακα μετά την εισαγωγή των κλειδιών 71, 56, 10 και 90 αντίστοιχα.

	0	1	2	3	4	5	6	7	8	9	10
(a)		12				71			52		
(b)		12				71	56		52		
(g)		12				71	56	5	52		10
(d)		12	90			71	56	5	52	19	10

Σχήμα 8.7: Διπλός κατακερματισμός.

Στη συνέχεια δίνεται η διαδικασία `double_insert` για την εισαγωγή στοιχείων σε πίνακα με τη μέθοδο του διπλού κατακερματισμού. Με `hash1` και `hash2` συμβολίζονται οι δύο συναρτήσεις κατακερματισμού που περιγράφηκαν προηγουμένως.

```
     procedure double_insert(key);
1.   location <-- hash1(key);
2.   if A[location]=0 then
3.      A[location] <-- key; retutn true;
4.   else
5.      increment <-- Hash2(key);
6.      last <-- (location+(n-1)*increment) MOD m;
7.      repeat location <-- (location+increment) mod m
8.     until (A[location]=0) or (location=last);
9.      if A[location]=0 then
10.        A[location] <-- key;
11.        retutn true;
12.     else retutn false;
```

Πρόταση.
Η μέση τιμή του αριθμού των συγκρίσεων για την επιτυχή και την ανεπιτυχή αναζήτηση σύμφωνα με τη μέθοδο του διπλού κατακερματισμού είναι αντίστοιχα:

$$\begin{aligned} \mathrm{E} &\approx -\frac{\ln(1-a)}{a} \\ \mathrm{A} &\approx \frac{1}{1-a} \end{aligned}$$

Απόδειξη.
Η πιθανότητα κάποια θέση να είναι κατειλημμένη είναι a. Επομένως για την ανεπιτυχή αναζήτηση ισχύει:

$$\begin{aligned} \mathrm{A} &= \sum_{k=1}^{m} k\ a^{k-1}\ (1-a) \ = \ \sum_{k=1}^{m} k\ a^{k-1} \ - \ \sum_{k=1}^{m} k\ a^{k} \\ &= \sum_{k=1}^{m} a^{k-1} - m\ a^{m} \end{aligned}$$

Αν το m είναι πολύ μεγάλο, τότε ο αφαιρετέος είναι πολύ μικρός, ενώ το άθροισμα μπορεί να θεωρηθεί ως άθροισμα απείρων όρων φθίνουσας ακολουθίας. Έτσι προκύπτει η αλήθεια της πρότασης.

Ο αριθμός των συγκρίσεων για μία επιτυχή αναζήτηση ισούται με τον αριθμό των συγκρίσεων για την εισαγωγή του αντίστοιχου κλειδιού. Αν θεωρηθεί ότι ο πίνακας δημιουργείται εισάγοντας τα κλειδιά ένα-προς-ένα, τότε ο παράγοντας φόρτωσης αυξάνει κατά μικρά βήματα. Άρα η τιμή του E ισούται με το μέσο όρο των τιμών του A, καθώς ο παράγοντας φόρτωσης αυξάνει από 0 ως a. Συνεπώς:

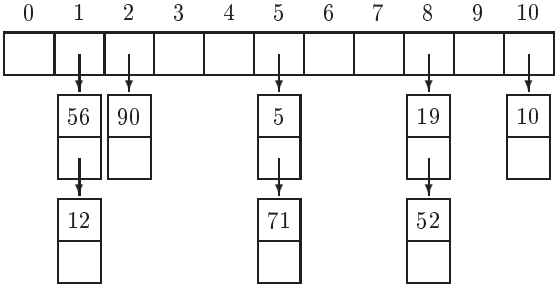
$$\mathrm{E} \ = \ \frac{1}{a}\int_0^a A\ \mathrm{d}a \ = \ \frac{1}{a}\int_0^a \frac{1}{1-a}\ \mathrm{d}a$$

από όπου προκύπτει η αντίστοιχη αλήθεια της πρότασης. □

8.9 Κατακερματισμός με Αλυσίδες

Αναφέραμε ότι η οικογένεια των μεθόδων του κλειστού κατακερματισμού δεν χρησιμοποιεί επιπλέον χώρο πέραν του χώρου του συγκεκριμένου πίνακα. Μία άλλη μεγάλη οικογένεια μεθόδων κατακερματισμού, η οικογένεια της **ανοικτής διεύθυνσης** (open addressing), χρησιμοποιεί αλυσίδες που ξεκινούν από τις m θέσεις του πίνακα και μπορούν να επεκταθούν δυναμικά. Σε επίπεδο τύπων

υποθέτουμε ότι υπάρχει μία δομή record με δύο πεδία, όπου το πρώτο πεδίο `data` αφορά στα καθαρά δεδομένα, ενώ το δεύτερο πεδίο `ptr` παριστά το δείκτη προς τον επόμενο κόμβο της αλυσίδας. Το διπλανό σχήμα δείχνει ένα παράδειγμα της δομής αυτής με $m = 11$ και $n = 8$ (τα ίδια κλειδιά με το προηγούμενο παράδειγμα). Πιο συγκεκριμένα, πρέπει να έχουμε υπόψη ότι η παραλλαγή αυτή λέγεται μέθοδος με **ξεχωριστές αλυσίδες** (separate chaining) καθώς υπάρχει και η μέθοδος των **σύμφυτων αλυσίδων** (coalesced chaining), η οποία όμως δεν θα μας απασχολήσει στη συνέχεια. Ο ψευδοκώδικας `chainsearch` που ακολουθεί δείχνει τη διαδικασία επιτυχούς και ανεπιτυχούς αναζήτησης.



Σχήμα 8.8: Κατακερματισμός με αλυσίδες.

```
     procedure chainsearch(key);
1.   address <-- hash(key); j <-- A[address];
2.   while (j<>nil) do
3.      if j.data=key then return true
4.      else j <-- j.ptr
5.   return false
```

Πρόταση.
Η μέση τιμή του αριθμού των συγκρίσεων κατά την αναζήτηση σε κατρακερματισμό με αλυσίδες, σε περίπτωση επιτυχούς και ανεπιτυχούς αναζήτησης είναι αντιστοίχως:

$$\mathrm{E} \;=\; 1 + \frac{\alpha}{2} - \frac{1}{2m}$$

και

$$\mathrm{A} \;\approx\; 1 + \alpha$$

Απόδειξη.
Θα εξετάσουμε κατ'αρχήν την περίπτωση της ανεπιτυχούς αναζήτησης. Υποθέτοντας μία τυχαία συνάρτηση κατακερματισμού, αναμένουμε ότι κάθε κλειδί θα κατευθυνθεί

σε οποιαδήποτε θέση από τις m θέσεις του πίνακα με την ίδια πιθανότητα $1/m$ (εντολή 1). Στη συνέχεια η αναζήτηση θα συνεχισθεί στην αντίστοιχη αλυσίδα (εντολές 3-5), που θα τη διασχίσει μέχρι το τέλος της, ώστε να επιστραφεί η ένδειξη `false`. Επομένως, η επίδοση εξαρτάται από το μήκος της αλυσίδας, και εφ'όσον συνολικά έχουν εισαχθεί n κλειδιά σε όλες τις m αλυσίδες, έπεται ότι το μέσος μήκος της αλυσίδας είναι $n/m = \alpha$. Άρα, η πολυπλοκότητα της ανεπιτυχούς αναζήτησης είναι $\Theta(1+\alpha)$ στη μέση περίπτωση.

Για να εξετάσουμε την περίπτωση της επιτυχούς αναζήτησης θα υποθέσουμε ότι κάθε νέο κλειδί εισάγεται στο τέλος της αντίστοιχης αλυσίδας. Επομένως το πλήθος των προσπελάσεων που θα πραγματοποιηθούν για μία επιτυχή αναζήτηση είναι μία μονάδα περισσότερο από το πλήθος που απαιτούνται για την εισαγωγή του αντίστοιχου κλειδιού. Όταν εισάγεται το i-οστό κλειδί, το μήκος της αντίστοιχης αλυσίδας είναι $(i-1)/m$. Επομένως ισχύει:

$$\begin{aligned} \mathrm{E} &= \frac{1}{n}\sum_{i=1}^{n}\left(1+\frac{i-1}{m}\right) = 1+\frac{1}{nm}\sum_{i=1}^{n}(i-1) \\ &= 1+\frac{1}{nm}\,\frac{(n-1)n}{2} = 1+\frac{\alpha}{2}-\frac{1}{2m} \end{aligned}$$

Συνεπώς και πάλι προκύπτει ότι η πολυπλοκότητα είναι $\Theta(1+\alpha)$ στη μέση περίπτωση. □

Στο σημείο αυτό πρέπει να προσεχθεί ότι στην περίπτωση της ανοικτής διεύθυνσης ισχύει $\alpha \leq 1$, αλλά στην περίπτωση των αλυσίδων δεν ισχύει αυτός ο περιορισμός. Επομένως η επίδοση της επιτυχούς και της ανεπιτυχούς αναζήτησης είναι ικανοποιητική όταν το n είναι της τάξης του m, επομένως το α είναι της τάξης της μονάδας.

8.10 Βιβλιογραφική Συζήτηση

Η δυαδική αναζήτηση απασχόλησε τους ερευνητές της Πληροφορικής στις αρχές της δεκαετίας του 1970. Οι πρώτες αναλύσεις της μεθόδου αναφέρονται στα άρθρα [42, 74]. Την περίοδο εκείνη η μέθοδος ήταν δημοφιλής και εκδοχές της δημοσιεύονταν σε πλήθος βιβλίων και άρθρων. Ωστόσο, τότε δεν ήταν προφανές αυτό που είναι σήμερα. Στο άρθρο [84] παρουσιάζονται και αναλύονται τα λάθη που είχαν παρεισφρήσει σε 26 δημοσιεύσεις της εποχής σχετικά με το θέμα. Εξ άλλου ο Bentley ισχυρίζεται λίγοι προγραμματιστές μπορούν να κωδικοποιήσουν τη μέθοδο χωρίς λάθη μέσα σε δύο ώρες [7]. Γενικεύσεις της μεθόδου μπορούν να βρεθούν στα άρθρα [48, 83].

Δημοφιλής μέθοδος αναζήτησης υπήρξε και η αναζήτηση με παρεμβολή. Προτάθηκε το 1977 από τους Perl-Reingold [126] και εν συνεχεία μελετήθηκε στα άρθρα [19, 56, 86, 125], αυτή καθώς και οι παραλλαγές της με σειριακή αναζήτηση [55] και με δυαδική αναζήτηση [136]. Οι παραλλαγές Alternate και Retrieve διαπτυπώθηκαν από τον Willard [172].

Πέραν των δύο αυτών μεθόδων, στη βιβλιογραφία αναφέρονται και άλλες μέθοδοι, που όμως δεν άντεξαν στο χρόνο. Για παράδειγμα, αναφέρεται η Fibonaccian μέθοδος [40, 120], η αναζήτηση άλματος [150, 73] και η πολυωνυμική αναζήτηση [149].

Η μέθοδος του κατακερματισμού υπήρξε εξίσου δημοφιλής και ερευνήθηκε σε βάθος τις δεκαετίες του 1970 και 1980. Οι πρώτες βασικές εργασίες σχετικά με τον κατακερματισμό ήταν τα άρθρα [103, 127]. Σχετικά με την επιλογή της κατάλληλης συνάρτησης κατακερματισμού είναι τα άρθρα [75, 107]. Θεμελιωτές της τετραγωνικής μεθόδου ήταν οι Bell και Radke [5, 129], ενώ αντίστοιχα στο διπλό κατακερματισμό αναφέρονται τα άρθρα [63, 94]. Κατά καιρούς έχουν εμφανισθεί διάφορες επισκοπήσεις της περιοχής, όπως τα άρθρα [30, 89, 104, 146]. Το βιβλίο των Gonnet-BeazaYates συμπεριλαμβάνει σε συνοπτικό τρόπο πλήθος μεθόδων κατακερματισμού και αντίστοιχες αναλύσεις [53]. Τη δεκαετία του 1990 η έρευνα στράφηκε στη σχεδίαση δυναμικών μεθόδων κατακερματισμού για χρήση σε δευτερεύουσα μνήμη και προέκυψαν πολλές νέες μέθοδοι. Οι μέθοδοι αυτές δεν θα εξετασθούν στα πλαίσια του βιβλίου αυτού, καθώς ταιριάζουν περισσότερο στο αντικείμενο των Βάσεων Δεδομένων (Data Bases). Ωστόσο, ο ενδιαφερόμενος αναγνώστης παραπέμπεται στην επισκόπηση [34].

Η Άσκηση 4 βασίζεται στο άρθρο [44], η Άσκηση 7 στο άρθρο [12], η Άσκηση 9 στο άρθρο [49], η Άσκηση 10 στα άρθρα [98, 99], η Άσκηση 11 στο άρθρο [91], ενώ η Άσκηση 12 στο άρθρο [3].

8.11 Ασκήσεις

1. Να βρεθεί η μέση τιμή του αριθμού των αναζητήσεων για την περίπτωση της ανεπιτυχούς σειριακής αναζήτησης σε ταξινομημένο πίνακα. Να θεωρηθεί ότι η πιθανότητα αναζήτησης ενός ανύπαρκτου κλειδιού από τα δημιουργούμενα διαστήματα μεταξύ των κλειδιών είναι ίση για όλα τα διαστήματα.

2. Δίνεται ταξινομημένος πίνακας με n στοιχεία, όπου επιτρέπεται η ύπαρξη πολλαπλών τιμών. Να τροποποιηθεί ο αλγόριθμος της δυαδικής αναζήτησης, ώστε να βρίσκει την πρώτη εμφάνιση του αναζητούμενου κλειδιού σε χρόνο της τάξης $\Theta(\log n)$.

3. Δίνεται ταξινομημένος πίνακας με n στοιχεία, όπου επιτρέπεται η ύπαρξη πολλαπλών τιμών. Να τροποποιηθεί ο αλγόριθμος της δυαδικής αναζήτησης, ώστε να βρίσκει το μεγαλύτερο κλειδί που είναι μικρότερο από το αναζητούμενο κλειδί, το οποίο όμως δεν υπάρχει (ανεπιτυχής αναζήτηση). Αν το αναζητούμενο είναι το μικρότερο στοιχείο του πίνακα, τότε να επιστρέφεται μήνυμα λάθους.

4. Η μέθοδος των **παραγεμισμένων λιστών** (padded lists) προσπαθεί να βελτιώσει τη δυαδική αναζήτηση χρησιμοποιώντας δύο πίνακες ίσου μεγέθους: ένα πίνακα για δεδομένα και ένα πίνακα για λογικές σημαίες. Αρχικά ο πίνακας δεδομένων περιέχει και κάποιες κενές θέσεις ομοιόμορφα διαμοιρασμένες μεταξύ των κατειλημμένων θέσεων. Στις κενές αυτές θέσεις αποθηκεύονται τυχαίες τιμές έτσι ώστε ο πίνακας να παραμένει ταξινομημένος, ενώ οι αντίστοιχες θέσεις του πίνακα σημαιών παίρνουν τιμές `false`. Η αναζήτηση ενός κλειδιού γίνεται στον πίνακα δεδομένων κατά τα γνωστά, αλλά παράλληλα ελέγχεται αν η αντίστοιχη λογική σημαία είναι `true`. Σε περίπτωση διαγραφής η σημαία από `true` γίνεται `false`. Τέλος, αν σε περίπτωση εισαγωγής η θέση δεν είναι κατειλημμένη, τότε η σημαία από `false` γίνεται `true`, ενώ αν η θέση είναι κατειλημμένη, τότε ακολουθείται μία διαδικασία ολισθήσεων. Για μεγαλύτερη αποτελεσματικότητα περιοδικά απαιτείται η αναδιοργάνωση των πινάκων, έτσι ώστε οι κενές θέσεις να μοιράζονται και πάλι ομοιόμορφα μεταξύ των κατειλημμένων θέσεων. Να σχεδιασθεί και να αναλυθούν αλγόριθμοι υλοποίησης των διαφόρων λειτουργιών των παραγεμισμένων λιστών (δηλαδή, εισαγωγή, διαγραφή, αναζήτηση).

5. Με βάση την δυαδική αναζήτηση να σχεδιασθεί και να αναλυθεί μέθοδος αναζήτησης σε ταξινομημένο πίνακα που να τον τριχοτομεί (αντί να τον διχοτομεί).

6. Ποιά είναι η πιθανότητα 3 άτομα μεταξύ 100 τυχαίων ατόμων να έχουν γενέθλια την ίδια ημέρα;

7. Έστω ένας πίνακας με n ταξινομημένα στοιχεία, όπου $n = i \times j$. Έτσι χωρίζουμε τον πίνακα σε i υποπίνακες με j στοιχεία ο καθένας. Προκειμένου να ψάξουμε ένα κλειδί πρώτα το συγκρίνουμε με σειριακό τρόπο με το τελευταίο κλειδί του κάθε υποπίνακα. Με τον τρόπο αυτό καταλαβαίνουμε σε ποιόν υποπίνακα πρέπει να συνεχισθεί η αναζήτηση. Σε περίπτωση επιτυχούς αναζήτησης, στη χειρότερη περίπτωση θα εκτελεσθούν $i + j$ συγκρίσεις, στη μέση περίπτωση θα εκτελεσθούν $\frac{i+1}{2} + \frac{j+1}{2}$ συγκρίσεις. Για ποιές τιμές των i, j η επίδοση βελτιστοποιείται;

8. Στη συνέχεια δίνεται η διαδικασία αναζήτησης `Fibonacci`. Υποτίθεται ότι αναζητείται το `key` στον ταξινομημένο πίνακα `A[0..n-1]`. Αν το κλειδί βρεθεί, τότε η μεταβλητή `position` επιστρέφει τη θέση του, αλλιώς επιστρέφει 0. Σημειώνεται, επίσης, ότι η εντολή `while` υπολογίζει τον αριθμό Fibonacci που είναι αμέσως μεγαλύτερος του `n`. Να αναλυθεί η πολυπλοκότητα του αλγορίθμου.

```
procedure Fibonacci(key,position)
j<--1;
while F(j)<n+1 do j<--j+1;
done <-- false;
mid <-- n-F(j-2)+1; f1 <-- F(j-2); f2 <-- F(j-3);
while (not done) and (key<>A[mid]) do
   if (mid<=0) or (key>A[mid]) then
       if f1=1 then
          done <-- true;
       else
          mid <-- mid+f2; f1 <-- f1-f2; f2 <-- f2-f1;
   else if f2=0 then done <-- true;
   else
       mid <-- mid-f2; temp <-- f1-f2; f1 <-- f2; f2 <-- temp;
if done then position <-- 0 else position <-- mid
```

9. Έστω ένας πύργος με n ορόφους, ενώ επίσης μας δίνεται μία λάμπα και πρέπει να βρούμε το χαμηλότερο όροφο από τον οποίο θα σπάσει, αν την αφήσουμε σε ελεύθερη πτώση. Η πρώτη λύση είναι να αρχίσουμε από τον πρώτο όροφο και να συνεχίσουμε προς το δεύτερο, τρίτο κλπ μέχρι να βρούμε την απάντηση. Προφανώς η λύση αυτή έχει γραμμική πολυπλοκότητα. Να σχεδιασθούν δύο αλγόριθμοι μη γραμμικής πολυπλοκότητας για την εύρεση της αντοχής της λάμπας, ενώ μας διατίθενται δύο λάμπες για την εκτέλεση του πειράματος. Να αποδειχθεί η πολυπλοκότητα του κάθε αλγορίθμου.

10. Συχνά παρουσιάζεται η ανάγκη αναζήτησης k κλειδιών σε ένα ταξινομημένο πίνακα n στοιχείων. Να σχεδιασθούν και να αναλυθούν τρεις αλγόριθμοι μαζικής αναζήτησης, όπου δηλαδή τα κλειδιά να αναζητούνται με μία κλήση του αλγορίθμου και όχι με διαδοχικές k κλήσεις. Οι τρεις αλγόριθμοι να στηρίζονται στη σειριακή, τη δυαδική και την αναζήτηση παρεμβολής.

11. Η **γραμμική αναζήτηση με διαχωρισμό της ακολουθίας** (split sequence linear search) λειτουργεί ως εξής. Έστω ότι κατά την εισαγωγή ενός κλειδιού `key` εφαρμόζεται η συνάρτηση κατακερματισμού `hash(key)` και προκύπτει μία θέση i του πίνακα, η οποία όμως είναι κατειλημμένη. Αν το κλειδί που είναι αποθηκευμένο στη θέση i είναι μεγαλύτερο (μικρότερο) από το αναζητούμενο κλειδί, τότε η επόμενη θέση όπου θα συνεχισθεί η σύγκριση δίνεται αντίστοιχα από τις σχέσεις:

$$(\texttt{hash(key)} + \texttt{c}_1) \mod \texttt{m} \quad \text{και} \quad (\texttt{hash(key)} + \texttt{c}_2) \mod \texttt{m}$$

όπου τα $\texttt{c}_1$ και $\texttt{c}_2$ είναι δύο άνισες ακέραιες σταθερές. Έτσι δημιουργούνται δύο ακολουθίες θέσεων, όπου θα κατευθυνθούν τα συνώνυμα και η πρωτεύουσα συγκέντρωση πράγματι μειώνεται. Να σχεδιασθεί η μέθοδος στις λεπτομέρειές της (δηλαδή, εισαγωγή, διαγραφή, αναζήτηση) και να αναλυθεί η χειρότερη περίπτωσή της.

12. Ο **ταξινομημένος κατακερματισμός** (ordered hashing) μπορεί να εφαρμοσθεί τόσο με τη γραμμική και τη μη γραμμική αναζήτηση, όσο και με το διπλό κατακερματισμό. Σύμφωνα με τη μέθοδο αυτή όταν προκύπτει σύγκρουση κατά την εισαγωγή, τότε τη συγκεκριμένη θέση καταλαμβάνει το κλειδί με τη μικρότερη τιμή εκτοπίζοντας το κλειδί με τη μεγαλύτερη τιμή, το οποίο πρέπει να επανα-εισαχθεί. Να σχεδιασθούν οι αλγόριθμοι εισαγωγής, διαγραφής και αναζήτησης. Να αναλυθεί η επιτυχής και η ανεπιτυχής αναζήτηση για την περίπτωση του ταξινομημένου διπλού κατακερματισμού.

13. Αν είναι γνωστή εκ των προτέρων η πιθανότητα αναζήτησης κάθε κλειδιού, τότε να σχεδιασθεί και να αναλυθεί η διαδικασία εισαγωγής, διαγραφής και αναζήτησης για μία παραλλαγή για τη μέθοδο κατακερματισμού με αλυσίδες, η οποία να εκμεταλλεύεται αυτό το γεγονός.

9

Αλγόριθμοι Ταξινόμησης

Περιεχόμενα Κεφαλαίου

Η **ταξινόμηση** (sorting) τοποθετεί ένα σύνολο κόμβων ή εγγραφών σε μία ιδιαίτερη σειρά (αύξουσα ή φθίνουσα) με βάση την τιμή του (πρωτεύοντος) κλειδιού της εγγραφής. Σκοπός της ταξινόμησης είναι στη συνέχεια η διευκόλυνση της αναζήτησης των στοιχείων του αντίστοιχου συνόλου. Για παράδειγμα, είναι γνωστό ότι η αναζήτηση ενός κλειδιού σε μη ταξινομημένο πίνακα με n εγγραφές γίνεται σειριακά με συγκρίσεις της τάξης $\mathrm{O}(n)$, ενώ σε ταξινομημένο πίνακα η δυαδική αναζήτηση απαιτεί κόστος της τάξης $\mathrm{O}(\log n)$. Η χρησιμότητα της ταξινόμησης αποδεικνύεται στην πράξη σε περιπτώσεις αναζήτησης αριθμητικών ή αλφαβητικών δεδομένων, όπως σε βιβλιοθηκονομικά συστήματα, λεξικά, τηλεφωνικούς καταλόγους, καταλόγους φόρου εισοδήματος και γενικά παντού όπου γίνεται αναζήτηση αποθηκευμένων αντικείμενων. Στη συνέχεια δίνεται ένας τυπικός ορισμός της ταξινόμησης.

Ορισμός.
Δοθέντων των στοιχείων $a_1, a_2, \ldots, , a_n$ η ταξινόμηση συνίσταται στη διάταξης (permutation) της θέσης των στοιχείων, ώστε να τοποθετηθούν σε μία σειρά $a_{k_1}, a_{k_2}, \ldots, a_{k_n}$ έτσι ώστε, δοθείσης μίας **συνάρτησης διάταξης** (ordering function), f, να ισχύει:

$$f(a_{k_1}) \leq f(a_{k_2}) \leq \ldots \leq f(a_{k_n})$$

□

Αξίζει να σημειωθεί ότι η προηγούμενη συνάρτηση διάταξης μπορεί να τροποποιηθεί, ώστε να καλύπτει και την περίπτωση όπου η ταξινόμηση γίνεται με φθίνουσα τάξη (descending sequence) μεγέθους του κλειδιού. Στη γενικότερη περίπτωση μπορεί να θεωρηθεί ότι η ταξινόμηση στηρίζεται σε δύο ή περισσότερα κλειδιά της εγγραφής. Για παράδειγμα, σε πολλά παιχνίδια της τράπουλας οι παίκτες ταξινομούν τα χαρτιά τους με πρώτο κλειδί το χρώμα του φύλλου (πχ. μπαστούνια, σπαθιά, καρά και κούπες) και με δεύτερο κλειδί την αξία του φύλλου (πχ. Άσσος, Ρήγας, Ντάμα, Βαλές, 10, ..., 2).

Όπως γνωρίζουμε από το μάθημα των Δομών Δεδομένων, το αντικείμενο της ταξινόμησης είναι πολύ πλούσιο καθώς μπορεί να εφαρμοσθεί σε πολλά περιβάλλοντα (όπως, **εσωτερική** (internal) ταξινόμηση στην κύρια μήνη ή **εξωτερική** (external) ταξινόμηση στη δευτερεύουσα μνήμη), και υπό πολλές συνθήκες (όπως, **επιτοπίως** (in situ, in place) ή με επιπλέον χώρο, ως προς τη στατιστική κατανομή των κλειδιών προς ταξινόμηση κλπ.). Στο κεφάλαιο αυτό θα εξετάσουμε εκ νέου τους γνωστούς αλγορίθμους εσωτερικής ταξινόμησης ώστε να αποδείξουμε και τυπικά τις (γνωστές εξ άλλου) εκφράσεις των πολυπλοκοτήτων τους. Αρχικά θα εξετάσουμε μεθόδους ταξινόμησης που βασίζονται στη σύγκριση, ενώ στη συνέχεια θα μελετήσουμε άλλες μεθόδους. Επίσης, θα απαντήσουμε στο ενδιαφέρον

ερώτημα: "Πόσο γρήγορα μπορεί να γίνει η ταξινόμηση", με σκοπό να αποδείξουμε το θεωρητικό όριο της πολυπλοκότητας των αλγορίθμων ταξινόμησης.

9.1 Ταξινόμηση με Εισαγωγή

Αρχικά θα εξετάσουμε τον αλγόριθμο της ταξινόμησης με εισαγωγή και θα καταλάβουμε τη συμπεριφορά του σε δύο περιπτώσεις: δίνοντας στην είσοδο τους πίνακες A1=[1,2,3,4,5,6] και A2=[6,5,4,3,2,1], με στοιχεία που είναι ήδη ταξινομημένα ή αντίστροφη σειρά ταξινομημένα. Παρατηρούμε ότι οι περιπτώσεις αυτές είναι ακραίες και δεν μπορούν να χαρακτηρισθούν ως η μέση περίπτωση, η περίπτωση δηλαδή όπου τα στοιχεία του πίνακα εμφανίζονται με μία τυχαία σειρά. Με την ίδια προσέγγιση θα ασχοληθούμε με την ταξινόμηση με επιλογή στο αμέσως επόμενο Κεφάλαιο 9.2. Ακολουθεί ο ψευδοκώδικας για τον αλγόριθμό μας.

```
     procedure insert
1.   for i <-- 2 to n do
2.       x <-- A[i];
3.       j <-- i-1;
4.       while j>0 and x<A[j] do
5.          A[j+1] <-- A[j]
6.          j <-- j-1
7.       A[j+1] <-- x
```

Η μέθοδος αυτή διακρίνει τον πίνακα σε δύο τμήματα: την ακολουθία πηγής (source sequence) και την ακολουθία προορισμού (destimation sequence). Δηλαδή, λαμβάνει στοιχεία από την ακολουθία πηγής και τα κατευθύνει στη σωστή θέση στην ακολουθία προορισμού. Έτσι κάθε φορά το μέγεθος της ακολουθίας πηγής μειώνεται κατά ένα, ενώ το μέγεθος της ακολουθίας προορισμού αυξάνεται κατά ένα. Το κάθε φορά λαμβανόμενο στοιχείο είναι πρώτο της ακολουθίας πηγής (δες την εντολή 3). Το στοιχείο αυτό συγκρίνεται με τα στοιχεία της ακολουθίας προορισμού αρχίζοντας από το τέλος και συνεχίζοντας προς την αρχή μέχρι να εντοπίσει την κατάλληλη θέση.

Πρόταση.
Η πολυπλοκότητα της ταξινόμησης με εισαγωγή για την καλύτερη, τη μέση και τη χειρότερη περίπτωση είναι $\Theta(n)$, $\Theta(n^2)$ και $\Theta(n^2)$, αντιστοίχως.

Απόδειξη
Αν στην είσοδο θεωρηθεί ο πίνακας A1, τότε η ταξινόμηση συμπεριφέρεται πολύ αποτελεσματικά και δεν εισέρχεται μέσα στο σώμα της εντολής 4. Έτσι ουσιαστικά το κάθε φορά λαμβανόμενο στοιχείο από την ακολουθία πηγής συγκρίνεται με ένα μόνο στοιχείο (όπως προκύπτει από τη συνθήκη ελέγχου 4). Θεωρώντας αυτή τη σύγκριση ως την πράξη βαρόμετρο, καταλήγουμε ότι για ένα πίνακα μεγέθους n στοιχείων, ο αντίστοιχος αριθμός συγκρίσεων είναι $n-1$. Επομένως προκύπτει ότι στην καλύτερη περίπτωση η πολυπλοκότητα είναι γραμμική $\Theta(n)$.

Τώρα ας θεωρήσουμε την περίπτωση όπου στην είσοδο δίνεται ο πίνακας A2. Το κάθε φορά επιλεγόμενο στοιχείο είναι μικρότερο από τα ήδη τοποθετημένα στην ακολουθία προορισμού. Έτσι εκτελείται ένας συγκεκριμένος αριθμός συγκρίσεων μέχρι να τοποθετηθεί το νέο στοιχείο στη σωστή θέση, δηλαδή στην πρώτη θέση της ακολουθίας προορισμού. Έτσι η εντολή 5 εκτελείται i φορές, όπου το i μεταβάλλεται από 1 μέχρι $n-1$. Συνεπώς και πάλι θεωρώντας την πράξη αυτή ως βαρόμετρο, καταλήγουμε ότι για ένα πίνακα μεγέθους n στοιχείων, ο αντίστοιχος αριθμός συγκρίσεων είναι $\sum_{i=1}^{n-1} i = n(n-1)/2$. Επομένως η πολυπλοκότητα στη χειρότερη περίπτωση είναι τετραγωνική $\Theta(n^2)$.

Το σημαντικότερο ερώτημα προκύπτει στην περίπτωση ταξινόμησης τυχαίων δεδομένων εισόδου, δηλαδή στην περίπτωση όπου οποιαδήποτε από τις $n!$ διατάξεις των δεδομένων εισόδου μπορεί να εμφανισθεί με ίση πιθανότητα $1/n!$. Η επίδοση αυτή μπορεί να προκύψει θεωρώντας ότι το στοιχείο που κάθε φορά λαμβάνεται από την ακολουθία πηγής θα διανύσει το μισό δρόμο μέχρι την αρχή της ακολουθίας προορισμού. Όπως προηγουμένως, και πάλι θα προκύψει ότι στη μέση αυτή περίπτωση η επίδοση περιγράφεται από μία τετραγωνική συνάρτηση, επομένως και στην περίπτωση αυτή η πολυπλοκότητα είναι $\Theta(n^2)$. □

9.2 Ταξινόμηση με Επιλογή

Από το Κεφάλαιο 1.5 αντιγράφουμε τον ψευδοκώδικα της ταξινόμησης με επιλογή.

```
     procedure select
1.   for i <-- 1 to n-1 do
2.      minj <-- i; minx <-- A[i];
3.      for j <-- i+1 to n do
4.         if A[j]<minx then
5.            minj <-- j
6.            minx <-- A[j]
7.      A[minj] <-- A[i]
8.      A[i] <-- minx
```

Πρόταση.
Η πολυπλοκότητα της ταξινόμησης με επιλογή είναι $\Theta(n^2)$ σε κάθε περίπτωση.

Απόδειξη
Όπως φαίνεται στον ψευδοκώδικα, ουσιαστικά ο αλγόριθμος της ταξινόμησης με επιλογή σαρώνει τον πίνακα ώστε να εντοπίσει το μικρότερο στοιχείο και να το τοποθετήσει στην πρώτη θέση. Στη συνέχεια περιορίζει την αναζήτηση στα υπόλοιπα $n-1$ στοιχεία ώστε να εντοπίσει το μικρότερο μεταξύ αυτών και να το τοποθετήσει στη δεύτερη θέση κοκ. Επομένως, εύκολα προκύπτει ότι η εντολή 4 (δηλαδή, `if A[j]<minx`) θα εκτελεσθεί τον ίδιο αριθμό επαναλήψεων ανεξαρτήτως του περιεχομένου του πίνακα εισόδου. Είτε, λοιπόν, στην είσοδο δοθεί ο πίνακας A1 είτε ο A2, το αντίστοιχο πλήθος συγκρίσεων θα είναι: $\sum_{i=1}^{n-1} i = n(n-1)/2$. Επομένως σε κάθε περίπτωση η πολυπλοκότητα είναι τετραγωνική $\Theta(n^2)$. □

Από το παράδειγμα αυτό εξάγεται το συμπέρασμα ότι η ταξινόμηση με επιλογή έχει την ίδια επίδοση (δηλαδή, τετραγωνική πολυπλοκότητα) ανεξαρτήτως των δεδομένων εισόδου. Έτσι επιβεβαιώνουμε κάτι που αναφέραμε στο Κεφάλαιο 2, δηλαδή ότι ο αλγόριθμος αυτός είναι **σταθερός** (robust), επειδή έχει ταυτόσημες καλύτερη, χειρότερη και μέση περίπτωση. Αντίθετα, δεν συμβαίνει το ίδιο με τον αλγόριθμο ταξινόμησης με εισαγωγή.

Καθώς η εξέταση του πλήθους των συγκρίσεων ήταν εύκολη υπόθεση, στη συνέχεια θα εξετάσουμε τον αλγόριθμο αυτό ώστε να προσδιορίσουμε το πλήθος των καταχωρήσεων. Καθώς υπάρχουν αρκετές καταχωρήσεις στον αλγόριθμο, επιλέγουμε ως σημαντικότερη την καταχώρηση `minx <-- A[j]` της εντολής 6 εντός του εσωτερικού βρόχου παρά τις καταχωρήσεις που εκτελούνται εκτός του εσωτερικού αλλά εντός του εξωτερικού βρόχου. Εξ άλλου είναι εύκολο να διαπιστωθεί ότι οι καταχωρήσεις στις εντολές `minj <-- i; minx <-- A[i]; A[minj] <-- A[i]`, και `A[i] <-- minx` (εντολές 2, 7, 8) θα εκτελεσθούν συνολικά $4(n-1)$ φορές.

Πρόταση.
Η μέση τιμή του πλήθους των καταχωρήσεων κατά την ταξινόμηση με επιλογή είναι $\Theta(n \log n)$.

Απόδειξη
Ας υποθέσουμε ότι τα στοιχεία του πίνακα είναι διακριτά και ότι κάθε μία από τις $n!$ διατάξεις είναι ισοπίθανο να συμβεί. Στην i-οστή επανάληψη ο αλγόριθμος

αναζητά το μικρότερο μεταξύ των $n - i + 1$ στοιχείων του πίνακα στα δεξιά της θέσης i (συμπεριλαμβανομένης της θέσης αυτής). Έστω ότι με $R(m)$ συμβολίζουμε το πλήθος των καταχωρήσεων που απαιτούνται για την επιλογή του ελάχιστου μεταξύ των m στοιχείων, οπότε για τη μέση τιμή του $R(m)$ ισχύει:

$$R(m) \;=\; S(m)/m!$$

όπου $S(m)$ είναι το πλήθος των καταχωρήσεων που αθροίζονται κατά την εκτέλεση των $m!$ διαφορετικών διατάξεων των m στοιχείων.

Ας θεωρήσουμε τις $(m-1)!$ περιπτώσεις όπου το ελάχιστο στοιχείο εντοπίζεται στην τελευταία θέση του πίνακα. Στην περίπτωση αυτή το πλήθος των καταχωρήσεων είναι

$$S(m-1) + (m-1)!$$

καθώς ο αλγόριθμος συμπεριφέρεται σαν να υπήρχαν κατ'αρχήν $(m-1)$ μη επαναλαμβανόμενα στοιχεία, οπότε στη συνέχεια θα εκτελούνταν άλλη μία καταχώρηση για κάθε μία από τις $(m-1)!$ διαφορετικές διατάξεις. Αν εξαιρεθεί η τελευταία περίπτωση (δηλαδή, το ελάχιστο βρίσκεται στην τελευταία θέση), τότε απομένουν προς εξέταση $(m-1)(m-1)!$ περιπτώσεις. Στις περιπτώσεις αυτές το ελάχιστο στοιχείο βρίσκεται από την πρώτη μέχρι την $(m-1)$-οστή θέση, οπότε για κάθε ομάδα $(m-1)!$ διατάξεων απαιτούνται $S(m-1)$ καταχωρήσεις. Αυτό συνολικά δίνει άλλες

$$(m-1)S(m-1)$$

καταχωρήσεις. Έτσι προκύπτει η αναδρομική εξίσωση:

$$S(m) \;=\; mS(m-1) + (m-1)!$$

με αρχική συνθήκη $S(1) = 0$. Άρα, για τη μέση τιμή του $R(m)$ ισχύει:

$$R(m) \;=\; R(m-1) + 1/m \qquad\qquad m > 1$$

που επιλύεται εύκολα και δίνει τη λύση:

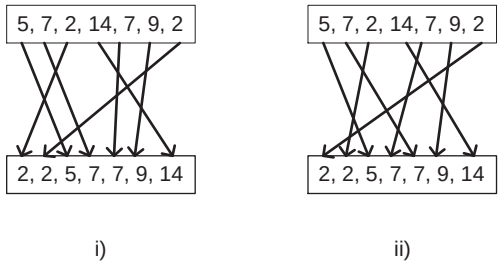
$$R(m) \;=\; \sum_{k=2}^{m} \frac{1}{k} \;=\; H_m - 1$$

Λαμβάνοντας υπόψη το σύνολο των $n-1$ επαναλήψεων του εξωτερικού βρόχου (εντολή 1) προκύπτει ότι η μέση τιμή του πλήθους των καταχωρήσεων είναι:

$$\begin{aligned} T(n) &= \sum_{i=1}^{n-1}(H_{n-i+1}-1) = \sum_{i=2}^{n}(H_i-1) \\ &= \left(\frac{1}{2}\right)+\left(\frac{1}{2}+\frac{1}{3}\right)+\ldots+\left(\frac{1}{2}+\frac{1}{2}+\ldots+\frac{1}{n}\right) \\ &= \sum_{k=2}^{n}(n-k+1)\frac{1}{k} = \ldots = (n+1)H_n-2n \end{aligned}$$

Άρα προκύπτει ότι στη μέση περίπτωση το πλήθος των καταχωρήσεων είναι της τάξης $\Theta(n\log n)$. Ακόμη και αν προσθέταμε τις $4(n-1)$ καταχωρήσεις που εκτελούνται εκτός του εσωτερικού βρόχου, και πάλι το αποτέλεσμα δεν θα άλλαζε. □

Ένα τελικό σχόλιο. Μία μέθοδος ταξινόμησης λέγεται **ευσταθής** (stable), αν διατηρεί τη σχετική θέση πριν και μετά τη διάταξη για τα στοιχεία με την ίδια τιμή. Σε αντίθετη περίπτωση λέγεται **ασταθής**. Στο Σχήμα 9.1 παρουσιάζεται μία περίπτωση ευσταθούς και μία περίπτωση ασταθούς αλγορίθμου. Η ταξινόμηση με εισαγωγή είναι ευσταθής αν και η μέθοδος αυτή έχει κακή σταθερή πολυπλοκότητα $\Theta(n^2)$. Για την περίπτωση της γρήγορης ταξινόμησης που θα μελετήσουμε στη συνέχεια, θα παρατηρήσουμε ότι η τελευταία αυτή μέθοδος δεν είναι ευσταθής καθώς εκτελεί περιττές μετακινήσεις ίσων κλειδιών. Παρ'όλα αυτά δεν παύει να θεωρείται μία πολύ αποτελεσματική μέθοδος.



Σχήμα 9.1: (i) Ευσταθής και (ii) Ασταθής Ταξινόμηση

9.3 Γρήγορη Ταξινόμηση

Η **γρήγορη ταξινόμηση** (quicksort), που αλλιώς ονομάζεται και ταξινόμηση με **διαμερισμό και ανταλλαγή** (partition exchange sort), θεωρείται ένας από τους top-10 αλγορίθμους ως προς την πρακτική χρησιμότητά και το ενδιαφέρον που προκάλεσαν από τη θεωρητική σκοπιά. Η γρήγορη ταξινόμηση αποτελεί ένα κλασικό παράδειγμα αλγορίθμου της οικογενείας Διαίρει και Βασίλευε. Όπως δηλώνει το πρώτο όνομα του αλγορίθμου, πρόκειται για μία πολύ αποτελεσματική μέθοδο ταξινόμησης (πλην ελαχίστων εξαιρέσεων). Όπως δηλώνει το δεύτερο όνομά του, τα βασικά χαρακτηριστικά του είναι δύο: η ανταλλαγή και ο διαμερισμός. Πρώτον, δηλαδή, εκτελούνται ανταλλαγές μεταξύ των στοιχείων του πίνακα έτσι ώστε τα στοιχεία με μικρότερες τιμές να μετακινηθούν προς τη μία πλευρά, ενώ τα στοιχεία με μεγαλύτερες τιμές να μετακινηθούν προς την άλλη πλευρά του πίνακα. Έτσι, επακολουθεί η εφαρμογή του δεύτερου χαρακτηριστικού, δηλαδή του διαμερισμού του πίνακα σε δύο μικρότερους που ταξινομούνται ανεξάρτητα μεταξύ τους. Είναι προφανές ότι στους δύο υποπίνακες επιφυλάσσεται η ίδια αντιμετώπιση: ανταλλαγή και διαμερισμός.

```
     procedure quicksort(left,right);
1.   if left<right then
2.      i <-- left; j <-- right+1; pivot <-- A[left];
3.      repeat
4.         repeat i <-- i+1 until A[i]>=pivot;
5.         repeat j <-- j-1 until A[j]<=pivot;
6.         if i<j then swap(A[i],A[j]);
7.      until j<=i;
8.      swap(A[left],A[j]);
9.      quicksort(left,j-1);
10.     quicksort(j+1,right)
```

Η προηγούμενη διαδικασία `quicksort` υποθέτει ότι δίνεται προς ταξινόμηση ένας πίνακας `A[1..n]`, που περιέχει ακεραίους αριθμούς. Ο `pivot` είναι ένα στοιχείο του πίνακα, το οποίο παίρνει την τελική του θέση στον πίνακα, ώστε κατόπιν να γίνει ο διαμερισμός (δηλαδή, να γίνουν οι κλήσεις της `partition`) σε δύο υποπίνακες με μικρότερα και μεγαλύτερα στοιχεία από τον `pivot` αντίστοιχα. Η επιλογή του `pivot` μπορεί να γίνει κατά πολλούς τρόπους. Στο σημείο αυτό δεχόμαστε ότι ως `pivot` λαμβάνεται το πρώτο στοιχείο του πίνακα. Έτσι, ο πίνακας σαρώνεται από τα αριστερά προς τα δεξιά αναζητώντας το πρώτο στοιχείο με τιμή μεγαλύτερη ή ίση με την τιμή του `pivot`. Ύστερα, ο πίνακας

σαρώνεται από τα δεξιά προς τα αριστερά αναζητώντας το πρώτο στοιχείο με τιμή μικρότερη ή ίση με την τιμή του `pivot`. Όταν εντοπισθούν δύο τέτοια στοιχεία, τότε αυτά ανταλλάσσονται ώστε κατά το δυνατόν να τείνουν να πλησιάσουν προς την τελική τους θέση. Κατόπιν συνεχίζουμε τις σαρώσεις προσπαθώντας να εντοπίσουμε άλλα παρόμοια ζεύγη και να τα ανταλλάξουμε. Κάποια στιγμή οι δύο δείκτες σάρωσης διασταυρώνονται. Τότε η σάρωση σταματά και ο `pivot` λαμβάνει την τελική του θέση εκτελώντας άλλη μία ανταλλαγή μεταξύ του `pivot` και του στοιχείου όπου σταμάτησε ο δείκτης της σάρωσης από δεξιά. Έτσι, η διαδικασία μπορεί να συνεχίσει στους δύο υποπίνακες κατά τα γνωστά. Το επόμενο σχήμα δείχνει τα βήματα του κώδικα μέχρι τον πρώτο διαμερισμό.

Αρχικά κλειδιά	52	12	71 $i\uparrow$	56	5	10	19	90	45 $j\uparrow$
1η ανταλλαγή	52	12	45	56 $i\uparrow$	5	10	19 $j\uparrow$	90	71
2η ανταλλαγή Διασταύρωση δεικτών	52	12	45	19	5	10 $j\uparrow$	56 $i\uparrow$	90	71
Ανταλλαγή και διαμερισμός	[10	12	45	19	5]	52	[56	90	71]

Σχήμα 9.2: Διαδικασία διαμερισμού.

Τώρα ας προχωρήσουμε στην εύρεση της πολυπλοκότητας του αλγορίθμου. Κατ'αρχήν θα εξετάσουμε τη χειρότερη περίπτωση που είναι ευκολότερη, και στη συνέχεια την καλύτερη και τη μέση περιπτωση.

Πρόταση.
Η πολυπλοκότητα της γρήγορης ταξινόμησης είναι $\Theta(n^2)$ στη χειρότερη περίπτωση.

Απόδειξη
Η περίπτωση αυτή συμβαίνει όταν σε πίνακα μεγέθους n, το επιλεγόμενο στοιχείο ως `pivot` είναι το μικρότερο ή το μεγαλύτερο, ώστε τα μεγέθη των δύο υποπινάκων που θα προκύψουν να είναι 0 και $n-1$ (αφού η μία θέση του συγκεκριμένου πίνακα θα καταληφθεί από τον ίδιο τον `pivot`). Έτσι εύκολα προκύπτει η ακόλουθη αναδρομική εξίσωση:

$$T(n) \;=\; T(n-1) + n$$

Ο όρος n του δεξιού σκέλους αντιστοιχεί στις συγκρίσεις που θα εκτελεσθούν κατά τη σάρωση του πίνακα (εντολές 4-5)[1]. Η ανωτέρω αναδρομική εξίσωση είναι εύκολη υπόθεση καθώς διαδοχικά ισχύει:

$$\begin{aligned} T(n-1) &= T(n-2) + (n-1) \\ T(n-2) &= T(n-3) + (n-2) \\ &\vdots \\ T(2) &= T(1) + 2 \end{aligned}$$

Αθροίζοντας τα αριστερά και δεξιά σκέλη αυτών των εξισώσεων και θεωρώντας ότι $T(0) = T(1) = 0$ προκύπτει ότι:

$$T(n) = T(1) + \sum_{i=2}^{n} i = n(n+1)/2 - 1 = \Theta(n^2)$$

□

Πρόταση.
Η πολυπλοκότητα της γρήγορης ταξινόμησης είναι $\Theta(n \log n)$ στην καλύτερη περίπτωση.

Απόδειξη
Για να αναλύσουμε την καλύτερη περίπτωση αρκεί να διαπιστώσουμε ότι αυτή συμβαίνει όταν κάθε φορά ως `pivot` επιλέγεται ένα στοιχείο που λαμβάνοντας την τελική του θέση υποδιαιρεί τον πίνακα σε δύο υποπίνακες ίσου μεγέθους. Επομένως ισχύει η αναδρομική εξίσωση:

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n$$

με αρχικές συνθήκες $T(0) = T(1) = 0$. Για την ευκολία της ανάλυσης υποθέτουμε ότι $n = 2^k$, οπότε:

$$T(n) = 2T(n/2) + n \quad \Rightarrow \quad \frac{T(n)}{n} = \frac{T(n/2)}{n/2} + 1$$

Η τελευταία αναδρομική εξίσωση είναι πλέον εύκολη υπόθεση καθώς ισχύει διαδοχικά:

$$\frac{T(n/2)}{n/2} = \frac{T(n/4)}{n/4} + 1$$

[1] Στην πραγματικότητα μπορεί να εκτελεσθούν περισσότερες από n συγκρίσεις, αλλά σε κάθε περίπτωση ο αριθμός των συγκρίσεων είναι μικρότερος από $2n$, γεγονός που δεν αλλάζει το τελικό συμπέρασμα.

$$\vdots$$

$$\frac{T(2)}{2} \;=\; \frac{T(1)}{1} + 1$$

Αθροίζοντας τα αριστερά και δεξιά σκέλη αυτών των εξισώσεων προκύπτει ότι:

$$\frac{T(n)}{n} \;=\; \frac{T(1)}{1} + \log n \;\Rightarrow\; T(n) \;=\; n + n\log n \;=\; \Theta(n\log n)$$

□

Πρόταση.
Η πολυπλοκότητα της γρήγορης ταξινόμησης είναι $\Theta(n \log n)$ στη μέση περίπτωση.

Απόδειξη
Για να εξετάσουμε τη μέση περίπτωση θα υποθέσουμε ότι για n στοιχεία, κάθε διάταξη των στοιχείων αυτών (όπου $n!$ είναι το πλήθος των διατάξεων) είναι ισοπίθανη. Επομένως, ο κάθε φορά επιλεγόμενος `pivot` τελικά επιλέγεται με τυχαίο τρόπο. Θα χειρισθούμε τη μέση περίπτωση με τη βοήθεια της επόμενης αναδρομικής εξίσωσης για $n \geq 2$:

$$T(n) \;=\; n + \frac{1}{n}\sum_{k=0}^{n-1}(T(k) + T(n-k-1))$$

με αρχικές συνθήκες $T(0) = T(1) = 0$. Λόγω συμμετρίας και των αρχικών συνθηκών, η προηγούμενη αναδρομική εξίσωση μετασχηματίζεται σε:

$$T(n) \;=\; n + \frac{2}{n}\sum_{k=2}^{n-1} T(k)$$

Τώρα, τη σχέση αυτή πολλαπλασιάζουμε επί n και προκύπτει:

$$n\,T(n) \;=\; n^2 + 2\sum_{k=2}^{n-1} T(k)$$

Επίσης, στην ίδια σχέση αντικαθιστούμε το n με $n-1$ και πολλαπλασιάζουμε με $n-1$, οπότε προκύπτει

$$(n-1)\,T(n-1) \;=\; (n-1)^2 + 2\sum_{k=2}^{n-2} T(k)$$

Αφαιρώντας τα αντίστοιχα σκέλη των δύο τελευταίων εκφράσεων και εκτελώντας απλή άλγεβρα, προκύπτει:

$$n\,T(n) - (n-1)\,T(n-1) \;=\; 2n-1+2T(n-1) \;\;\Rightarrow$$
$$\frac{T(n)}{n+1} \;=\; \frac{T(n-1)}{n} + \frac{2n-1}{n(n+1)}$$

Με διαδοχικές αντικαταστάσεις του n με $n-1$ προκύπτει:

$$\begin{aligned}
\frac{T(n-1)}{n} &= \frac{T(n-2)}{n-1} + \frac{2n-3}{(n-1)n} \\
\frac{T(n-2)}{n-1} &= \frac{T(n-3)}{n-2} + \frac{2n-5}{(n-2)(n-1)} \\
&\vdots \\
\frac{T(2)}{3} &= \frac{T(1)}{2} + \frac{3}{2\times 3}
\end{aligned}$$

Αθροίζοντας αντίστοιχα τα αριστερά και τα δεξιά σκέλη και απλοποιώντας προκύπτει:

$$\frac{T(n)}{n+1} \;=\; \frac{2n-1}{n(n+1)} + \frac{2n-3}{(n-1)n} + \ldots + \frac{3}{2\times 3} \;=\; \ldots \;=\; \sum_{i=2}^{n}\left(\frac{3}{i+1} - \frac{1}{i}\right)$$

Επομένως πρέπει να υπολογίσουμε το άθροισμα του δεξιού σκέλους:

$$\sum_{i=2}^{n}\left(\frac{3}{i+1} - \frac{1}{i}\right) \;=\; 3\left(H_{n+1} - 1 - \frac{1}{2}\right) - (H_n - 1) \;=\; \ldots \;=\; 2H_n - \frac{7}{2} + \frac{2}{n+1}$$

Τελικώς έχουμε:

$$\frac{T(n)}{n+1} \;=\; 2H_n - \frac{7}{2} + \frac{2}{n+1} \;\;\Rightarrow$$
$$T(n) \;=\; 2(n+1)H_n - \frac{7}{2}(n+1) + 2 = \Theta(n\log n)$$

□

Η γρήγορη ταξινόμηση δεν είναι επιτόπιος αλγόριθμος, δηλαδή δεν αρκείται στο χώρο του συγκεκριμένου πίνακα αλλά χρειάζεται επιπλέον χώρο. Ο χώρος αυτός δεν φαίνεται καθαρά στον ανωτέρω κώδικα αλλά απαιτείται από το μεταγλωττιστή κατά την αναδρομή. Εύλογα προκύπτει η ερώτηση σχετικά με το μέγεθος αυτού του απαιτούμενου χώρου. Στη χειρότερη περίπτωση το μέγιστο βάθος της αναδρομής θα είναι $n-1$ κλήσεις, αριθμός υπερβολικός. Για το λόγο αυτό έχει σχεδιασθεί μία επαναληπτική εκδοχή της γρήγορης ταξινόμησης, δηλαδή μία εκδοχή που υλοποιεί την απαραίτητη στοίβα, όπου τοποθετούνται τα όρια του μικρότερου υποπίνακα, όπως φαίνεται στην επόμενη διαδικασία `iterative_quicksort`.

```
     procedure iterative_quicksort(left,right);
1.   repeat
2.      while (left<right) do
3.         i <-- left; j <-- r+1; pivot <-- A[left];
4.         repeat
5.            repeat i <-- i+1 until A[i]>=pivot;
6.            repeat j <-- j-1 until A[j]<=pivot;
7.            if i<j then swap(A[i],A[j]);
8.         until j<=i;
9.         swap(A[left],A[j]);
10.        if ((j-p)<(q-j)) then
11.           push(j+1); push(q); q <-- j-1
12.        else
13.           push(p); push(j-1); p <-- j+1
14.     if stack is empry then return
15.     pop(q); pop(p)
16.  until (false)
```

Πρόταση.
Η χωρική πολυπλοκότητα της γρήγορης ταξινόμησης είναι $\Theta(n \log n)$, στη μέση περίπτωση.

Απόδειξη
Ο απαιτούμενος χώρος εκφράζεται από την αναδρομική εξίσωση:

$$S(n) \leq \begin{cases} 2 + S(\lfloor n-1 \rfloor/2) & n > 1 \\ 0 & n \leq 1 \end{cases}$$

Επομένως υποθέτοντας ότι $n = 2^k$, αρκεί να επιλύσουμε την εξίσωση:

$$\begin{aligned} S(n) &= S(n/2) + 2 = (S(n/4) + 2) + 2 = S(n/4) + 4 = \ldots = \\ &= S(1) + 2k = 2\log n = \Theta(\log n) \end{aligned}$$

□

9.4 Στατιστικά Διάταξης

Με τον όρο **στατιστικά διάταξης** (order statistics) εννοούμε την περίπτωση όπου, δεδομένου ενός πίνακα A με n αταξινόμητα στοιχεία, πρέπει να αναζητήσουμε

το μικρότερο, το μεγαλύτερο, το μεσαίο ή γενικά το k-οστό στοιχείο του πίνακα με βάση την τιμή του κλειδιού του. Παρά το γεγονός ότι το αντικείμενο είναι πρόβλημα αναζήτησης και όχι ταξινόμησης, θα το εξετάσουμε στο σημείο αυτό για λόγους που θα φανούν στη συνέχεια.

Ας αρχίσουμε από τα εύκολα, δηλαδή έστω ότι θέλουμε να βρούμε ταυτόχρονα το μικρότερο και το μεγαλύτερο στοιχείο ενός πίνακα. Η προφανής λύση είναι να εκτελέσουμε δύο σαρώσεις, μία σάρωση για την εύρεση του μικρότερου και στη συνέχεια μία σάρωση για την εύρεση του μεγαλύτερου. Είναι ευνόητο ότι η πολυπλοκότητα αυτής της πρώτης προσέγγισης είναι $\Theta(n)$. Με τη διαδικασία που ακολουθεί προσπαθούμε με μία μόνο σάρωση να επιτύχουμε την ταυτόχρονη εύρεση των δύο στοιχείων που μας ενδιαφέρουν.

```
     procedure maxmin1
1.   max <-- A[1]; min <-- A[1];
2.   for i <-- 2 to n do
3.      if A[i]>max then max <-- A[i]
4.      if A[i]<min then min <-- A[i]
5.   return max, min
```

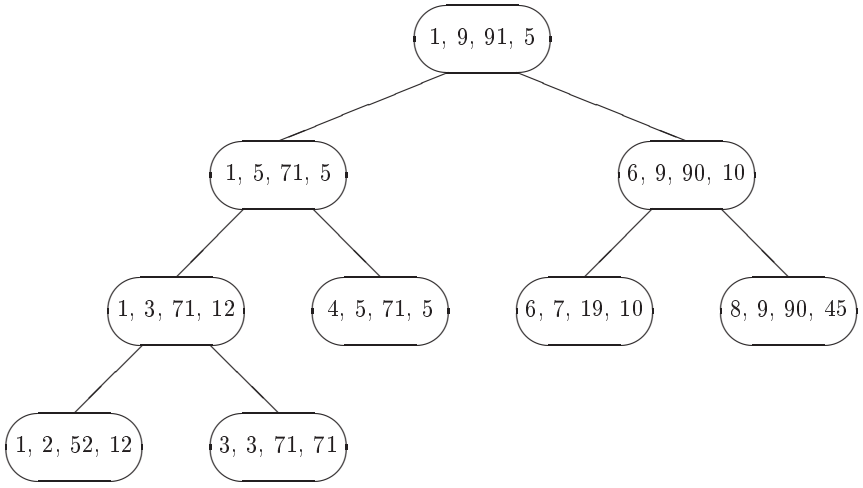
Η διαδικασία αυτή δεν παύει να είναι μία απλοϊκή προσέγγιση. Αν και γίνεται μία σάρωση του πίνακα, κάθε στοιχείο υποβάλλεται σε δύο συγκρίσεις, που όπως είπαμε είναι ένα σύνηθες βαρόμετρο. Η κατάσταση θα μπορούσε να βελτιωθεί αν αντικαθιστούσαμε τις εντολές 3-4 με τις εντολές:

```
3.      if A[i]>max then max <-- A[i]
4.      else if A[i]<min then min <-- A[i]
```

Ωστόσο, και αυτή η εκδοχή μπορεί να αποδειχθεί ότι δεν βοηθά στη χειρότερη περίπτωση. Η επόμενη διαδικασία, που προσπαθεί να εκμεταλλευθεί την τεχνική του Διαίρει και Βασίλευε, υποθέτει ότι με τα ορίσματα `i,j` δίνουμε τα όρια του πίνακα και με τα ορίσματα `fmax,fmin` μας επιστρέφονται οι τιμές που μας ενδιαφέρουν. Επίσης υποθέτουμε ότι η συνάρτηση `max` επιστρέφει το μέγιστο μεταξύ δύο στοιχείων, ενώ η συνάρτηση `min` επιστρέφει το ελάχιστο μεταξύ των δύο στοιχείων.

```
     procedure maxmin2(i,j,fmax,fmin)
1.   if i=j then
2.      max <-- A[i]; min <-- A[i];
3.   else if i=j-1 then
4.      if A[i]<A[j] then
```

```
5.          fmax <-- A[j]; fmin <-- A[i];
6.       else
7.          fmax <-- A[i]; fmin <-- A[j];
8.    else
9.       middle <-- (i+j)/2;
10.      maxmin2(i,mid,gmax,gmin);
11.      maxmin2(mid+1,j,hmax,hmin);
11.      fmax <-- max(gmax,hmax);
12.      fmin <-- min(gmin,hmin);
```



Σχήμα 9.3: Κλήσεις για τον υπολογισμό του ελάχιστου και του μέγιστου.

Έστω ότι το περιεχόμενο του πίνακα A είναι τα 9 στοιχεία 52, 12, 71, 56, 5, 10, 19, 90, 45. Στο επόμενο σχήμα απεικονίζεται ένα δένδρο με κόμβους που περιέχουν τα ορίσματα των κλήσεων. Διασχίζοντας το δένδρο με προτεραιότητα κατά βάθος (dfs) μπορούμε να αποτυπώσουμε την επακριβή σειρά των διαφόρων κλήσεων.

Πρόταση.
Η πολυπλοκότητα της maxmin2 είναι $\Theta(n)$ στη χειρότερη περίπτωση.

Απόδειξη
Η πολυπλοκότητα της διαδικασίας αυτής μπορεί να ευρεθεί επιλύοντας την αναδρομική

εξίσωση:

$$T(n) = \begin{cases} 0 & \text{αν } n = 1 \\ 1 & \text{αν } n = 2 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 2 & \text{αν } n > 2 \end{cases}$$

όπου οι σταθεροί όροι εκφράζουν το κόστος των συγκρίσεων στις εντολές 4 και 11-12. Θεωρώντας τη χειρότερη περίπτωση και ότι $n = 2^k$ διαδοχικά έχουμε:

$$\begin{aligned} T(n) &= 2T(n/2) + 2 \\ &= 2(2T(n/4) + 2) + 2 \;=\; 4T(n/4) + 4 + 2 \\ &\vdots \\ &= 2^k + 2^{k-1} + \ldots + 2^1 \\ &= 2n - 2 \end{aligned}$$

Επομένως, παρότι ο τελευταίος αλγόριθμος είναι βελτιωμένος δεν μπορεί να υπερβεί το όριο της γραμμικής πολυπλοκότητας $\Theta(n)$. □

9.4.1 Στατιστικά σε Μέσο Γραμμικό Χρόνο

Έστω ότι επιθυμούμε να βρούμε το k-οστό στοιχείο ενός αταξινόμητου πίνακα. Αυτό μπορεί να επιτευχθεί ταξινομώντας τον πίνακα και λαμβάνοντας το περιεχόμενο της αντίστοιχης θέσης του πίνακα. Γνωρίζουμε ήδη από το αντικείμενο των Δομών Δεδομένων ότι το κάτω όριο (που θα αποδειχθεί στη συνέχεια) των αλγορίθμων ταξινόμησης που στηρίζονται σε συγκρίσεις είναι $\mathrm{O}(n \log n)$. Επομένως το όριο αυτό προσδιορίζει και την πολυπλοκότητα της μεθόδου που ανάγει το πρόβλημα της επιλογής σε πρόβλημα ταξινόμησης. Στη συνέχεια θα επιλύσουμε το πρόβλημα της επιλογής του k-οστού στοιχείου με μία μέθοδο που στηρίζεται στη γρήγορη ταξινόμηση, όπως φαίνεται από τον επόμενο αλγόριθμο. Ο αλγόριθμος αυτός σχεδιάστηκε από τον C.A.R. Hoare [67]. Σημειώνεται ότι πρέπει να ισχύει: $left \leq k \leq right$.

```
     procedure find(left,right,k);
1.   if left=right then return A[left]
2.   else
3.      i <-- left; j <-- right+1; pivot <-- A[left];
4.      repeat
5.         repeat i <-- i+1 until A[i]>=pivot;
6.         repeat j <-- j-1 until A[j]<=pivot;
```

```
7.          if i<j then swap(A[i],A[j]);
8.       until j<=i;
9.       swap(A[left],A[j]);
10.      if k=j then return A[j]
11.      else if (k<j) then find(left,j-1,k)
12.      else find(j+1,right,k-j);
```

Παρατηρούμε ότι η διαδικασία `find` είναι σχεδόν ταυτόσημη με τη διαδικασία `quicksort`. Πιο συγκεκριμένα, οι εντολές 3-9 ταυτίζονται με αντίστοιχες εντολές της `quicksort`, καθώς αφορούν στις διαδικασίες επιλογής του `pivot` και του διαμερισμού. Στη συνέχεια διαφοροποιούνται οι εντολές 10-12. Η ουσία είναι ότι καθώς ο `pivot` τοποθετείται οριστικά στη θέση `j` του πίνακα, αποφασίζουμε αν θα τερματίσουμε ή αν θα συνεχίσουμε την αναζήτηση του k-οστού στοιχείου στον αριστερό ή στο δεξιό υποπίνακα με αναδρομικό τρόπο. Η ανάλυση του αλγορίθμου είναι πλέον αρκετά προφανής με βάση όλα τα προηγούμενα.

Πρόταση.
Η πολυπλοκότητα της `find` είναι $\Theta(n^2)$, $\Theta(n)$ και $\Theta(n)$, στη χειρότερη, τη μέση και την καλύτερη περίπτωση, αντιστοίχως.

Απόδειξη
Κατ'αρχήν η χειρότερη περίπτωση είναι $\Theta(n^2)$ και αυτό θα συμβεί όταν συνεχώς ο επιλεγόμενος `pivot` είναι το μικρότερο ή το μεγαλύτερο στοιχείο. Θα μελετήσουμε τη μέση περίπτωση για την οποία ισχύει:

$$T(n) \;=\; n + \sum_{k=0}^{n-1} \frac{1}{n}\, T(k)$$

με αρχικές συνθήκες $T(0) = T(1) = 0$. Επομένως:

$$T(n) \;=\; n + \frac{1}{n} \sum_{k=2}^{n-1} T(k)$$

Τώρα, τη σχέση αυτή πολλαπλασιάζουμε επί n και προκύπτει:

$$n\, T(n) \;=\; n^2 + \sum_{k=2}^{n-1} T(k)$$

Επίσης, στην ίδια σχέση αντικαθιστούμε το n με $n-1$ και πολλαπλασιάζουμε με $n-1$, οπότε προκύπτει

$$(n-1)\,T(n-1) \;=\; (n-1)^2 + \sum_{k=2}^{n-2} T(k)$$

Αφαιρώντας τα αντίστοιχα σκέλη των δύο τελευταίων εκφράσεων και εκτελώντας απλή άλγεβρα, προκύπτει:

$$\begin{aligned}
n\,T(n) - (n-1)\,T(n-1) &= 2n-1+T(n-1) \;\Rightarrow \\
T(n) - T(n-1) &= 2 - \frac{1}{n} \;\Rightarrow \\
T(n-1) - T(n-2) &= 2 - \frac{1}{n-1} \;\Rightarrow \\
&\vdots \\
T(2) - T(1) &= 2 - \frac{1}{2}
\end{aligned}$$

Αθροίζοντας αντίστοιχα τα αριστερά και τα δεξιά σκέλη και απλοποιώντας προκύπτει:

$$\begin{aligned}
T(n) &= 2(n-1) - \left(\frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}\right) \\
&= 2(n-1) + 1 - \left(1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}\right) \\
&= 2n - 1 - H_n
\end{aligned}$$

Καθώς για τον αρμονικό αριθμό ισχύει $H_n \approx \ln n$, έπεται ότι η πολυπλοκότητα της μέσης περίπτωσης της διαδικασίας `find` είναι $\Theta(n)$. Ομοίως, γραμμική είναι και η πολυπλοκότητα της καλύτερης περίπτωσης. □

9.4.2 Στατιστικά σε Γραμμικό Χρόνο Χειρότερης Περίπτωσης

Σε αυτή την ενότητα θα μελετήσουμε τον γραμμικό αλγόριθμο `Select`, για την εύρεση του $k-$οστού μεγαλύτερου στοιχείου ο οποίος αναπτύχθηκε από τους Blum et. al. [10]. Το πρόβλημα που είχε ο αλγόριθμος `find` ήταν ότι ο διαχωρισμός της αρχικής ακολουθίας δεν ήταν κατ' ανάγκην καλός και μπορούσε να οδηγήσει σε μεγάλα μεγέθη υποπροβλημάτων.

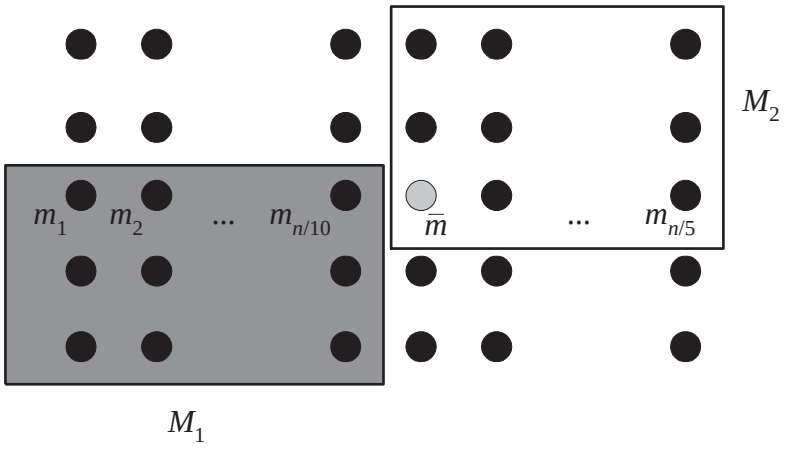
Ο αλγόριθμος `Select`, βασίζεται στην εξής ιδέα για τον καλύτερο διαχωρισμό:

> Αυτό που χρειάζεται είναι να επιλέξουμε το στοιχείο διαχωρισμού, βάσει ενός καλύτερου δείγματος. Αυτό το δείγμα μπορούμε να το χτίσουμε χωρίζοντας το M σε ομάδες π.χ. των 5 στοιχείων, να λάβουμε τους μέσους για κάθε ομάδα και με τους μέσους να κατασκευάσουμε μια μεγαλύτερη ομάδα, η οποία αποτελεί δειγματισμό των στοιχείων του M. Ο μέσος αυτής της ομάδας (μέσος των μέσων) είναι εγγυημένα καλό στοιχείο διαχωρισμού.

Η ιδέα αυτή παράγει σωστό διαχωρισμό, πέραν αυτού του βήματος όμως, ο αλγόριθμος λειτουργεί ακριβώς όπως ο `find`. Ο παρακάτω ψευδοκώδικας, υλοποιεί τον αλγόριθμο `Select`.

```
Select(M, i)
(* Εύρεση του i−οστού μεγαλύτερου στοιχείου του M *)
1.   n ← |M|;
2.   if n ≤ 100 then
3.        Ταξινόμηση του M και απευθείας εύρεση του i−οστού μεγαλύτερου;
4.   else
5.        Διαχωρισμός του M σε υποσύνολα M_1, M_2, ..., M_⌈n/5⌉ 5 στοιχείων το καθένα
          (το M_⌈n/5⌉, ενδέχεται να περιέχει ≤ 5 στοιχεία);
6.        Ταξινόμηση M_j, 1 ≤ j ≤ ⌈n/5⌉;
7.        Απευθείας εύρεση του μέσου m_j σε καθένα M_j;
8.        m̄ ←Select((m_1, m_2, ..., m_⌈n/5⌉), ⌈n/10⌉);
(* Εύρεση του μέσου των μέσων, έστω m̄ *)
9.        M_1 ← {m ∈ M; m̄ < s}; (* Τα μικρότερα του m̄ στοιχεία *)
10.       M_2 ← {m ∈ M; m̄ > s}; (* Τα μεγαλύτερα του m̄ στοιχεία *)
11.       if i ≤ |M_1| then
12.            Select(M_1, i);
13.       else
14.            Select(M_2, i − |M_1|);
15.       fi
16.  fi
```

Η διαδικασία που εκτελείται στις γραμμές 5-8 μπορεί να φανεί στο Σχήμα 9.4. Στη γραμμή 5, χωρίζουμε το M σε ομάδες των 5 στοιχείων τις $M_1, M_2, \ldots, M_{\lceil \frac{n}{5} \rceil}$, τις οποίες ταξινομούμε (γραμμή 6) και βρίσκουμε τον μέσο κάθε ομάδας, έστω



Σχήμα 9.4: Διαίρεση σε πεντάδες. Τα στοιχεία του M_1 είναι μικρότερα του $\overline{m}$ και τα στοιχεία M_2 μεγαλύτερα ίσα του $\overline{m}$

m_j ο μέσος της M_j. Κάθε ομάδα απεικονίζεται με μια στήλη και η τοποθέτηση των στοιχείων στη στήλη είναι τέτοια ώστε οι τιμές να αυξάνουν καθώς κινούμαστε προς τα πάνω σε κάθε στήλη. Στη γραμμή 8, βρίσκουμε το στοιχείο $\overline{m}$, το οποίο είναι ο μέσος όλων των m_j, $1 \leq j \leq \lceil \frac{n}{5} \rceil$, δηλαδή *ο μέσος των μέσων*, $\overline{m}$, το στοιχείο που είναι εγγυημένα καλός διαχωριστής. Από το σημείο αυτό και έπειτα ο αλγόριθμος προχωρά, όπως ακριβώς και στο `find`.

Τα στοιχεία που περικλείονται από το γκρί ορθογώνιο στο Σχήμα 9.4 είναι αυτά τα οποία σίγουρα έχουν τιμή μικρότερη ίση από το $\overline{m}$ και άρα σίγουρα ανήκουν στο M_1. Αυτό μπορεί να φανεί, δεδομένου ότι τα στοιχεία του γκρίζου ορθογωνίου είναι μικρότερα από τον μέσο της στήλης στην οποία ανήκουν και ο μέσος καθεμίας από της στήλες είναι μικρότερος από τον μέσο των μέσων, $\overline{m}$. Με εντελώς όμοιο συλλογισμό, τα στοιχεία του διαφανούς ορθογωνίου, είναι όλα μεγαλύτερα ίσα του $\overline{m}$ και ανήκουν στο M_2. Τα δε M_1, M_2, περιέχουν πιθανώς και άλλα στοιχεία, αλλά αυτά που περιβάλλονται από ορθογώνια, περιέχονται σίγουρα σε αυτά τα σύνολα.

Φράσσοντας τον μέγιστο αριθμό στοιχείων που μπορεί να έχει ένα τέτοιο σύνολο, δίνει την δυνατότητα φραγής του μέγιστου κόστους μιας αναδρομικής κλήσης της `Select`, κάτι που θα βοηθήσει στην απόδειξη της γραμμικότητάς της. Το παρακάτω Λήμμα προκύπτει εύκολα.

Λήμμα 9.1. *Το ελάχιστο μέγεθος του M_1 ή του M_2 είναι $\frac{3n}{10} - 6$*

Απόδειξη. Από την παρατήρηση του Σχήματος 9.4, προκύπτει ότι καθένα από τα ορθογώνια περιέχει τουλάχιστον $3\left(\left\lceil \frac{1}{2}\left\lceil \frac{n}{5}\right\rceil\right\rceil - 2\right) \geq \frac{3n}{10} - 6$ στοιχεία. Αυτό αφού δεν μετράμε την τελευταία πεντάδα που μπορεί να μην είναι γεμάτη καθώς και την πεντάδα που περιέχει το $\overline{m}$. ∎

Από το Λήμμα 9.1 προκύπτει ότι το μέγιστο υποπρόβλημα αν λάβουμε σαν στοιχείο διαχωρισμού το $\overline{m}$ περιέχει $\frac{7n}{10} + 6$ στοιχεία. Έστω $T(n)$, ο μέγιστος χρόνος εκτέλεσης του αλγορίθμου, για κάθε σύνολο M που περιέχει n στοιχεία και κάθε i.

Λήμμα 9.2. *Η παρακάτω αναδρομική σχέση περιγράφει την πολυπλοκότητα του αλγορίθμου* `Select`.

$$T(n) \leq \begin{cases} \Theta(1) & , \text{ για } n \leq 80 \\ T(\lceil n/5 \rceil) + T(7n/10 + 6) + g(n) & , \text{ για } n > 80 \end{cases}$$

Απόδειξη. Ο αλγόριθμος θα εκτελεστεί μια φορά για τον μέσο των μέσων (γραμμή 8), δηλαδή για $\lceil n/5 \rceil$ στοιχεία, οπότε προκύπτει ο όρος $T(\lceil n/5 \rceil)$ και μια φορά αναδρομικά, είτε για το M_1 (γραμμή 12) είτε για το M_2 (γραμμή 14), όπου ο χρόνος που ξοδεύεται είναι το πολύ $T(7n/10 + 6)$, λόγω του Λήμματος 9.1. Επίσης θα πρέπει $7n/10 + 6 < n \Rightarrow n > 20$. Ο όρος $g(n) = O(n)$ αφού οι υπόλοιπες γραμμές του προγράμματος απαιτούν γραμμικό πλήθος βημάτων. Έστω λοιπόν σταθερά b έτσι ώστε $g(n) \leq bn$, για κάθε n μεγαλύτερο από μία σταθερά. ∎

Θεώρημα 9.1. *Ο αλγόριθμος* `Select` *τρέχει σε χρόνο γραμμικό ως προς το πλήθος της εισόδου*

Απόδειξη. Θα αποδείξουμε με επαγωγή ότι $T(n) \leq cn$, όπου c μία σταθερά εξαρτώμενη από τη σταθερά στον όρο $O(n)$ της αναδρομικής σχέσης.

Για $n \leq 80$: Προφανές.

Για $n > 80$: Υποθέτουμε ότι ισχύει $T(\ell) \leq c\ell$, $\ell < n$. Θα δείξουμε ότι ισχύει και για $\ell = n$. Έχουμε

$$\begin{aligned} T(n) &\leq T(\lceil n/5 \rceil) + T(7n/10 + 6) + bn \\ &\leq c\lceil n/5 \rceil + \frac{7cn}{10} + 6c + bn \\ &\leq \frac{cn}{5} + c + \frac{7cn}{10} + 6c + bn \\ &= \frac{9cn}{10} + 7c + bn \leq cn \end{aligned} \tag{9.1}$$

όπου η 9.1 ισχύει για

$$\frac{bn}{n + 7 + \frac{9n}{10}} \leq c$$

Αυτή η ανισότητα ισχύει για κάθε $n > 80$ αν καθορίζουμε επακριβώς το b.

Συνολικά έχουμε

$$T(n) \leq \begin{cases} \Theta(1), & n \leq 80 \\ cn & n > 80 \end{cases}$$

■

9.5 Ταξινόμηση με Συγχώνευση

Με τον όρο συγχώνευση (merging) εννοούμε την πράξη της δημιουργίας ενός νέου ταξινομημένου πίνακα που περιέχει όλα τα στοιχεία δύο (ή και περισσοτέρων) πινάκων, που είναι ήδη ταξινομημένοι. Η διαδικασία `Merge` που ακολουθεί υλοποιεί τη συγχώνευση δύο ταξινομημένων πινάκων `A` και `B` (με n και m ακεραίους αντίστοιχα) στον πίνακα `C`. Θεωρείται ότι οι δύο πίνακες έχουν και μία επιπλέον θέση που χρησιμεύει για την αποθήκευση ενός φρουρού (sentinel).

```
     procedure merge;
1.   i <-- 1; j <-- 1;
2.   A[n+1] <-- maxint; B[m+1] <-- maxint;
2.   for k <-- 1 to n+m do
3.      if A[i]<B[j] then
4.         C[k] <-- A[i]; i <-- i+1
5.      else
6.         C[k] <-- B[j]; j <-- j+1
```

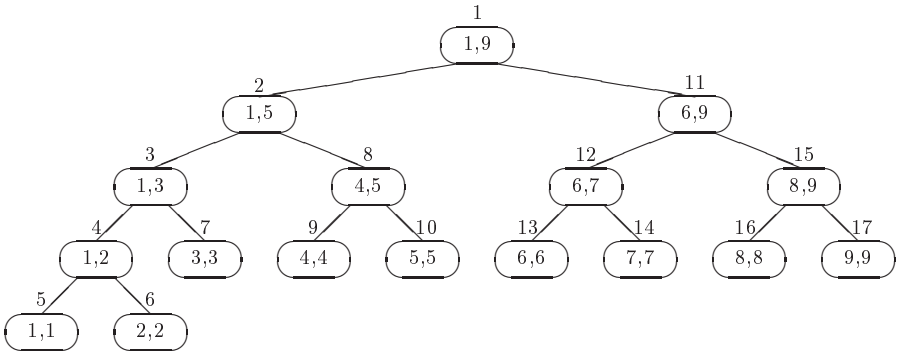
Στην ουσία η συγχώνευση γίνεται με τη βοήθεια δύο δεικτών `i` και `j` που σαρώνουν τους δύο πίνακες και διαδοχικά συγκρίνουν στοιχεία από τους πίνακες `A` και `B`. Έτσι, κάθε φορά το μικρότερο στοιχείο μεταφέρεται στον πίνακα εξόδου `C`. Ο αναγνώστης μπορεί πολύ εύκολα να καταλάβει τη λειτουργία της διαδικασίας `Merge` και δεν χρειάζονται περισσότερες επεξηγήσεις. Από το βρόχο της εντολής 2, είναι προφανές ότι η πολυπλοκότητα της συγχώνευσης είναι $\Theta(n+m)$, θεωρώντας φραγμένο από επάνω το κόστος της ερώτησης της εντολής 3-6.

Στη φιλοσοφία της διαδικασίας αυτής βασίζεται μία σημαντικότατη μέθοδος ταξινόμησης, η ονομαζόμενη ταξινόμηση με ευθεία συγχώνευση (straight merge sort). Κατά καιρούς η μέθοδος αυτή έχει υλοποιηθεί με πλήθος τρόπων, όπως επαναληπτικά ή αναδρομικά, για λίστες ή πίνακες κοκ. Στη συνέχεια ακολουθεί μία εκδοχή της για πίνακες, η οποία αποτελείται από δύο τμήματα: την αναδρομική `merge_sort` και τη `merge`. Στις εντολές 16-18 της `merge_sort` βρίσκεται το μεσαίο στοιχείο του πίνακα και εκτελούνται 2 κλήσεις στους δύο υποπίνακες που προκύπτουν. Από εδώ προκύπτει ότι η μέθοδος ανήκει στην οικογένεια των αλγορίθμων Διαίρει και Βασίλευε. Η διαδικασία `merge` που καλείται όταν προκύψουν στοιχειώδεις υπο-...-υπο-πίνακες αναλαμβάνει τις συγκρίσεις των στοιχείων και τα τακτοποιεί με τη βοήθεια ενός βοηθητικού πίνακα `B`.

```
     procedure merge(left,middle,right);
1.   first <-- left; second <-- middle+1; temp <-- left;
2.   while (first<=middle) and (second<=right) do
3.      if A[first]<=A[second] then
4.         B[temp] <-- A[first]; first <-- first+1;
5.      else
6.         B[temp] <-- A[second]; second <-- second+1;
7.      temp <-- temp+1
8.   if first<=middle then
9.      for k <-- first to middle do
10.        B[temp] <-- A[k]; temp <-- temp+1
11.  else
12.     for k <-- second to right do
13.        B[temp] <-- A[k]; temp <-- temp+1
14.  for k <-- left to right do A[k] <-- B[k]

     procedure merge_sort(left,right);
15.  if left<right then
```

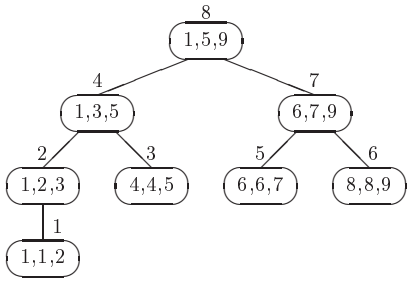
```
16.     middle <-- (left+right) div 2;
17.     merge_sort(left,middle);
18.     merge_sort(middle+1,right);
19.     merge(left,middle,right)
```



Σχήμα 9.5: Σειρά κλήσεων της merge_sort.

Αν εφαρμόσουμε τις ανωτέρω διαδικασίες σε πίνακα με τα 9 γνωστά κλειδιά, τότε η mergesort θα κληθεί 17 με τη σειρά που απεικονίζεται επάνω από κάθε κόμβο του επομένου σχήματος, ενώ μέσα σε κάθε κόμβο φαίνονται τα ορίσματα της αντίστοιχης κλήσης. Στο Σχήμα 9.6 φαίνονται 8 κλήσεις της merge, με την αντίστοιχη τάξη εκτός του κόμβου και τα ορίσματα εντός του κόμβου. Και στις δύο περιπτώσεις η διάσχιση του δένδρου ακολουθεί τη λογική της αναζήτησης με προτεραιότητα βάθους (dfs).

Πρόταση.
Η πολυπλοκότητα της `merge_sort` είναι $\Theta(n \log n)$ στη χειρότερη περίπτωση.



Σχήμα 9.6: Σειρά κλήσεων της merge.

Απόδειξη
Παρατηρώντας τη διαδικασία `merge_sort` εξάγουμε την επόμενη αναδρομική εξίσωση:

$$T(n) \;=\; T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n$$

με αρχική συνθήκη $T(1) = 1$, ενώ οι τρεις όροι του δεξιού σκέλους αντιστοιχούν στις εντολές 17-19. Συγκεκριμένα το κόστος σε συγκρίσεις της εντολής 19 προκύπτει από όσα αναφέρθηκαν προηγουμένως για τη διαδικασία συγχώνευσης 2 πινάκων. Για την ευκολία της ανάλυσης υποθέτουμε ότι $n = 2^k$, οπότε η ανωτέρω αναδρομική εξίσωση είναι πλέον εύκολη υπόθεση καθώς ισχύει διαδοχικά:

$$\begin{aligned} T(n) &= 2T(n/2) + n \;=\; 2(2T(n/4) + n/2) + n \;=\; 4T(n/4) + 2n \;=\; \ldots \\ &= 8T(n/8) + 3n \;=\; 2^k T(n/2^k) + kn \;=\; n + n\log n \;=\; \Theta(n \log n) \end{aligned}$$

□

Η μέθοδος ταξινόμησης με ευθεία συγχώνευση αξίζει την προσοχή μας γιατί είναι πολύ ευσταθής (stable) μέθοδος, δηλαδή ανεξάρτητα από τη φύση των δεδομένων θα εκτελέσει τον ίδιο αριθμό συγκρίσεων στη μέση και τη χειρότερη περίπτωση. Το χαρακτηριστικό αυτό έρχεται σε αντίθεση με το μειονέκτημα της γρήγορης ταξινόμησης, που διακρίνεται από πολυπλοκότητα $\mathrm{O}(n^2)$ για τη χειρότερη περίπτωση. Ωστόσο, το συμπέρασμα αυτό προκύπτει γιατί στην ανάλυση λάβαμε υπόψη μας μόνο το πλήθος των εκτελούμενων συγκρίσεων. Σε μία πραγματική υλοποίηση, η χρήση του βοηθητικού πίνακα `B` και οι εκτελούμενες καταχωρήσεις είναι επίσης πράξεις μη αμελητέου χρονικού κόστους, οι οποίες επιβαρύνουν τη συνολική επίδοση. Τελικώς, αν και η ταξινόμηση με συγχώνευση δεν είναι ίσως η καλύτερη μέθοδος ταξινόμησης στην κύρια μνήμη, εν τούτοις αποτελεί τη βάση για μεθόδους εξωτερικής ταξινόμησης.

9.6 Ταξινόμηση του Shell

Η μέθοδος που προτάθηκε από τον Shell έχει βασικό ότι χρησιμοποιεί μία ακολουθία ακεραίων $h_1, h_2, \ldots, h_k$, όπου ισχύει: $h_1 > h_2 > \ldots > h_{k-1} > h_k = 1$ και γι'αυτό φέρει και την ονομασία ταξινόμηση με **μειούμενες αυξήσεις** (diminishing increment). Στην πράξη η μέθοδος λειτουργεί για οποιεσδήποτε τιμές της ακολουθίας $h_1, h_2, \ldots, h_{k-1}$ αλλά θα πρέπει να ισχύει $h_k = 1$.

Θα μπορούσε να υποστηριχθεί ότι η ταξινόμηση του Shell είναι μία παραλλαγή της ταξινόμησης με εισαγωγή. Ουσιαστικά, η μέθοδος αποτελείται από k φάσεις, όπου στην i-οστή φάση (για $1 \leq i \leq k$) θεωρείται το βήμα h_i, οπότε ακολουθώντας τη λογική της ταξινόμησης με εισαγωγή τίθενται σε σωστή διάταξη μεταξύ

τους τα στοιχεία του πίνακα που απέχουν h_i θέσεις. Προφανώς η k-οστή και τελευταία φάση είναι ένα κλασικό πέρασμα της ταξινόμησης με εισαγωγή που ολοκληρώνει τη διαδικασία. Πρέπει να τονισθεί ότι αποδεδειγμένα, αν ο πίνακας είναι ταξινομημένος με βάση κάποιο βήμα, τότε παραμένει ταξινομημένος με βάση το ίδιο βήμα ακόμη και αν εφαρμοσθούν ένα η περισσότερα επόμενα βήματα στη συνέχεια. Επίσης, καθώς η μέθοδος αυτή στηρίζεται στην ταξινόμηση με εισαγωγή, είναι και αυτή ευσταθής, δηλαδή δεν ανταλλάσσει αμοιβαία στοιχεία με ίσα κλειδιά.

Το παράδειγμα του Σχήματος 9.7 δείχνει το περιεχόμενο του πίνακα με τα γνωστά 9 κλειδιά και τον τρόπο ανταλλαγής τους καθώς ο πίνακας ταξινομείται σταδιακά με 3 βήματα μεγέθους 4, 2 και 1. Μεταξύ των γραμμών που δείχνουν το πέρας κάθε φάσης, παρουσιάζονται οι εκτελούμενες ανταλλαγές στοιχείων του πίνακα. Στο παράδειγμα αυτό, επίσης, παρατηρούμε χαρακτηριστικά πως ο πίνακας παραμένει ταξινομημένος με βήμα 4 ακόμη και μετά το πέρας της φάσης με βήμα 2.

Αρχικά Κλειδιά	52	12	71	56	5	10	19	90	45
	5				52				
					45				52
		10				12			
			19				71		
Βήμα 4	5	10	19	56	45	12	71	90	52
							52		71
				12		56			
Βήμα 2	5	10	19	12	45	56	52	90	71
			12	19					
						52	56		
								71	90
Βήμα 1	5	10	12	19	45	52	56	71	90

Σχήμα 9.7: Ταξινόμηση με μειούμενες αυξήσεις.

Ο ψευδοκώδικας που ακολουθεί παρουσιάζει αλγοριθμικά το προηγούμενο σκεπτικό. Ο ψευδοκώδικας δεν είναι αρκετά γενικός καθώς θεωρεί μία απλή ακολουθία με 3 βήματα μεγέθους 4, 2 και 1, ώστε να συμφωνεί με το παράδειγμα. Η δεύτερη παρατήρηση είναι ότι για επιπλέον λόγους ευκολίας θεωρεί ότι στα αριστερά του πίνακα υπάρχουν τρεις επιπλέον θέσεις, όπου τοποθετούνται κόμβοι φρουροί με τιμές ίσες προς τα βήματα αντίστοιχα.

```
procedure shellsort;
```

```
1.   h[1] <-- 4; h[2] <-- 2; h[3] <-- 1;
2.   for m <--  1 to 3 do
3.      k <-- h[m]; s <-- -k
4.      for i <-- k+1 to n do
5.         x <-- A[i]; j <-- i-k;
6.         if s=0 then s <-- -k;
7.         s <-- s+1; A[s] <-- x;
8.         while x<A[j] do
9.            A[j+k] <-- A[j]; j <-- j-k
10.        A[j+k] <-- x
```

Η απλή αυτή λογική της μεθόδου του Shell πάσχει κατά το ότι η ακολουθία των ακεραίων παραμένει μία παράμετρος προς βελτιστοποίηση. Δηλαδή, έχουν προταθεί και έχουν αναλυθεί με περισσότερη ή λιγότερη δυσκολία αρκετές ακολουθίες, αλλά θα μπορούσε να προταθεί μία νέα ακολουθία και να αποδειχθεί ότι αυτή είναι καλύτερη από κάθε προηγούμενη και πιθανώς η βέλτιστη. Στη συνέχεια θα παρουσιάσουμε την ανάλυση της χειρότερης περίπτωσης για την ακολουθία που προτάθηκε από τον ίδιο τον Shell και δίνεται από τις σχέσεις: $h_1 = \lfloor n/2 \rfloor$ και $h_{i+1} = \lfloor h_i/2 \rfloor$.

Πρόταση.
Η πολυπλοκότητα της `shellsort` είναι $\Theta(n^2)$ στη χειρότερη περίπτωση, για την ακολουθία του Shell, όπου $h_1 = \lfloor n/2 \rfloor$ και $h_{i+1} = \lfloor h_i/2 \rfloor$.

Απόδειξη
Για να αποδείξουμε ότι η πολυπλοκότητα της χειρότερης περίπτωσης για την ακολουθία του Shell είναι $\Theta(n^2)$, πρώτα θα αποδείξουμε ότι ισχύει το $\Omega(n^2)$ και μετά το $O(n^2)$. Έστω ότι $n = 2^k$, οπότε κάθε βήμα είναι άρτιος αριθμός εκτός από το τελευταίο βήμα που ισούται με τη μονάδα. Έστω, επίσης, ότι αρχικά τα $n/2$ μεγαλύτερα στοιχεία τοποθετούνται στις άρτιες θέσεις, ενώ τα $n/2$ μικρότερα στοιχεία τοποθετούνται στις περιττές θέσεις. Μέχρι να φθάσουμε στο τελευταίο βήμα, όλα τα $n/2$ μεγαλύτερα στοιχεία είναι ακόμη στις άρτιες θέσεις και όλα τα $n/2$ στοιχεία είναι στις μικρότερες θέσεις. Επομένως το i-οστό μικρότερο στοιχείο (για $i \leq n/2$) βρίσκεται στην $(2i-1)$-οστή θέση πριν αρχίσει το τελικό πέρασμα και πρέπει να εκτελεσθούν $i-1$ ανταλλαγές μέχρι να φθάσει στην τελική του θέση. Άρα, για όλα τα $n/2$ μικρότερα στοιχεία θα απαιτηθούν $\sum_{i=1}^{n/2}(i-1) = \frac{n^2-n}{8} = \Omega(n^2)$ ανταλλαγές. Το Σχήμα 9.8 απεικονίζει το παράδειγμα που περιγράφηκε πριν.

Για να αποδείξουμε το δεύτερο σκέλος, αρκεί να θεωρήσουμε ότι κατά το

Αρχικά	1	5	2	6	3	7	4	8
Βήμα 4	1	5	2	6	3	7	4	8
Βήμα 2	1	5	2	6	3	7	4	8
Βήμα 1	1	2	3	4	5	6	7	8

Σχήμα 9.8: Ταξινόμηση με μειούμενες αυξήσεις.

πέρασμα με βήμα h_k, ουσιαστικά εκτελείται μία ταξινόμηση με εισαγωγή (τετραγωνικής πολυπλοκότητας) για ένα σύνολο n/h_k στοιχείων. Επομένως, για το σύνολο των περασμάτων με αυτό το βήμα η πολυπλοκότητα είναι $O(h_k(n/h_k)^2) = O(n^2/h_k)$. Επομένως για το σύνολο των βημάτων προκύπτει η πολυπλοκότητα $O(\sum_{i+1}^{t} n^2/h_t) = O(n^2 \sum_{i+1}^{t} 1/h_t)$. Καθώς η ακολουθία των βημάτων είναι γεωμετρική, έπεται ότι $\sum_{i+1}^{t} 1/h_t < 2$. Έτσι προκύπτει το ζητούμενο, δηλαδή ότι η πολυπλοκότητα της ταξινόμησης του Shell για τη δεδομένη ακολουθία είναι $O(n^2)$. □

Στη βιβλιογραφία έχουν παρουσιασθεί πολλές εναλλακτικές προτάσεις για την ακολουθία των βημάτων και έχουν αναλυθεί. Για παράδειγμα, για την ακολουθία $1, 3, 7, \ldots, 2^k - 1$ του Hibbard, αν και εικάζεται από πειραματικά αποτελέσματα ότι η πολυπλοκότητα είναι $O(n^{5/4})$, το όριο που έχει αποδειχθεί είναι $\Theta(n^{3/2})$.

9.7 Όρια Αλγορίθμων Ταξινόμησης

Μέχρι στιγμής εξετάσθηκαν μέθοδοι ταξινόμησης με πολυπλοκότητα της τάξης $O(n^2)$ ή $O(n \log n)$. Τι εκφράζει η κάθε μία από αυτές τις τάξεις πολυπλοκότητας; Ο μέθοδος της ταξινόμησης με εισαγωγή, της ταξινόμησης με επιλογή και της ταξινόμησης με ανταλλαγή (bubblesort) είναι οι λεγόμενες **ευθείες** (straight) μέθοδοι που είναι απλές στην κατανόηση, την κωδικοποίηση και την ανάλυση αλλά αποτελούν μία οικογένεια αργών αλγορίθμων ταξινόμησης. Για τις μεθόδους αυτές ισχύει η επόμενη θεώρηση.

Δοθέντος ενός πίνακα A με n διακριτά στοιχεία λέγεται ότι υπάρχει μία **αντιστροφή** (inversion) αν ισχύει $A[i] > A[j]$ για κάποια $i < j$. Με απλά λόγια, στην ακολουθία $S = 3, 14, 1, 5, 9$ οι αντιστροφές είναι (3,1), (14,1), (14,5) και(14,9).

Πρόταση.

Κάθε αλγόριθμος που σε κάθε βήμα απομακρύνει το πολύ μία αντιστροφή απαιτεί $n(n-1)/2$ βήματα στη χειρότερη περίπτωση και $n(n-1)/4$ βήματα στη μέση περίπτωση.

Απόδειξη.
Αν υποθέσουμε ότι ο πίνακας A περιέχει μία τυχαία διάταξη των τιμών $1, 2, \ldots, n$ από το σύνολο των $n!$ διατάξεων. Ορίζουμε ως **ανεστραμμένο** (reverse) του πίνακα A, τον πίνακα $\mathrm{A}^R = (\mathrm{A}[n], \mathrm{A}[n-1], \ldots, \mathrm{A}[1])$. Σημειώνεται ότι $(\mathrm{A}^R)^R = \mathrm{A}$, ενώ για $n > 1$ δεν υπάρχει περίπτωση κάποιος πίνακας να αποτελεί αντίστροφο του εαυτού του. Έτσι οι $n!$ διατάξεις μπορούν να χωρισθούν σε $n!/2$ ζεύγη, όπου κάθε ζεύγος αποτελείται από μία διάταξη και την αντεστραμμένη της. Υπάρχουν $n(n-1)/2$ ζεύγη τιμών i, j όπου $1 \leq j < i \leq n$. Για κάθε τέτοιο ζεύγος τιμών (i, j) και για κάθε ζεύγος διάταξης και της αντεστραμμένης της $(\mathrm{A}, \mathrm{A}^R)$, θα υπάρχει μόνο μία αντιστροφή στη μία διάταξη από τις δύο του κάθε ζεύγους. Επομένως ο συνολικός αριθμός αντιστροφών σε όλες τις διατάξεις είναι $n(n-1)/2$, ενώ η μέση τιμή των αντιστροφών είναι $n(n-1)/4$. □

Πρόταση.
Δεδομένου ενός πίνακα που περιέχει μία τυχαία διάταξη με n στοιχεία, το άθροισμα των αποστάσεων που διανύονται από τα στοιχεία κατά την ταξινόμησή τους είναι $(n^2-1)/3$.

Απόδειξη.
Αν υποθέσουμε ότι ο πίνακας A περιέχει μία τυχαία διάταξη των τιμών $1, 2, \ldots, n$. Δοθέντος ενός τυχαίου στοιχείου $\mathrm{A}[j]$, η απόσταση που θα διανύσει είναι $\mathrm{A}[j] - j$. Επομένως θεωρώντας το σύνολο των στοιχείων ισχύει ότι η μέση τιμή της απόστασης ισούται με:

$$\begin{aligned}
\mathrm{E}[\mathrm{A}[j] - j] &= \frac{|1-j| + |2-j| + \ldots + |j-j| + \ldots + |n-j|}{n} \\
&= \frac{1}{n}\left(\sum_{i=1}^{j-1} i + \sum_{i=1}^{n-j} i\right) = \frac{1}{n}\left(\frac{j(j-1)}{2} + \frac{(n-j)(n-j+1)}{2}\right)
\end{aligned}$$

Συνεπώς η μέση διανυόμενη απόσταση είναι:

$$\sum_{j=1}^{n} (\mathrm{E}[\mathrm{A}[j] - j]) = \frac{1}{n}\sum_{j=1}^{n}\left(\frac{j(j-1)}{2} + \frac{(n-j)(n-j+1)}{2}\right)$$

Οι όροι εντός του αθροίσματος είναι ίδιοι για συμπληρωματικές τιμές του j και

επομένως ισχύει:

$$\sum_{j=1}^{n} (\mathrm{E}[\mathrm{A}[j] - j]) \;=\; \frac{1}{n} \sum_{j=1}^{n} j(j-1) \;=\; \ldots \;=\; \frac{n^2-1}{3}$$

Συνεπώς ένας αλγόριθμος ταξινόμησης που μετακινεί τα στοιχεία κατά ένα σταθερό αριθμό θέσεων σε κάθε βήμα απαιτεί $\mathrm{O}(n^2)$ βήματα. □

Τελικά τίθεται το ερώτημα "Πόσο γρήγορα μπορεί μία μέθοδος να ταξινομήσει έναν πίνακα;". Θα αποδειχθεί ότι όταν μία μέθοδος ταξινόμησης στηρίζεται μόνο σε συγκρίσεις και ανταλλαγές κλειδιών, τότε στη χειρότερη περίπτωση δεν μπορεί να ταξινομήσει σε χρόνο κατώτερο του $\mathrm{O}(n \log n)$.

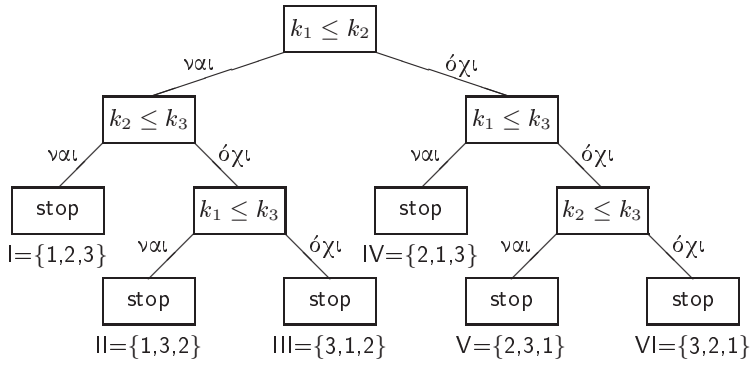
Πριν την απόδειξη της σχετική πρότασης είναι αναγκαίο να περιγραφεί η διαδικασία ταξινόμησης με ένα **δένδρο αποφάσεων** (decision tree). Ένα μονοπάτι του δένδρου δείχνει μία πιθανή διαδοχή από υπολογισμούς που ο αλγόριθμος πιθανώς να ακολουθήσει. Για παράδειγμα, ας υποτεθεί ότι πρέπει να ταξινομηθούν τρία κλειδιά: τα k_1, k_2 και k_3 αντίστοιχα. Η σειρά εισόδου των κλειδιών είναι k_1, k_2 και k_3 που φαίνεται στο δένδρο αποφάσεων ως (1,2,3). Αρχικά συγκρίνονται τα κλειδιά k_1 και k_2. Αν $k_1 < k_2$ τότε η σειρά (1,2,3) δεν αλλάζει, αλλιώς γίνεται (2,1,3).

Στο επόμενο σχήμα φαίνονται όλα τα πιθανά ενδεχόμενα που μπορούν να συμβούν σε έναν αλγόριθμο ταξινόμησης ανάλογα με τη σειρά σύγκρισης των κλειδιών στο δένδρο. Από το σχήμα αυτό φαίνεται ότι το δένδρο αποφάσεων είναι ένα δυαδικό δένδρο. Τα φύλλα του δένδρου είναι αριθμημένα από I ως VI και αποτελούν τα μοναδικά σημεία τερματισμού κάθε αλγορίθμου ταξινόμησης. Το δένδρο αυτό έχει 3!=6 φύλλα που εξασφαλίζει ότι ο αλγόριθμος βρίσκει πάντοτε τη διάταξη εκείνη που αντιστοιχεί στην ταξινομημένη σειρά.

Πρόταση.
Κάθε δένδρο αποφάσεων που ταξινομεί n διακεκριμένα κλειδιά έχει ύψος τουλάχιστον $\log n!$.

Απόδειξη.
Ένα δένδρο αποφάσεων έχει $n!$ φύλλα που αντιστοιχούν σε κάθε μία από τις $n!$ διατάξεις των n κλειδιών. Ας υποτεθεί ότι το δένδρο είναι πλήρες και ότι το ύψος του είναι h. Τότε ο αριθμός των φύλλων είναι 2^{h-1}, οπότε συνεπάγεται ότι $n! = 2^{h-1}$. Λογαριθμώντας και τα δύο μέλη της σχέσης προκύπτει:

$$\log n! \;=\; \log 2^{h-1} \;=\; h-1$$



Σχήμα 9.9: Δένδρο αποφάσεων.

Το δένδρο δεν είναι πλήρες και άρα η πρόταση ισχύει. □

Πρόταση.
Κάθε αλγόριθμος που ταξινομεί κάνοντας μόνο συγκρίσεις και ανταλλαγές κλειδιών έχει στη χειρότερη περίπτωση χρόνο τάξης $\mathrm{O}(n \log n)$.

Απόδειξη.
Πρέπει να αποδειχθεί ότι σε κάθε δένδρο αποφάσεων με $n!$ φύλλα υπάρχει ένα μονοπάτι μήκους $c \cdot n \log n$, όπου c είναι μία σταθερά. Από την προηγούμενη πρόταση προκύπτει ότι κάθε μονοπάτι είναι μήκους $\log n!$. Αντικαθιστώντας με βάση τον τύπο του Stirling προκύπτει:

$$h \;=\; \log n! \;=\; \log \sqrt{2\pi n} + n \log n - n \log e \;=\; \mathrm{O}(n \log n)$$

□

9.8 Ταξινόμηση με Μέτρημα

Ας υποθέσουμε ότι δίνεται ένας πίνακας A με n στοιχεία, που λαμβάνουν τιμές από 1 μέχρι k, όπου $k < n$. Θα εκτελέσουμε μία ταξινόμηση με τη βοήθεια ενός βοηθητικού πίνακα C μεγέθους k, ενώ το αποτέλεσμα θα αποθηκευθεί στον πίνακα B. Με απλά λόγια η ταξινόμηση αυτή δεν είναι επιτόπια. Δίνεται ο επόμενος ψευδοκώδικας και θα σχολιασθεί στη συνέχεια με τη βοήθεια ενός παραδείγματος.

```
     procedure countsort
1.   for i <-- 1 to k do C[i] <-- 0;
```

```
2.   for i <-- 1 to n do C[A[i]] <-- C[A[i]]+1;
3.   for i <-- 2 to k do C[i] <-- C[i]+C[i-1]
4.   for i <-- n downto 1 do
5.      B[C[A[i]]] <-- A[i]; C[A[i]] <-- C[A[i]]-1
```

5	7	3	5	2	1	4	3

Πίνακας 9.1: Ταξινόμηση με μέτρημα (αρχικός πίνακας).

Έστω $n = 8, k = 7$ και ο πίνακας A με το περιεχόμενο του Πίνακα 9.1. Με την εντολή 2 καταμετρώνται οι εμφανίσεις της κάθε τιμής (από τις k) και τοποθετούνται σε αντίστοιχες θέσεις του πίνακα C. Με την εντολή 3 σαρώνεται ο πίνακας C και προκύπτει η νέα μορφή του C, όπου κάθε θέση προκύπτει ως το άθροισμα των τιμών των δύο προηγούμενων θέσεων. Το περιεχόμενο του πίνακα C μετά το πέρας των εντολών 2-3 δίνονται στον Πίνακα 9.2. Με το βρόχο της εντολής 4, στον πίνακα B τοποθετείται η ζητούμενη ταξινομημένη ακολουθία και τερματίζει ο αλγόριθμος.

1	1	2	1	2	0	1

1	2	4	5	7	7	8

Πίνακας 9.2: Ταξινόμηση με μέτρημα (βοηθητικός πίνακας).

Παρατηρούμε ότι ο ψευδοκώδικας αποτελείται από τέσσερις διαδοχικούς βρόχους. Εξ αυτών οι δύο εκτελούν n επαναλήψεις, ενώ άλλοι δύο εκτελούν k επαναλήψεις. Ευνόητο είναι ότι η πολυπλοκότητα του αλγορίθμου $\Theta(n + k)$. Με την παραδοχή ότι $k < n$, έπεται ότι η πολυπλοκότητα είναι $\Theta(n)$. Επιτύχαμε δηλαδή ένα γραμμικό αλγόριθμο ταξινόμησης σε αντίθεση με τα προηγούμενα; Η ερώτηση είναι παγίδα γιατί στην παρούσα περίπτωση (α) ο αλγόριθμος δεν στηρίζεται στις συγκρίσεις, και (β) αν $k >> n$ τότε προφανώς δεν ισχύει το τελικό συμπέρασμα.

Τώρα ας προσέξουμε τον επόμενο κώδικα. Οι πρώτες δύο εντολές είναι πανομοιότυπες με τις πρώτες δύο εντολές της διαδικασίας `countsort` αλλά στη συνέχεια υπάρχει διαφορά. Πιο συγκεκριμένα, δεν υπάρχει ο βοηθητικός πίνακας B, και επίσης παρατηρούμε ένα διπλό βρόχο στις εντολές 4-6. Αν εκτελέσουμε

τον αλγόριθμο, έστω και με χαρτί και μολύβι, θα διαπιστώσουμε ότι ουσιαστικά εκτελεί την ίδια λειτουργία με την προηγούμενη διαδικασία.

```
     procedure countsort2
1.   for i <-- 1 to k do C[i] <-- 0;
2.   for i <-- 1 to n do C[A[i]] <-- C[A[i]]+1;
3.   i <-- 0;
4.   for j <-- 1 to k do
5.      for m <-- C[j] down to 1 do
6.         i <-- i+1; A[i] <-- j
```

Πρόταση.
Η πολυπλοκότητα της `countsort2` είναι $\Theta(n+k)$ στη χειρότερη περίπτωση.

Απόδειξη.
Η περίπτωση θυμίζει το παράδειγμα που είχαμε επιλύσει στο Κεφάλαιο 1.5 (όπου το άνω όριο του βρόχου δεν ήταν ένας κλασικός δείκτης αλλά το περιεχόμενο μίας θέσης του πίνακα). Σύμφωνα, λοιπόν, με μία πρώτη προσέγγιση, με βάση τα όρια των βρόχων `for` μπορούμε να φθάσουμε στο συμπέρασμα ότι η πολυπλοκότητα είναι $O(kn)$, επειδή n είναι η τιμή του `C[i]` στην εντολή 5 στη χειρότερη περίπτωση. Ωστόσο, η ανάλυση αυτή δεν είναι σφικτή.

Ο εξωτερικός βρόχος `for` της εντολής 4 είναι συγκεκριμένος, δηλαδή έχουμε k επαναλήψεις. Η εντολή 6 θα εκτελεσθεί ακριβώς n φορές και όχι kn. Αυτό προκύπτει με βάση την εξής παρατήρηση. Ας θεωρήσουμε την j-οστή επανάληψη του εξωτερικού βρόχου. Ο έλεγχος του εσωτερικού βρόχου `for` θα εκτελεσθεί `C[j]+1` φορές. Επομένως, ο συνολικός αριθμός ελέγχων στο σημείο αυτό είναι:

$$\sum_{i=1}^{k} (C[i]+1) \;=\; \sum_{i=1}^{k} C[i] \;+\; \sum_{i=1}^{k} 1 \;=\; n+k$$

Συνεπώς και πάλι καταλήγουμε στο ίδιο συμπέρασμα ως προς την πολυπλοκότητα της διαδικασίας. □

9.9 Ταξινόμηση με Βάση τη Ρίζα

Η **ταξινόμηση με βάση τη ρίζα** (radix sort) είναι μία γενίκευση της ταξινόμησης με μέτρημα. Με απλά λόγια, έστω ότι δίνονται προς ταξινόμηση ένας πίνακας με n ακεραίους μέγεθους d ψηφίων. Εκτελούμε μία ταξινόμηση με μέτρημα ως προς το τελευταίο, δηλαδή το λιγότερο σημαντικό ψηφίο (least significant digit).

Έτσι προκύπτει μία νέα μορφή του πίνακα, που και πάλι ταξινομούμε με μέτρημα αλλά αυτή τη φορά ως προς το δεύτερο ψηφίο από το τέλος. Η διαδικασία αυτή συνεχίζεται και τελειώνει όταν ο προκύπτων πίνακας ταξινομείται ως προς το πρώτο, δηλαδή το περισσότερο σημαντικό ψηφίο (most significant digit).

Σημειώνεται ότι στην ταξινόμηση με βάση τη ρίζα δεν υπάρχει περιορισμός ως προς το εύρος τιμών των κλειδιών, όπως υπάρχει στην ταξινόμηση με μέτρημα (η γνωστή παράμετρος k). Όμως όταν λαμβάνεται σε κάθε κλήση το αντίστοιχο ψηφίο, τότε εκεί προφανώς τα ψηφία λαμβάνουν τιμές εντός του διαστήματος 1-9, και έτσι συνιστάται η χρήση της ταξινόμησης με μέτρημα. Η μέθοδος θα μπορούσε να εφαρμοσθεί και σε συμβολοσειρές, όπου κάθε χαρακτήρας θα λάμβανε τιμές στο διάστημα a-z ή α-ω (δηλαδή, 26 ή 24 διαφορετικές τιμές αντίστοιχα). Από το χρησιμοποιούμενο, λοιπόν, αριθμητικό σύστημα (δηλαδή το δεκαδικό, δυαδικό κλπ. σε περίπτωση αριθμών ή το διάστημα 1-24/26 σε περίπτωση χαρακτήρων) προκύπτει η ονομασία της μεθόδου (radix). Η επόμενη συνοπτικότατη διαδικασία καταγράφει αυτήν την περιγραφή.

```
   procedure radixsort
1. for i <-- 1 to d do
2.    countsort(d)
```

Έστω ότι δίνονται προς ταξινόμηση ο πίνακας `A` με τα γνωστά 9 κλειδιά 52, 12, 71, 56, 5, 10, 19, 90 και 45, όπως φαίνεται στον Πίνακα 9.3. Αν καλέσουμε την `countsort` της εντολής 2 με όρισμα `d=1` (θεωρούμε ότι η αρίθμηση των ψηφίων αρχίζει από το λιγότερο σημαντικό), τότε θα προκύπτει η δεύτερη μορφή του πίνακα, ενώ με τη δεύτερη κλήση της `countsort` και όρισμα `d=2` θα προκύψει η τελική μορφή του πίνακα.

52	12	71	56	5	10	19	90	45

10	90	71	52	12	5	45	56	19

5	10	12	19	45	52	56	71	90

Πίνακας 9.3: Ταξινόμηση με βάση τη ρίζα.

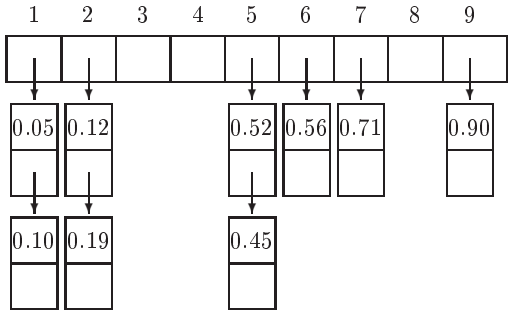
Η ανάλυση της πολυπλοκότητας της ταξινόμησης με βάση της ρίζα είναι εύκολη υπόθεση. Εφ'όσον η ταξινόμηση με μέτρημα έχει γραμμική πολυπλοκότητα $\Theta(n+k)$, έπεται ότι η πολυπλοκότητα της ταξινόμηση με βάση τη ρίζα είναι $\Theta(dn+dk)$. Θεωρώντας το d σταθερά, προκύπτει και πάλι γραμμική πολυπλοκότητα.

9.10 Ταξινόμηση με Κάδους

Η **ταξινόμηση με κάδους** (bucket sort) στηρίζεται στην υπόθεση ότι δίνονται προς ταξινόμηση n αριθμοί που ανήκουν στο διάστημα [0,1). Το διάστημα αυτό υποδιαιρείται σε n ίσα υποδιαστήματα (τους λεγόμενους κάδους), όπου κατανέμονται τα ισάριθμα στοιχεία προς ταξινόμηση. Επειδή θεωρούμε ότι τα στοιχεία υπακούουν σε μία ομοιόμορφη κατανομή, δεν αναμένουμε να ανήκουν πάρα πολλά στοιχεία σε ένα συγκεκριμένο υποδιάστημα. Η τελική ταξινομημένη ακολουθία προκύπτει ταξινομώντας τα στοιχεία του κάθε κάδου και κατόπιν λαμβάνοντας τους κάδους στη σειρά. Η επόμενη διαδικασία περιγράφει όσα αναφέραμε. Εκτός του πίνακα A με τα n στοιχεία, χρησιμοποιείται και ένας βοηθητικός πίνακας B με επίσης n θέσεις για τη δημιουργία συνδεδεμένων λιστών.

```
     procedure bucketsort
1.   for i <-- 1 to n do insert(A[i],B[floor(n*A[i])])
2.   for i <-- 1 to n do insertsort(B[i])
3.   return (B[1], B[2], ..., B[n])
```

Για παράδειγμα, έστω ότι δίνεται ο πίνακας A με τα γνωστά $n = 9$ στοιχεία από τα προηγούμενα παραδείγματα, διαιρεμένα δια 100. Θεωρούμε έναν πίνακα B με 9 θέσεις, όποτε τα υποδιαστήματα που αντιστοιχούν στις θέσεις του πίνακα B είναι [0-0,11), [0,11-0,22), [0,22-0,33), ..., [0,88-1). Το Σχήμα 9.10 απεικονίζει την κατάσταση μετά το πέρας της εντολής 1, οπότε έχουν εισαχθεί όλα τα στοιχεία στις αντίστοιχες συνδεδεμένες λίστες. Εννοείται ότι στη συνέχεια τα στοιχεία κάθε λίστας πρέπει να ταξινομηθούν, ώστε να δοθούν στην έξοδο.



Σχήμα 9.10: Ταξινόμηση με κάδους.

Πρόταση.
Η πολυπλοκότητα της `bucketsort` είναι γραμμική στη μέση περίπτωση.

Απόδειξη.
Προκειμένου να βρούμε την πολυπλοκότητα της μεθόδου πρέπει να εστιάσουμε στην εντολή 2, γιατί προφανώς η εντολή 1 έχει γραμμικό κόστος. Στην εντολή 2 ουσιαστικά εκτελούμε n ανεξάρτητες ταξινομήσεις με εισαγωγή με n_i στοιχεία η κάθε μία, όπου το n_i είναι μία τυχαία μεταβλητή. Επομένως, το συνολικό κόστος των εντολών 1-2 είναι:

$$\begin{aligned} \mathrm{E}[T(n)] &= \mathrm{E}\left[\Theta(n) + \sum_{i=1}^{n} \mathrm{O}\left(n_i^2\right)\right] \\ &= \Theta(n) + \sum_{i=1}^{n} \mathrm{E}\left[\mathrm{O}(n_i^2)\right] = \Theta(n) + \sum_{i=1}^{n} \mathrm{O}\left(\mathrm{E}[n_i^2]\right) \end{aligned}$$

Ορίζουμε μία τυχαία μεταβλητή X_{ij} που ισούται με 1 αν το στοιχείο $A[j]$ κατανέμεται στον i-οστό κάδο, ενώ αλλιώς ισούται με 0 (όπου $1 \leq i, j \leq n$). Άρα ισχύει $n_i = \sum_{j=1}^{n} X_{ij}$ και συνεπώς:

$$\begin{aligned} \mathrm{E}[n_i^2] &= \mathrm{E}\left[\left(\sum_{j=1}^{n} X_{ij}\right)^2\right] = \mathrm{E}\left[\sum_{j=1}^{n}\sum_{k=1}^{n} X_{ij}X_{ik}\right] \\ &= \mathrm{E}\left[\sum_{j=1}^{n} X_{ij}^2 + \sum_{j=1}^{n} \sum_{\substack{k=1 \\ k \neq j}}^{n} X_{ij}X_{ik}\right] \end{aligned}$$

Θα επιλύσουμε τα δύο αθροίσματα ξεχωριστά. Η τυχαία μεταβλητή X_{ij} ισούται με 1 με πιθανότητα $1/n$. Επομένως:

$$\mathrm{E}[X_{ij}^2] = 1 \times \frac{1}{n} + 0 \times (1 - \frac{1}{n}) = \frac{1}{n}$$

Σε σχέση με το δεύτερο άθροισμα ισχύει ότι $k \neq j$ και επομένως οι δύο τυχαίες μεταβλητές είναι ανεξάρτητες. Από το Κεφάλαιο 2.1 γνωρίζουμε ότι: $\mathrm{E}[XY] = \mathrm{E}[X] \times \mathrm{E}[Y]$. Συνεπώς:

$$\mathrm{E}[X_{ij}X_{ik}] = \mathrm{E}[X_{ij}]\ \mathrm{E}[X_{ik}] = \frac{1}{n}\,\frac{1}{n} = \frac{1}{n^2}$$

Άρα αντικαθιστώντας προκύπτει:

$$\begin{aligned} \mathrm{E}[n_i^2] &= \sum_{j=1}^{n} \mathrm{E}[X_{ij}^2] + \sum_{j=1}^{n} \sum_{\substack{k=1 \\ k \neq j}}^{n} \mathrm{E}[X_{ij}\ X_{ik}] \\ &= \sum_{j=1}^{n} \frac{1}{n} + \sum_{j=1}^{n} \sum_{\substack{k=1 \\ k \neq j}}^{n} \frac{1}{n^2} \\ &= n\frac{1}{n} + n(n-1)\frac{1}{n^2} = 2 - \frac{1}{n} \end{aligned}$$

Επιστρέφοντας στην αρχική σχέση έχουμε:

$$\mathrm{E}[T(n)] = \Theta(n) + \sum_{i=1}^{n}(2 - \frac{1}{n}) = \Theta(n) + \Theta(n) = \Theta(n)$$

□

Στη συνέχεια ακολουθεί μία κομψότερη απόδειξη του προηγούμενου αποτελέσματος.

Πρόταση.
Η πολυπλοκότητα της `bucketsort` είναι γραμμική στη μέση περίπτωση.

Απόδειξη.
Η πιθανότητα ένα συγκεκριμένο στοιχείο να κατανεμηθεί στον i-οστό κάδο είναι $p = 1/n$. Η πιθανότητα υπακούει μία δυωνυμική κατανομή με προσδοκητή τιμή $\mathrm{E}[n_i] = np = 1$ και απόκλιση $\mathrm{Var[n_i]} = np(1-p) = 1 - 1/n$. Από το Κεφάλαιο 2.1 γνωρίζουμε ότι για οποιαδήποτε τυχαία μεταβλητή ισχύει:

$$\mathrm{E}[n_i^2] = \mathrm{Var}[n_i] + \mathrm{E}^2[n_i]$$

Συνεπώς ισχύει:

$$\mathrm{E}[n_i^2] = 1 - \frac{1}{n} + 1^2 = 2 - \frac{1}{n} = \Theta(1)$$

Άρα, αντικαθιστώντας την τελευταία έκφραση στον αρχικό τύπο του $\mathrm{E}[T(n)]$, τελικώς προκύπτει ότι η ταξινόμηση με κάδους έχει γραμμική πολυπλοκότητα $\Theta(n)$ στη μέση περίπτωση. Στη χειρότερη περίπτωση η πολυπλοκότητα είναι τετραγωνική $\Theta(n^2)$. □

9.11 Βιβλιογραφική Συζήτηση

Η ταξινόμηση είναι ένα εξαιρετικά τιμημένο αντικείμενο της Πληροφορικής, καθώς έχει απασχολήσει στο παρελθόν πλήθος ερευνητών λόγω της θεωρητικής και πρακτικής σπουδαιότητάς του. Οι αλγόριθμοι ταξινόμησης που παρουσιάσθηκαν προηγουμένως μπορούν να βρεθούν σε οποιοδήποτε διδακτικό εγχειρίδιο για δομές δεδομένων και αλγορίθμους. Ο 3ος τόμος του βιβλίου του Knuth είναι ίσως το πληρέστερο και βαθύτερο κείμενο [77]. Σύμφωνα με τον Knuth, κάθε αλγόριθμος ταξινόμησης κατατάσσεται σε μία από πέντε κατηγορίες αναλόγως με τη βασική λειτουργία του. Οι κατηγορίες είναι: με εισαγωγή, με ανταλλαγή, με επιλογή, με συγχώνευση και με κατανομή. Πλούσιο σε υλικό είναι επίσης και το βιβλίο των Gonnet-BaezaYates [53].

Καλές επισκοπήσεις μπορούν να βρεθούν στα βιβλία και άρθρα [38, 93, 100, 101, 110, 112]. Ιδιαιτέρως, στο άρθρο [112] οι αλγόριθμοι ταξινόμησης κατατάσσονται σε δύο κατηγορίες: αυτούς που έχουν εύκολο διαχωρισμό και δύσκολη σύνδεση (όπως, ταξινόμηση με εισαγωγή, ταξινόμηση με συγχώνευση, ταξινόμηση με μειούμενες αυξήσεις), και σε αυτούς που έχουν δύσκολο διαχωρισμό και εύκολη σύνδεση (όπως, ταξινόμηση φυσαλίδας, ταξινόμηση επιλογής, γρήγορη ταξινόμηση).

Η λεγόμενη ταξινόμηση φυσαλίδας ή ταξινόμηση με αντικατάσταση έχει υπερ-παρουσιασθεί σε διδακτικά εγχειρίδια αν και είναι η χειρότερη τετραγωνική μέθοδος. Παρότι δεν εξετάσθηκε στα πλαίσια του παρόντος βιβλίου παρά μόνο στις Ασκήσεις 8-9, σχετικό υλικό μπορεί να βρεθεί στα άρθρα [31, 32, 166]. Από τις τετραγωνικές μεθόδους ξεχωρίζει η μέθοδος της ταξινόμησης με εισαγωγή, η οποία έχει μελετηθεί περισσότερο από τις άλλες αντίστοιχες μεθόδους [95, 159], θεωρείται μάλιστα ότι είναι η προσφορότερη όταν ο πίνακας είναι περίπου ταξινομημένος [26]. Η ανάλυση για την πολυπλοκότητα των μετακινήσεων στην ταξινόμηση με επιλογή αναφέρεται στο άρθρο [41].

Η γρήγορη ταξινόμηση προτάθηκε από τον Hoare [68], και υπήρξε πολύ δημοφιλής καθώς αποτέλεσε αφορμή για πλήθος παραλλαγών και αναλύσεων [22, 72, 92, 95, 115, 116, 117, 132, 138, 139, 140, 141, 161, 162, 164]. Βασική αναφορά για τη μέθοδο ταξινόμησης με συγχώνευση αποτελεί το άρθρο [16], ενώ παραλλαγές της μεθόδου προτείνονται στα άρθρα [124, 160]. Η ταξινόμηση shellsort προτάθηκε από τον Shell [148], ενώ ο Hibbard βελτίωσε σημαντικά την αρχική μέθοδο στο άρθρο [66]. Απόδειξη για την πολυπλοκότητα της μέθοδου χρησιμοποιώντας την ακολουθία Hibbard αναφέρεται στο βιβλίο [169]. Περαιτέρω, η μέθοδος μελετήθηκε εκτενώς, όπως στα άρθρα [11, 14, 36, 37, 47, 71, 128, 142, 165, 167, 168, 175].

Η χρήση του δέντρου απόφασης για το κάτω όριο ταξινόμησης βάσει συγκρίσεων οφείλεται στους Ford-Johnson [43]. Η παρουσίαση των ταξινομήσεων που δεν

βασίζονται στις συγκρίσεις ακολουθεί την προσέγγιση του βιβλίου των Cormen-Leiserson-Rivest-Stein [27].

Η Άσκηση 8 βασίζεται στο άρθρο [39], η Άσκηση 14 στο βιβλίο [61], η Άσκηση 15 στα άρθρα [115, 116, 117], η Άσκηση 22 στο άρθρο [26], η Άσκηση 23 στο άρθρο [64], ενώ η Άσκηση 25 στο άρθρο [54].

Στα πλαίσια του βιβλίου αυτού δεν εξετάζονται αλγόριθμοι εξωτερικής ταξινόμησης (δηλαδή, σε δευτερεύουσα μνήμη), καθώς περισσότερο ταιριάζουν στο αντικείμενο των Βάσεων Δεδομένων. Ωστόσο, ο ενδιαφερόμενος αναγνώστης παραπέμπεται στο βιβλίο της Salzberg [135]. Επίσης, ο αναγνώστης παραπέμπεται στο άρθρο [38], όπου εξετάζονται οι λεγόμενοι προσαρμοζόμενοι (adaptive) αλγόριθμοι ταξινόμησης, δηλαδή αλγόριθμοι των οποίων η πολυπλοκότητα εξαρτάται από την ποσότητα της αταξίας στον υπό ταξινόμηση πίνακα.

9.12 Ασκήσεις

1. Ποιά από τις επόμενες πράξεις μπορούν να υλοποιηθούν καλύτερα αν προηγουμένως ο αντίστοιχος πίνακας έχει ταξινομηθεί;

   - εύρεση της ελάχιστης τιμής,
   - εύρεση της μέγιστης τιμής,
   - εύρεση της μέσης τιμής,
   - εύρεση της μεσαίας τιμής, και
   - εύρεση της συχνότερης τιμής (mode).

2. Ποιός είναι ο ελάχιστος αριθμός συγκρίσεων για την ταξινόμηση 5 διακριτών ακεραίων; Πόσες συγκρίσεις και μετακινήσεις απαιτούνται στην καλύτερη, μέση και χειρότερη περίπτωση. Να επιλυθεί η άσκηση θεωρώντας 8 διακριτούς αριθμούς.

3. Επί ευθείας γραμμής τοποθετούνται $2n$ δίσκοι, όπου εναλλάσσονται ένας μαύρος και ένας λευκός. Ας υποθέσουμε ότι οι δίσκοι περιττής (άρτιας) τάξης είναι μαύροι (λευκοί). Πρέπει να συγκεντρώσουμε όλους τους μαύρους στα αριστερά και όλους τους λευκούς στα δεξιά της ευθείας γραμμής. Η μοναδική επιτρεπόμενη κίνηση είναι να ανταλλάξουμε μεταξύ τους δύο γειτονικούς δίσκους. Ποιός είναι ο ελάχιστος αριθμός μετακινήσεων;

4. Επί ευθείας γραμμής τοποθετούνται $2n$ δίσκοι, όπου οι πρώτοι n είναι μαύροι και επόμενοι n είναι λευκοί. Ποιός είναι ο ελάχιστος αριθμός

μετακινήσεων ώστε στη γραμμή να εναλλάσσονται μαύροι και λευκοί (ο πρώτος δίσκος να είναι μαύρος);

5. Να σχεδιασθεί και να αναλυθεί η μέθοδος **εισαγωγής με δυαδική εισαγωγή** (binary insertion sort), όπου χρησιμοποιείται η δυαδική αναζήτηση για την εύρεση της σωστής θέσης όπου θα γίνει η κάθε φορά εισαγωγή του επόμενου στοιχείου μέσα στην ακολουθία προορισμού. Η ανάλυση να θεωρήσει το πλήθος των συγκρίσεων και το πλήθος των μετακινήσεων.

6. Να σχεδιασθεί και να αναλυθεί αλγόριθμος που να εντοπίζει τους 2 μεγαλύτερους αριθμούς σε ένα πίνακα με n διαφορετικούς ακεραίους.

7. Δίνεται ένας πίνακας με n κλειδιά και ζητείται να βρεθούν τα μικρότερα k κλειδιά, όπου $1 < k < n/2$, χωρίς όμως να ταξινομηθεί πρώτα ο πίνακας. Να σχεδιασθούν τουλάχιστον δύο τρόποι.

8. Δίνεται αταξινόμητος πίνακας με n στοιχεία και ζητείται το k-οστό μεγαλύτερο. Να σχεδιασθεί λεπτομερώς αλγόριθμος που θα ταξινομεί τα πρώτα k στοιχεία και κατόπιν θα εξετάζει τα επόμενα $n-k$ με τη βοήθεια της ταξινόμησης με εισαγωγή. Να βρεθεί η πολυπλοκότητα του προτεινόμενου αλγορίθμου.

9. Σε ένα πίνακα `A[0..n-1]` τα στοιχεία `A[0]`, `A[2]`, `A[4]` κλπ είναι τα μικρότερα στοιχεία, ενώ τα `A[1]`, `A[3]`, `A[5]` κλπ είναι τα μεγαλύτερα. Να βρεθεί ο συνολικός αριθμός των συγκρίσεων (ως συνάρτηση του n) που θα κάνει η μέθοδος ταξινόμησης με εισαγωγή, και η μέθοδος του Shell με δύο μειούμενες αυξήσεις μεγέθους 2 και 1, αντιστοίχως.

10. Ο πίνακας `A[0..n-1]` περιέχει τις n διακριτές τιμές από 0 μέχρι $n-1$. Τι εκτελεί ο επόμενος κώδικας; Τι πολυπλοκότητα έχει; Ο πίνακας `B` ορίζεται ομοίως.

```
for (i=0; i<n; i++)
   B[A[i]]=A[i]
```

11. Δίνεται ο επόμενος κώδικας. Σε τι μοιάζει και σε τι διαφέρει με τον προηγούμενο κώδικα;

```
for (i=0; i<n; i++)
   while (A[i] != i)
      swap(i,A[i])
```

12. Σε ένα πίνακα υπάρχουν διπλά κλειδιά που πρέπει να απομακρυνθούν. Ποιά μέθοδος ταξινόμησης αν τροποποιηθεί εκτελεί αυτήν τη λειτουργία πιο αποτελεσματικά;

13. Έστω ότι πρέπει να ταξινομηθεί ένας πίνακας με n κλειδιά, όπου τα πρώτα n_1 κλειδιά είναι ήδη ταξινομημένα, ενώ τα τελευταία n_2 δεν είναι. (Ισχύει βέβαια $n = n_1 + n_2$.) Ας θεωρηθεί διαδοχικά ότι: $n_1 \gg n_2$ και ότι $n_1 \approx n_2$. Να σχεδιασθούν και να αναλυθούν εναλλακτικοί τρόποι για κάθε περίπτωση.

14. Δίνεται ταξινομημένος πίνακας `A[0..n-1]`. Με βάση τα στοιχεία του πίνακα αυτού υπολογίζονται τα στοιχεία:

$$b_1 = A[0] + A[1] \quad \text{και} \quad b_i = b_{i-1} + A[i] \ \text{ για } \ 1 < i \leq n-1$$

Να σχεδιασθεί γραμμικός αλγόριθμος δημιουργίας ενός πίνακα `B[0..2*n-2]` που να περιέχει τα στοιχεία του πίνακα `A` και τα στοιχεία b_i.

15. Η **ταξινόμηση με το μέσο όρο** (meansort) είναι μία παραλλαγή της γρήγορης ταξινόμησης σύμφωνα με την οποία κατά το πρώτο πέρασμα λαμβάνεται ως άξονας το κλειδί της πρώτης θέσης, όπως δηλαδή και στη γρήγορη ταξινόμηση. Όμως κατά τη διάρκεια του πρώτου περάσματος υπολογίζοντα οι μέσες τιμές των στοιχείων των δύο υποπινάκων και στη συνέχεια χρησιμοποιούνται ως άξονες για τις περαιτέρω υποδιαιρέσεις των υποπινάκων. Να σχεδιασθεί λεπτομερώς ο αλγόριθμος. Να βρεθούν τα πλεονεκτήματα και τα μειονεκτήματα της μεθόδου σε σχέση με τη γρήγορη ταξινόμηση, ως προς το πλήθος των περασμάτων και των ανταλλαγών. Να αποδειχθεί ότι και η παραλλαγή αυτή είναι $O(n^2)$. (Hint: αρκεί να βρεθεί μία κατάλληλη διάταξη θεωρώντας ένα μικρό πίνακα).

16. Να σχεδιασθούν και να αναλυθούν επαναληπτικοί και αναδρομικοί αλγόριθμοι ταξινόμησης με **συγχώνευση 3 δρόμων** (3-way merge). Δηλαδή, ένας πίνακας n θέσεων θα πρέπει να διαιρεθεί σε τρία τμήματα μεγέθους $\lfloor n/3 \rfloor$, $\lfloor (n+1)/3 \rfloor$ και $\lfloor (n+2)/3 \rfloor$, αυτά με τη σειρά τους να ταξινομηθούν και στη συνέχεια να γίνει η συγχώνευση.

17. Δεδομένου πίνακα μεγέθους n, γενικεύοντας την προηγούμενη άσκηση να σχεδιασθούν και να αναλυθούν επαναληπτικοί και αναδρομικοί αλγόριθμοι ταξινόμησης με συγχώνευση $\sqrt{n}$ δρόμων.

18. Ως γνωστό η απλή συγχώνευση δύο ταξινομημένων πινάκων `A[0..n-1]` και `B[0..m-1]` έχει πολυπλοκότητα $O(n+m)$. Η πολυπλοκότητα αυτή

μπορεί να βελτιωθεί αν $n \gg m$. Για παράδειγμα, αν m=1, τότε το στοιχείο αυτό μπορεί να καταλάβει τη σωστή θέση μεταξύ των n στοιχείων με δυαδική αναζήτηση. Για το λόγο αυτό, η επόμενη τεχνική ονομάζεται **δυαδική συγχώνευση** (binary merging).

Σύμφωνα, λοιπόν, με τη μέθοδο αυτή ο πίνακας `A` υποδιαιρείται σε m+1 διαδοχικούς υποπίνακες. Αν το τελευταίο στοιχείο του πίνακα `B` είναι μικρότερο από το τελευταίο στοιχείο του προτελευταίου υποπίνακα του πίνακα `A`, τότε το στοιχείο αυτό μαζί με τα στοιχεία του τελευταίου υποπίνακα αποθηκεύονται στις τελευταίες θέσεις του πίνακα εξόδου. Σε αντίθετη περίπτωση βρίσκεται η θέση του m-οστού στοιχείου μεταξύ των στοιχείων του τελευταίου υποπίνακα, οπότε και πάλι το αποτέλεσμα αποθηκεύεται στον πίνακα εξόδου. Έτσι η μέθοδος συνεχίζει αναδρομικά μέχρι την εξάντληση και των δύο πινάκων `A` και `B`. Να σχεδιασθεί και να αναλυθεί η μέθοδος.

19. Να σχεδιασθεί και να αναλυθεί η μέθοδος της **ευθείας εισαγωγής δύο δρόμων** (two way straight insertion), σύμφωνα με την οποία ένας πίνακας με n κλειδιά ταξινομείται τοποθετώντας τα κλειδιά σε έναν άλλον πίνακα μεγέθους $2n$+1 θέσεων. Η μέθοδος τοποθετεί το πρώτο κλειδί ως μεσαίο στοιχείο του πίνακα εξόδου. Τα υπόλοιπα κλειδιά τοποθετούνται αριστερά ή δεξιά του μεσαίου στοιχείου ανάλογα με το μέγεθος του κλειδιού. Κατά τη διάρκεια της τοποθέτησης γίνονται οι απαραίτητες μετακινήσεις. Για παράδειγμα, αν τα κλειδιά είναι στον πίνακα εισόδου με τη σειρά 52, 12, 71, 56, 5, 10, 19, 90 και 45, τότε τοποθετούνται στον πίνακα εξόδου με τη μορφή παρουσιάζεται στον Πίνακα 9.4.

4	5	6	7	8	9	10	11	12	13	14
						52				
					12	52				
					12	52	71			
					12	52	56	71		
				5	12	52	56	71		
			5	10	12	52	56	71		
		5	10	12	19	52	56	71		
		5	10	12	19	52	56	71	90	
	5	10	12	19	45	52	56	71	90	

Πίνακας 9.4: Το πρόβλημα της ευθείας εισαγωγής δύο δρόμων.

20. Η μέθοδος της **δυαδικής εισαγωγής δύο δρόμων** (two way binary insertion) είναι παρόμοια προς την ευθεία εισαγωγή δύο δρόμων της Άσκησης 19, αλλά η παρεμβολή γίνεται με δυαδικό τρόπο. Να σχεδιασθεί και να αναλυθεί η μέθοδος.

21. Σύμφωνα με τη μέθοδο της **ταξινόμησης περιττής-άρτιας μετάθεσης** (odd-even transposition sort) στα περάσματα περιττής (άρτιας) τάξης συγκρίνονται τα στοιχεία `A[i]` και `A[i+1]`, όπου το `i` είναι περιττός (αντίστοιχα, άρτιος) αριθμός. Αν ισχύει `A[i]>A[i+1]`, τότε τα στοιχεία ανταλλάσσονται. Τα περάσματα επαναλαμβάνονται μέχρις ότου δεν συμβούν άλλες ανταλλαγές. Να σχεδιασθεί και να αναλυθεί αντίστοιχος αλγόριθμος.

22. Η ιδανικότερη μέθοδος για την ταξινόμηση πινάκων με στοιχεία που είναι περίπου ταξινομημένα είναι η εξής. Αρχικά ελέγχονται τα στοιχεία του πίνακα εισόδου για τον εντοπισμό ζευγών διαδοχικών κλειδιών που δεν ακολουθούν τη σχέση διάταξης. Το ζεύγος αυτό διαγράφεται από τον πίνακα εισόδου και αποθηκεύεται σε έναν πίνακα εξόδου. Μετά την αφαίρεση κάποιου ζεύγους, η διαδικασία συνεχίζεται με τη σύγκριση του στοιχείου αμέσως πριν το πρώτο του ζεύγους με το στοιχείο αμέσως μετά το δεύτερο του ζεύγους. Μετά την απομάκρυνση κάποιων ζευγών ο πίνακας εισόδου είναι πλέον ταξινομημένος. Στη συνέχεια ο πίνακας εξόδου ταξινομείται χρησιμοποιώντας τη γρήγορη ταξινόμηση, οπότε οι δύο ταξινομημένοι πίνακες συγχωνεύονται με μία απλή διαδικασία συγχώνευσης. Να σχεδιασθεί και να αναλυθεί η μέθοδος.

23. Η ακόλουθη διαδικασία της **κυκλικής ταξινόμησης** (cycle sort) είναι μία παραλλαγή που εντάσσεται στις ταξινομήσεις με μέτρημα (δες Κεφάλαιο 9.8). Υποτίθεται ότι ένα ιστόγραμμα (πίνακας `H`) διατηρεί τις συχνότητες εμφάνισης όλων των τιμών των κλειδιών που βρίσκονται στο διάστημα $1..m$. Ο αλγόριθμος να δοκιμασθεί για διαφορετικές τιμές του m και να υπολογισθεί η πολυπλοκότητα χρόνου και χώρου. Πότε η μέθοδος είναι αποτελεσματική;

```
     procedure cycle_sort;
1.   H[1] <-- 0;
2.   for k <-- 1 to m-1 do H[k+1] <-- H[k]+h[k];
3.   for i <-- 1 to n do
4.      k <-- A[i]; j <-- H[k]+1;
5.      if i>=j then
6.         if i<>j then
```

```
7.            temp <-- A[i];
8.            repeat
9.               swap(A[j],temp);
10.              H[k] <-- j; k <-- temp; j <-- H[k]+1;
11.           until i=j;
12.           A[i] <-- temp;
13.        H[k] <-- I;
```

24. Η μέθοδος **ταξινόμησης με ανταλλαγή ρίζας** (radix exchange sort) εξετάζει τα ψηφία από τα αριστερά προς τα δεξιά. Η μέθοδος θυμίζει τη γρήγορη ταξινόμηση επειδή αρχικά τα κλειδιά που έχουν ως πρώτο ψηφίο το Ο τοποθετούνται πριν τα κλειδιά που έχουν ως πρώτο ψηφίο το 1. Κατόπιν η επεξεργασία συνεχίζεται σε κάθε ένα από τα δύο αυτά υποσύνολα αναδρομικά για τα επόμενα ψηφία κατά τον ίδιο τρόπο.

    Η επόμενη διαδικασία `radix_exchange` θεωρεί ότι τα κλειδιά είναι θετικοί ακέραιοι. Ο πίνακας σαρώνεται από αριστερά και από δεξιά με τη βοήθεια δύο δεικτών, `i` και `j`, αναζητώντας κλειδιά που να αρχίζουν από 1 και 0, αντιστοίχως. Όταν τέτοια κλειδιά εντοπισθούν, τότε ανταλλάσσονται αμοιβαία. Η διαδικασία σάρωσης συνεχίζεται μέχρι να συναντηθούν οι δύο δείκτες. Η ταξινόμηση εκτελείται με την κλήση `radix_exchange(1,n,15)`. Ο τελεστής `shr` εκτελεί διολίσθηση προς δεξιά, ενώ το όρισμα `b` χρησιμεύει για τον έλεγχο του αντίστοιχου ψηφίου ξεκινώντας από την τιμή 15 (για το αριστερότερο ψηφίο) και φθάνοντας μέχρι και την τιμή 0 (για το δεξιότερο ψηφίο).

```
procedure radix_exchange(left,right,b);
if (right>left) and (b>=0) then
   i <-- left; j <-- right;
   repeat
      while (A[i] shr (b-1) mod 2=0) and (i<j) do i <-- i+1;
      while (A[j] shr (b-1) mod 2=1) and (i<j) do j <-- j-1;
      Swap(A[i],A[j]);
   until j <-- i;
   if (A[r] shr (b-1) mod 2=0) then j <-- j+1;
   radix_exchange(left,j-1,b-1); radix_exchange(j,right,b-1)
```

25. Η **ταξινόμηση γραμμικής εξέτασης** (linear probing sort) συνδυάζει στοιχεία της αναζήτησης παρεμβολής και του ταξινομημένου κατακερματισμού

με γραμμική εξέταση. Εφαρμόζεται όταν είναι γνωστό το εύρος των τιμών `m<n` των κλειδιών του πίνακα `A[0..n-1]`, ενώ επίσης θα πρέπει να θεωρηθεί ένας συμπληρωματικός πίνακας `n+w` θέσεων, όπου οι w θέσεις χρησιμεύουν ως περιοχή υπερχείλισης. Κάθε στοιχείο του αρχικού πίνακα τοποθετείται σε κενή θέση του βοηθητικού πίνακα εφαρμόζοντας την αναζήτησης παρεμβολής. Αν η θέση του βοηθητικού πίνακα είναι κατειλημμένη από άλλο κλειδί, τότε η θέση αυτή καταλαμβάνεται τελικά από το μικρότερο κλειδί, ενώ το μεγαλύτερο τοποθετείται στην επόμενη θέση του βοηθητικού πίνακα. Μετά την εισαγωγή όλων των στοιχείων στο βοηθητικό πίνακα, με ένα απλό πέρασμα τα στοιχεία τοποθετούνται και πάλι στον αρχικό πίνακα. Να σχεδιασθεί η μέθοδος, να δοκιμασθεί πειραματικά και να αναλυθεί η επίδοσή της.

26. Αν στον αλγόριθμο `Select` το δείγμα αλλάξει από 5 σε 7 ή 3 τότε ο αλγόριθμος θα συνεχίσει να έχει γραμμική πολυπλοκότητα στη χειρότερη περίπτωση;

27. Να δείξετε πως ο αλγόριθμος ταξινόμησης `Quicksort` μπορεί να αλλάξει ώστε η πολυπλοκότητά του να είναι $O(n \log n)$ στην χειρότερη περίπτωση.

28. Έστω ότι $X[1 \ldots n]$ και $Y[1 \ldots n]$ δύο ταξινομημένοι πίνακες. Να δώσετε έναν αλγόριθμο με $O(\log n)$ πολυπλοκότητα για την εύρεση του μεσαίου στοιχείου των $2n$ στοιχείων των πινάκων X και Y.

10

Τυχαίοι Αλγόριθμοι

Περιεχόμενα Κεφαλαίου

Οι πιθανότητες είναι μέρος της καθημερινότητάς μας. Για παράδειγμα, πόσο πιθανό είναι να υπάρχουν δύο άνθρωποι στη Νέα Υόρκη με τον ίδιο αριθμών τριχών; Με βάση την **αρχή των περιστερώνων** (pigeonhole principle)[1] αποδεικνύεται ότι η πιθανότητα είναι μονάδα. Επίσης στο Κεφάλαιο 8.5 έχουμε μιλήσει για το παράδοξο των γενεθλίων. Αυτά είναι παραδείγματα όπου ένα γεγονός θα συμβεί με σημαντική πιθανότητα ή και βεβαιότητα. Αντιθέτως, η πιθανότητα να εκλεγεί πρόεδρος των ΗΠΑ με μία ψήφο διαφορά είναι 1 προς 10.000.000. Η πιθανότητα κατά τη διάρκεια ενός χρόνου κάποιος να σκοτωθεί από κεραυνό είναι 1 προς 2.500.000. Η πιθανότητα κατά τη διάρκεια ενός χρόνου ένας μεγάλος μετεωρίτης να πέσει στη γη είναι 1 προς 100.000. Αυτά είναι παραδείγματα όπου η πιθανότητα να συμβεί ένα γεγονός είναι σχεδόν μηδενική. Σε τέτοιες περιπτώσεις θεωρούμε την πιθανότητα σαν να είναι πράγματι μηδενική. Στη λογική αυτή στηρίζονται οι λεγόμενοι **τυχαίοι** ή **τυχαιοποιημένοι** (randomized) αλγόριθμοι, των οποίων η συμπεριφορά δεν εξαρτάται μόνο από τα δεδομένα εισόδου αλλά και από τιμές που παράγονται από μία γεννήτρια τυχαίων αριθμών.

Στον αντίποδα των τυχαίων αλγορίθμων βρίσκονται οι λεγόμενοι **αιτιοκρατικοί** (deterministic) αλγόριθμοι, όπου η πολυπλοκότητα της χειρότερης περίπτωσης αποδεικνύεται με βεβαιότητα (δηλαδή, πιθανότητα 1). Σε μία τέτοια περίπτωση, βέβαια, υπεισέρχεται η πιθανότητα το πρόγραμμα να έχει λάθος ή να συμβεί οποιοδήποτε άλλο αρνητικό γεγονός (όπως βλάβη του υλικού, πτώση ρεύματος, κλπ). Έτσι είναι προτιμότερο το πρόγραμμά μας να είναι απλούστερο (άρα χωρίς λάθη) στηριζόμενοι με υπολογισμένο ρίσκο στο γεγονός ότι η χειρότερη περίπτωση θα συμβεί με ελάχιστη πιθανότητα.

10.1 Κατηγορίες Τυχαίων Αλγορίθμων

Διακρίνουμε τρεις κατηγορίες τυχαίων αλγορίθμων, τους αλγορίθμους τύπου Las Vegas, τύπου Monte Carlo και τύπου Sherwood. Οι αλγόριθμοι Las Vegas πάντοτε δίνουν την ίδια (σωστή) έξοδο για τα ίδια δεδομένα εισόδου, απλώς ο χρόνος εκτέλεσης εξαρτάται από την έξοδο της γεννήτριας των τυχαίων αριθμών. Απεναντίας οι αλγόριθμοι Monte Carlo απαιτούν το ίδιο χρόνο εκτέλεσης ανεξαρτήτως της εξόδου της γεννήτριας των τυχαίων αριθμών, αλλά δεν δίνουν πάντοτε την ίδια έξοδο για τα ίδια δεδομένα εισόδου. Προφανώς, η ιδιότητα αυτή δεν είναι επιθυμητή γιατί άλλοτε ο αλγόριθμος θα δίνει τη σωστή λύση και άλλοτε κάποια λανθασμένη.

Λέγεται λοιπόν ότι ένα αλγόριθμος Las Vegas (Monte Carlo) αποτυγχάνει

[1] Αν τοποθετηθούν $n > m$ περιστέρια σε m περιστερότρυπες, τότε υπάρχει τουλάχιστον μία περιστερότρυπα με περισσότερο από ένα περιστέρι.

αν δεν τερματίζει εντός ενός συγκεκριμένου χρονικού διαστήματος (αν δεν δίνει τη σωστή απάντηση). Η απαίτησή μας είναι οι αντίστοιχοι αλγόριθμοι να αποτυγχάνουν με κάποια πολύ μικρή πιθανότητα, δηλαδή n^{-a}, για δεδομένη τιμή της παραμέτρου a.

Τέλος, η κατηγορία αλγορίθμων Sherwood περιλαμβάνει αλγορίθμους που πάντοτε δίνουν απάντηση και η απάντηση αυτή είναι σωστή. Συχνά σε αιτιοκρατικούς αλγορίθμους η χειρότερη περίπτωση είναι σημαντικά χειρότερη από τη μέση περίπτωση. Ενσωματώνοντας, λοιπόν, την τυχαία επιλογή μίας γεννήτριας (με μηδαμινό κόστος) είναι δυνατόν να εξαλειφθεί η διαφορά μεταξύ της χειρότερης και της μέσης περίπτωσης. Καθώς μία τέτοια τεχνική βοηθά σε μία περισσότερο ομαλή εκτέλεση του αιτιοκρατικού αλγορίθμου, δεν τίθεται ζήτημα θεώρησης της κατανομής της πιθανότητας ο αλγόριθμος να μην τερματίσει ή να δώσει λανθασμένα αποτελέσματα.

Αυτή είναι η βάση των τυχαίων αλγορίθμων, των οποίων η ανάλυση είναι μία εναλλακτική οπτική γωνία στην ανάλυσης της μέσης περίπτωσης, όπου για την ευκολία μας μπορεί να υιοθετήσουμε περιοριστικές υποθέσεις. Οι τυχαίοι αλγόριθμοι έχουν καλύτερη και χειρότερη περίπτωση αλλά τα δεδομένα εισόδου που προκαλούν τις περιπτώσεις αυτές είναι άγνωστα και μπορεί να είναι οποιαδήποτε ακολουθία μεγέθους n.

10.2 Εξακρίβωση Επαναλαμβανόμενου Στοιχείου

Έστω ότι ένας πίνακας με n στοιχεία περιέχει $n/2+1$ διακριτά στοιχεία, όπου τα $n/2$ έχουν την ίδια τιμή. Το ζητούμενο είναι να βρούμε ποιά είναι η τιμή που εμφανίζεται $n/2$ φορές. Είναι εύκολο να σχεδιάσουμε έναν αιτιοκρατικό αλγόριθμο που να σαρώνει τον πίνακα και να διαπιστώνει ποιά είναι αυτή η τιμή μετά από $n/2+2 = \mathrm{O}(n)$ βήματα στη χειρότερη περίπτωση. Αυτή η περίπτωση ανταποκρίνεται στην κατάσταση όπου οι πρώτες $n/2+1$ θέσεις του πίνακα καταλαμβάνονται από αντίστοιχες διακριτές τιμές, ενώ η $(n/2+2)$-οστή είναι η πρώτη επανάληψη της πολλαπλής τιμής.

Τώρα θα εξετάσουμε έναν τυχαίο αλγόριθμο, δηλαδή έναν αλγόριθμο που στηρίζεται σε μία γεννήτρια τυχαίων αριθμών. Έστω, λοιπόν, ότι επιλέγουμε με τυχαίο τρόπο δύο διαφορετικές θέσεις του πίνακα και ελέγχουμε αν το περιεχόμενό τους είναι ίδιο. Αν πράγματι το περιεχόμενο είναι ίδιο, τότε αλγόριθμος τερματίζει, αλλιώς πρέπει το δειγματοληπτικό πείραμα να συνεχισθεί για όσο χρειασθεί. Προφανώς, ο αλγόριθμος αυτός ανήκει στην κατηγορία Las Vegas, δηλαδή ο αλγόριθμος είναι βέβαιο ότι τερματίζοντας δίνει τη σωστή απάντηση αλλά δεν είναι σταθερός ο απαιτούμενος χρόνος. Η επόμενη διαδικασία παρουσιάζει

αυτήν την περιγραφή.

```
     procedure repeatedelement
1.   flag <-- true;
2.   while flag do
3.      i <-- random(1,n); j <-- random(1,n)
4.      if (i<>j) and (A[i]=A[j]) then
5.         flag <-- false; return A[i]
```

Η πιθανότητα ότι μία δοκιμή θα είναι επιτυχής, δηλαδή θα εξακριβώσει τη ζητούμενη τιμή ισούται με $p = \frac{n/2}{n}\,\frac{n/2-1}{n} = \frac{n/2-1}{2n}$. Διαπιστώνεται, δηλαδή, ότι για $n = 10$, ισχύει $p = 0.2$, άρα σε μία τέτοια περίπτωση πρέπει να επαναλάβουμε τη δοκιμή με πιθανότητα 0.8. Η πιθανότητα ο αλγόριθμος να μην τερματίσει μετά από 10 δοκιμές είναι $(1-p)^{10} \approx 0.11$ και άρα η πιθανότητα να τερματίσει είναι 0.89 περίπου. Αντίστοιχα, η πιθανότητα να μην τερματίσει μετά από 100 δοκιμές είναι $(1-p)^{100} \approx 2 \times 10^{-10}$.

Στη γενική περίπτωση, η πιθανότητα να τερματίσει ο αλγόριθμος μετά από $ca \log n$ βήματα (για κάποια σταθερά c) είναι

$$(4/5)^{ca \log n} \;=\; n^{-ca \log(5/4)}$$

που δίνει n^{-a} αν $c = 1/\log(5/4)$. Συνεπώς, με πιθανότητα μεγαλύτερη από $1 - n^{-a}$, ο αλγόριθμος θα τερματίσει στη χειρότερη περίπτωση μετά από $\frac{a \log n}{\log(5/4)}$ πειράματα. Έτσι, θεωρώντας ως βαρόμετρο την εντολή 5, προκύπτει ότι η πολυπλοκότητα του τυχαίου αυτού αλγορίθμου είναι $O(\log n)$.

10.3 Εξακρίβωση Πλειοψηφικού Στοιχείου

Στην παράγραφο αυτή θα εξετάσουμε ένα παρεμφερές με το προηγούμενο πρόβλημα. Έστω ότι δίνεται ένας πίνακας με n στοιχεία. Λέγεται ότι ο πίνακας αυτός έχει ένα **πλειοψηφικό** (majority) στοιχείο, αν ένα στοιχείο του εμφανίζεται περισσότερο από $n/2$ φορές. Είναι όμως ενδεχόμενο να μην υπάρχει πλειοψηφικό στοιχείο στον πίνακα. Ας μελετήσουμε τον επόμενο αλγόριθμο.

```
     function maj
1.   i <-- random(1,n); x <-- A[i]; k <-- 0;
2.   for j <-- 1 to n do
3.      if A[j]=x then k <-- k+1
4.   return (k>n/2)
```

Αν ο πίνακας δεν έχει πλειοψηφικό στοιχείο, τότε ο αλγόριθμος `maj` ορθώς θα επιστρέψει `false`. Ωστόσο, αν ο πίνακας έχει πλειοψηφικό στοιχείο, τότε ο αλγόριθμος θα επιστρέψει `true` αλλά το αποτέλεσμα θα είναι ορθό με πιθανότητα $p > 1/2$. Όμως λάθος ίσο με 50% δεν μπορεί να είναι αποδεκτό. Προχωρούμε σε μία δεύτερη εκδοχή.

```
     function maj2
1.   if maj then return true
2.   else return maj
```

Αν ο πίνακας δεν έχει πλειοψηφικό στοιχείο, τότε ο αλγόριθμος `maj2` ορθώς θα επιστρέψει `false`, τιμή που ορθώς θα του περάσει ο αλγόριθμος `maj`. Από την άλλη πλευρά, αν ο πίνακας πράγματι έχει πλειοψηφικό στοιχείο και ορθώς ο αλγόριθμος `maj` επιστρέψει τιμή `true`, τότε και αλγόριθμος `maj2` θα επιστρέψει `true`. Αν όμως με πιθανότητα $p > 1/2$ ο αλγόριθμος `maj` επιστρέψει τιμή `false`, τότε ο αλγόριθμος `maj2` θα ξανακαλέσει τον αλγόριθμο `maj` που θα επιστρέψει και πάλι `true` ή `false` πιθανότητα $p > 1/2$ και $1-p$, αντίστοιχα. Στην περίπτωση αυτή θα λάβουμε ορθό αποτέλεσμα με πιθανότητα $p+(1-p)p > 3/4$. Επομένως επιτύχαμε σημαντική βελτίωση στην αξιοπιστία του αρχικού αλγορίθμου.

Η τελική μας εκδοχή είναι ο επόμενος αλγόριθμος Monte Carlo. Υποθέτουμε ότι $\epsilon > 0$ είναι η μέγιστη ανοχή της πιθανότητας λάθους. Είναι προφανές ότι η πολυπλοκότητα του αλγορίθμου είναι $O(n \log(1/\epsilon))$. Ο αλγόριθμος αυτός έχει μόνο διδακτική σπουδαιότητα καθώς είναι εύκολο να σχεδιάσουμε έναν αιτιοκρατικό αλγόριθμο με γραμμική πολυπλοκότητα.

```
     function majMC
1.   k <-- log(1/e)
2.   for j <-- 1 to k do
3.      if maj then return true
4.   return false
```

10.4 Αναζήτηση σε Διατεταγμένη Λίστα

Γνωρίζουμε από το μάθημα των Δομών Δεδομένων ότι μπορούμε να αναπαραστήσουμε μία δυναμική λίστα με τη βοήθεια ενός πίνακα. Ας μελετήσουμε την τεχνική αυτή με ένα παράδειγμα. Στο Σχήμα 10.1 παρουσιάζεται ένας δισδιάστατος πίνακας `A` με τα γνωστά 9 κλειδιά 52, 12, 71, 56, 5, 10, 19, 90 και 45, τα οποία εμφανίζονται στην πρώτη γραμμή του πίνακα. Στη δεύτερη γραμμή παρουσιάζονται κάποιες

τιμές που προσομοιώνουν τον κλασικό δείκτη των δυναμικών συνδεδεμένων λιστών. Ακολουθώντας αυτούς τους τεχνητούς δείκτες μπορούμε να πάρουμε τα στοιχεία σε αύξουσα διάταξη. Για να γίνει δυνατή η προσπέλαση και η επεξεργασία της δομής είναι απαραίτητο να έχουμε και μία βοηθητική μεταβλητή `header` που να δίνει τη θέση του μικρότερου στοιχείου της δομής. Στην περίπτωση μας ισχύει `header=5`. Επίσης παρατηρούμε ότι ο δείκτης του μεγαλύτερου στοιχείου είναι 0 ώστε να διευκολύνει το πέρας της προσπέλασης.

52	12	71	56	5	10	19	90	45
4	7	8	3	6	2	9	0	1

Πίνακας 10.1: Διατεταγμένη λίστα.

Η επόμενη διαδικασία αναζητά το κλειδί `key` που υπάρχει πράγματι στη δομή και επιστρέφει τη θέση του κλειδιού στη δομή. Είναι εύκολο να εξαχθεί η επίδοση της αναζήτησης στη δομή αυτή. Προφανώς στη μέση (χειρότερη) περίπτωση θα πρέπει να σαρωθεί ο μισός (ολόκληρος ο) πίνακας, και επομένως η αναζήτηση είναι γραμμική $\Theta(n)$.

```
     function search(key, header)
1.   i <-- header
2.   while key > A[1,i] do i <-- A[2,i]
3.   return i
```

Μία τυχαιοποιημένη εκδοχή έχει ως εξής.

```
     function searchrandom(key)
1.   i <-- random(1,n); y <-- A[1,i]
2.   if key<y then return search(key,header)
3.   else if key>y then return search(key,A[2,i])
4.   else return i
```

Εύκολα και πάλι προκύπτει ότι η νέα τυχαιοποιημένη συνάρτηση είναι γραμμικής πολυπλοκότητας. Ωστόσο, πληρώνοντας το αμελητέο κόστος της εντολής 1, η διαδικασία επιταχύνεται κατά 50% κατά μέσο όρο. Η συνάρτηση αυτή στηρίζεται στη φιλοσοφία Sherwood.

10.5 Διαγραφή σε Δυαδικό Δένδρο Αναζήτησης

Όπως είναι γνωστό από το μάθημα των Δομών Δεδομένων, η διαδικασία της διαγραφής σε δυαδικό δένδρο αναζήτησης είναι πιο σύνθετη από τη διαδικασία της εισαγωγής. Αν ο κόμβος που πρόκειται να διαγραφεί είναι τερματικός, τότε η περίπτωση είναι εύκολη. Επίσης σχετικά εύκολη είναι η περίπτωση, όπου ο διαγραφόμενος κόμβος έχει μόνο έναν απόγονο. Η δυσκολία του προβλήματος παρουσιάζεται όταν το στοιχείο που θα διαγραφεί έχει δύο απογόνους. Στην περίπτωση αυτή το διαγραφόμενο κλειδί αντικαθίσταται από το κλειδί του πιο δεξιού κόμβου του αριστερού υποδένδρου, ο οποίος (όπως μπορεί να αποδειχθεί) έχει το πολύ έναν απόγονο.

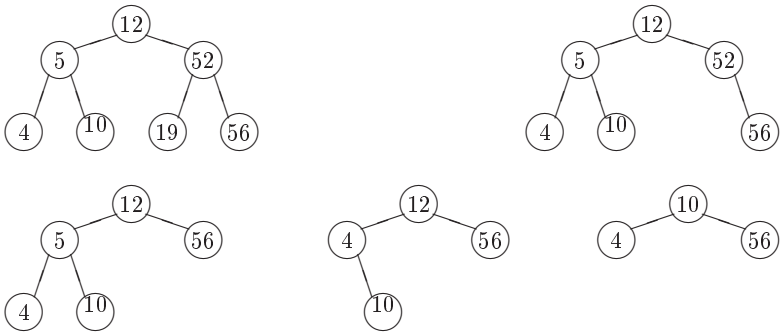
Στη συνέχεια δίνεται η αναδρομική διαδικασία `delete` που καλεί την επίσης αναδρομική διαδικασία `del`, όταν ο κόμβος με το κλειδί `key` έχει δύο απογόνους. Έτσι ανιχνεύεται ο προηγούμενος λεξικογραφικά κόμβος, δηλαδή ο δεξιότερος κόμβος του αριστερού υποδένδρου του κόμβου `q` που πρόκειται να διαγραφεί. Αν ο κόμβος αυτός είναι ο `r`, τότε το περιεχόμενό του αντικαθιστά το περιεχόμενο του `q` και ο χώρος του `r` επιστρέφεται στο σύστημα. Για την κατανόηση αναφέρεται ότι θεωρείται κόμβος με τρία πεδία (`data,left,right`), ακέραιου τύπου και τύπου δείκτη.

```
     procedure del(r)
1.   if r.right<>nil then del(r.right)
2.   else
3.      q.data <-- r.data; q <-- r; r <-- r.left

     procedure delete(key,p);
1.   if p=nil then write('Το κλειδί ',key,'δεν υπάρχει')
2.   else if key<p.data then delete(key,p.left)
3.   else if key>p.data then delete(key,p.right)
4.   else
5.      q:=p;
6.      if q.right=nil then p:=q.left
7.      else if q.left=nil then p:=q.right
8.      else del(q.left);
9.      dispose(q)
```

Για παράδειγμα στο επόμενο σχήμα δίνεται η διαδικασία της διαδοχικής διαγραφής των κλειδιών 19, 52, 5 και 12 από το αρχικό δένδρο.

Ο τρόπος υλοποίησης της διαδικασίας της διαγραφής δεν είναι ο μοναδικός. Για παράδειγμα, αν το υπό διαγραφή κλειδί έχει και αριστερό αλλά και δεξιό



Σχήμα 10.1: Διαγραφή κόμβων από δένδρο αναζήτησης.

υποδένδρο, τότε αυτό ισοδύναμα θα μπορούσε να αντικατασταθεί από τον επόμενο λεξικογραφικά κόμβο, δηλαδή τον αριστερότερο κόμβο του δεξιού υποδένδρου. Αυτές οι δύο εναλλακτικές υλοποιήσεις ονομάζονται **ασύμμετρες** (asymmetric), σε αντίθεση με τη μέθοδο της **συμμετρικής** (symmetric) διαγραφής, σύμφωνα με την οποία το υπό διαγραφή κλειδί αντικαθίσταται με τον προηγούμενο ή τον επόμενο λεξικογραφικά κόμβο είτε εναλλάξ είτε με τυχαίο τρόπο (δηλαδή, με 50%-50% πιθανότητα να χρησιμοποιηθεί ο αριστερός ή ο δεξιός υποψήφιος). Η υλοποίηση της συμμετρικής τυχαίας διαγραφής από δυαδικό δένδρο αναζήτησης είναι εύκολη υπόθεση για τον αναγνώστη.

Έχει αποδειχθεί ότι αν από τυχαίο δένδρο με n κόμβους διαγραφεί ένα κλειδί με την ασύμμετρη μέθοδο, τότε δεν προκύπτει ένα τυχαίο δένδρο. Πιο συγκεκριμένα, το μήκος εσωτερικού μονοπατιού δεν παραμένει τάξης $O(n \log n)$ αλλά αυξάνει[2]. Εμπειρικές μελέτες από τον Eppinger (1983) έδειξαν ότι αν χρησιμοποιηθεί η ασύμμετρη μέθοδος διαγραφής, τότε το μήκος εσωτερικού μονοπατιού γίνεται τάξης $O(n \log^3 n)$. Αντίθετα αν χρησιμοποιηθεί η συμμετρική μέθοδος διαγραφών, τότε το μήκος εσωτερικού μονοπατιού πράγματι παραμένει $O(n \log n)$. Συνεπώς η τεχνική της τυχαιοποιημένης διαγραφής δεν είναι χωρίς επίπτωση στη συνολική επίδοση της δομής κατά τις αναζητήσεις.

Πίσω από την συμμετρική διαγραφή βρίσκεται η φιλοσοφία των αλγορίθμων Sherwood. Στα επόμενα δύο παραδείγματα θα ασχοληθούμε με κλασικότερα παράδειγμα τυχαίων αλγορίθμων που ανήκουν σε μία από τις δύο άλλες κατηγορίες που αναφέρθηκαν, δηλαδή τις κατηγορίες Las Vegas ή Monte Carlo.

[2]Στο επόμενο κεφάλαιο θα αναλυθούν οι έννοιες του τυχαίου δένδρου και του μήκους εσωτερικού μονοπατιού.

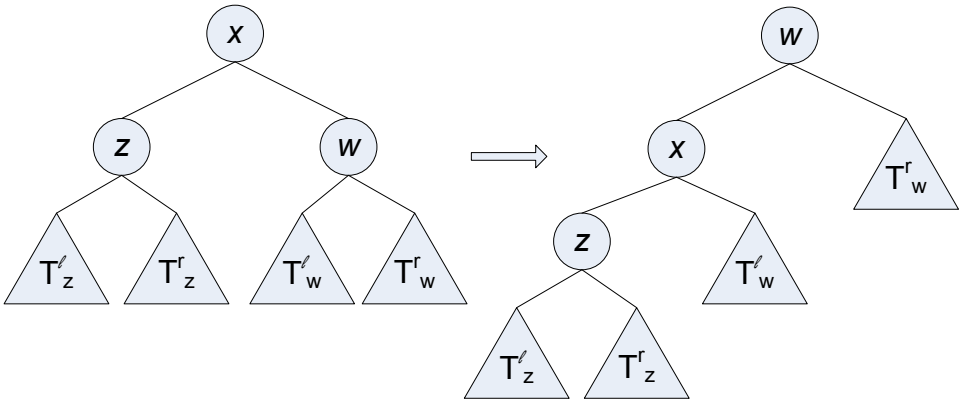
10.6 Τυχαία Δυαδικά Δένδρα

Στο σημείο αυτό θα ασχοληθούμε με δυαδικά δένδρα αναζήτησης, τα οποία βασίζονται στις τυχαίες επιλογές που κάνει ο αλγόριθμος κατά την κατασκευή τους. Τα συγκεκριμένα τυχαία δένδρα ονομάζονται **Δένδρα Αναζήτησης Σωρού** (ΔΑΣ για συντομία - treaps) και παρουσιάζονται αναλυτικά στο [145].

Έστω X ένα σύνολο με n στοιχεία, όπου το καθένα απαρτίζεται από ένα *κλειδί* και μία *προτεραιότητα*. Ένα ΔΑΣ για το X είναι ένα δυαδικό δένδρο με n κόμβους, το οποίο είναι δομημένο με ενδοδιάταξη ως προς τα κλειδιά και με διάταξη σωρού ως προς τις προτεραιότητες (παρόμοιο με δομές που ονομάζονται Δένδρο Προτεραιότητας [106] και Καρτεσιανό Δένδρο [163]). Η ενδοδιάταξη σημαίνει ότι το κλειδί κάθε κόμβου είναι μεγαλύτερο από τα κλειδιά του αριστερού υποδένδρου του και μικρότερο από τα κλειδιά του δεξιού υποδένδρου του, ενώ η προτεραιότητα του είναι μεγαλύτερη από τις προτεραιότητες των κόμβων των δύο υποδένδρων του (δηλαδή, είναι σωρός μεγίστων ὐ δες Κεφάλαιο **??**). Αν όλα τα κλειδιά, καθώς και οι προτεραιότητες, είναι διακριτά, τότε το ΔΑΣ είναι μοναδικό. Έτσι, το αντικείμενο με τη μεγαλύτερη προτεραιότητα θα είναι στην ρίζα, ενώ ο διαχωρισμός των υπόλοιπων κλειδιών στο αριστερό και δεξιό υποδένδρο θα γίνει με βάση την ενδοδιάταξη των κλειδιών.

Έστω T το ΔΑΣ που αποθηκεύει το σύνολο X. Δοθέντος του κλειδιού ενός στοιχείου $x \in X$, το στοιχείο μπορεί να ευρεθεί στο T χρησιμοποιώντας τον κλασικό αλγόριθμο σε δυαδικά δένδρα αναζήτησης. Προφανώς, ο χρόνος εύρεσης ισούται με το βάθος του κόμβου όπου βρίσκεται το x στο δένδρο T. Η εισαγωγή ενός νέου στοιχείου z στο δένδρο T επιτυγχάνεται σε δύο φάσεις. Αρχικά, με βάση την τιμή του κλειδιού του z εισάγουμε το z στο κατάλληλο φύλλο του T και συνεπώς η ενδοδιάταξη του δένδρου διατηρείται. Σε μία δεύτερη φάση ελέγχεται αν ο πατέρας του z έχει μικρότερη προτεραιότητα από τον z, διότι τότε παραβιάζεται η διάταξη σωρού. Σε μία τέτοια περίπτωση, λοιπόν, για την αποκατάσταση της διάταξης σωρού μεταφέρεται το z προς τα επάνω με απλές περιστροφές.

Η διαγραφή του x γίνεται με αντίστοιχο τρόπο. Δηλαδή, πρώτα βρίσκουμε τον κόμβο που περιέχει το x και έπειτα με απλές περιστροφές τον μεταφέρουμε προς τα κάτω μέχρι να γίνει φύλλο, ώστε να τον διαγράφουμε στη συνέχεια. Η απλή δεξιά περιστροφή ενός κόμβου x με δεξιό παιδί τον z και αριστερό τον w και με δεξιό υποδένδρο του z το T_z^r και αριστερό υποδένδρο του w το T_w^ℓ, είναι μία πράξη τοπικής μεταβολής της δομής, έτσι ώστε ο z θα έχει δεξιό παιδί τον x και ο x αριστερό παιδί το υποδένδρο T_z^r. Στο Σχήμα 10.2 φαίνεται μία αριστερή περιστροφή. Προσοχή, διότι οι απλές περιστροφές γίνονται έτσι ώστε να ισχύει η διάταξη σωρού ως προς τις προτεραιότητες. Στο Σχήμα 10.3



Σχήμα 10.2: Μία απλή αριστερή περιστροφή.

φαίνονται επακριβώς οι πράξεις εισαγωγής και διαγραφής. Σε κάθε κόμβο του σχήματος, το επάνω μέρος είναι το κλειδί (λεξικογραφική σειρά) και το κάτω μέρος είναι η προτεραιότητα.

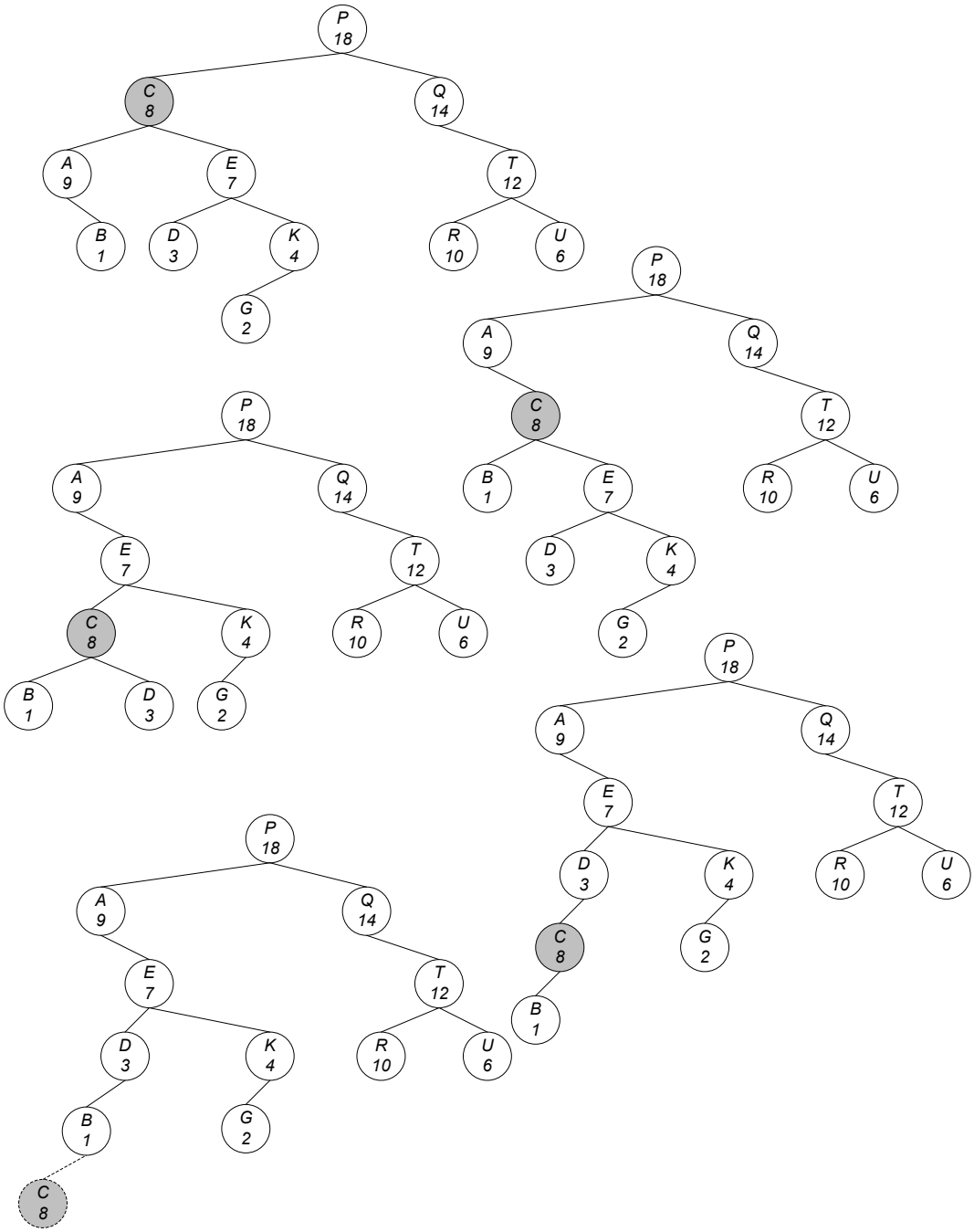
Οι προτεραιότητες στα ΔΑΣ εισάγουν την τυχαιότητα στην κατασκευή του δένδρου. Πιο συγκεκριμένα, κάθε φορά που ένα στοιχείο με κλειδί x εισάγεται στο ΔΑΣ T, παράγεται ένας τυχαίος αριθμός y και προσάπτεται στο στοιχείο ως προτεραιότητα.

Στη συνέχεια θα αναλύσουμε ένα σύνολο από ενδιαφέροντα μεγέθη στα ΔΑΣ, τα οποία θα μας επιτρέψουν να βρούμε τη χρονική πολυπλοκότητα των βασικών πράξεων. Αυτά τα μεγέθη είναι τα εξής:

$D(x)$**:** το βάθος ενός κόμβου x στο δένδρο, και

$SL(x), SR(x)$**:** το μήκος της δεξιάς ράχης του αριστερού υποδένδρου του x και το μήκος της αριστερής ράχης του δεξιού υποδένδρου του x. Η αριστερή (δεξιά) ράχη ενός δένδρου είναι το μονοπάτι από τη ρίζα προς το αριστερότερο (αντιστοίχως, δεξιότερο) φύλλο.

Έστω, λοιπόν, ένα ΔΑΣ T για το σύνολο X με n στοιχεία $x_i = (k_i, p_i)$ για $1 \leq i \leq n$, αριθμημένα με βάση το κλειδί κατά αύξουσα τάξη. Οι προτεραιότητες p_i είναι τυχαίες μεταβλητές. Αφού για καθορισμένο σύνολο κλειδιών k_i η μορφή του δένδρου T εξαρτάται από τις προτεραιότητες, τα προηγούμενα μεγέθη θα είναι και αυτά τυχαίες μεταβλητές και επομένως θα πρέπει να υπολογίσουμε μέσες τιμές. Η ανάλυση είναι πολύ απλή δεδομένου ότι κάθε ένα από τα μεγέθη αυτά μπορεί να αναπαρασταθεί με τη βοήθεια των εξής *δεικνυουσών μεταβλητών*:



Σχήμα 10.3: Διαγραφή (και αντιστρόφως, εισαγωγή) ενός στοιχείου με κλειδί C και προτεραιότητα 8 σε ένα ΔΑΣ.

- η $A_{i,j}$ είναι 1 αν το x_i είναι πρόγονος του x_j στο T, αλλιώς είναι 0.
- η $C_{i;\ell,m}$ είναι 1 αν το x_i είναι κοινός πρόγονος των x_ℓ και x_m, αλλιώς είναι 0.

Επίσης, θεωρούμε ότι κάθε κόμβος είναι πρόγονος του εαυτού του.

Λήμμα 10.1. *Έστω ότι $1 \leq \ell < m \leq n$. Τότε ισχύει:*

1. $D(x_\ell) = \sum_{i=1}^{n} A_{i,\ell}$
2. $SL(x_\ell) = \sum_{i=1}^{\ell-1} (A_{i,\ell-1} - C_{i;\ell-1,\ell})$
3. $SR(x_\ell) = \sum_{i=\ell+1}^{n} (A_{i,\ell+1} - C_{i;\ell,\ell+1})$

Απόδειξη. Η (1) είναι προφανής αφού το βάθος ενός κόμβου είναι ίσο με τον αριθμό των προγόνων του. Για τις (2) και (3) αρκεί να ασχοληθούμε με μία από τις δύο αφού είναι συμμετρικές (έστω τη (2)).

Αν η x_ℓ έχει αριστερό υποδένδρο, τότε ο χαμηλότερος κόμβος (ως προς το βάθος) στην δεξιά του υποδένδρου ράχη είναι το στοιχείο $x_{\ell-1}$. Είναι σαφές ότι σε αυτή την περίπτωση η ράχη αποτελείται από όλους τους προγόνους του $x_{\ell-1}$ εξαιρουμένων των προγόνων του x_ℓ. Όμως οι πρόγονοι του x_ℓ είναι επίσης πρόγονοι του $x_{\ell-1}$. Επίσης δεν υπάρχει κάποιο x_i, έτσι ώστε $i \geq \ell$, το οποίο να βρίσκεται επάνω σε αυτή τη ράχη. Αν η x_ℓ δεν έχει αριστερό υποδένδρο, τότε είτε $\ell = 1$, και σε αυτή την περίπτωση η (2) σωστά δίνει 0, ή το $x_{\ell-1}$ είναι πρόγονος του x_ℓ οπότε κάθε πρόγονος του $x_{\ell-1}$ είναι κοινός πρόγονος των $x_{\ell-1}$ και x_ℓ και η (2) δίνει σωστά 0. ∎

Έστω ότι $a_{i,j}$=E$[A_{i,j}]$ (Ε υποδηλώνει μέση τιμή) και $c_{i;\ell,m} =$ E$[C_{i;\ell,m}]$. Τότε από τη γραμμικότητα της μέσης τιμής προκύπτει το εξής συμπέρασμα.

Λήμμα 10.2. *Έστω ότι $1 \leq \ell < m \leq n$. Τότε ισχύει:*

1. $E[D(x_\ell)] = \sum_{i=1}^{n} a_{i,\ell}$
2. $E[SL(x_\ell)] = \sum_{i=1}^{\ell-1} (a_{i,\ell-1} - c_{i;\ell-1,\ell})$
3. $E[SR(x_\ell)] = \sum_{i=\ell+1}^{n} (a_{i,\ell+1} - c_{i;\ell,\ell+1})$

Στην ουσία η ανάλυση της πολυπλοκότητας έχει αναχθεί στην εύρεση των μέσων τιμών των $a_{i,j}$ και $c_{i;\ell,m}$. Αφού ασχολούμαστε με δεικνύουσες μεταβλητές έχουμε τα εξής:

$$a_{i,j} = \mathrm{E}[A_{i,j}] = Pr[A_{i,j} = 1] = Pr[x_i \text{ είναι πρόγονος του } x_j]$$

$$c_{i;\ell,j} = \mathrm{E}[C_{i;\ell,j}] = Pr[C_{i;\ell,j} = 1] = Pr[x_i \text{ είναι κοινός πρόγονος των } x_\ell \text{ και } x_m]$$

Ο καθορισμός αυτών των πιθανοτήτων και κατ' επέκταση των μέσων τιμών είναι δυνατός χρησιμοποιώντας το επόμενο *λήμμα προγόνων*.

Λήμμα 10.3. *Έστω ότι T είναι ένα ΔΑΣ στο X και έστω $1 \leq i, j \leq n$. Αν οι προτεραιότητες είναι διακριτές μεταξύ τους, τότε το x_i είναι πρόγονος του x_j στο T αν και μόνο αν για όλα τα x_h, όπου h μεταξύ των i και j, το στοιχείο x_i έχει τη μεγαλύτερη προτεραιότητα.*

Απόδειξη. Έστω ότι x_m είναι το στοιχείο με τη μεγαλύτερη προτεραιότητα στο T και έστω ότι $X' = \{x_v | 1 \leq v < m\}$ και $X'' = \{x_u | m < u \leq n\}$. Από τον ορισμό του ΔΑΣ, το x_m είναι η ρίζα του T και τα αριστερό και δεξιό υποδένδρο της θα είναι ΔΑΣ για τα X' και X'', αντιστοίχως.

Είναι προφανές ότι για οποιοδήποτε ζεύγος κόμβων που περιέχει τη ρίζα x_m, η σχέση ως προς το ποιός είναι πρόγονος είναι σωστή. Επιπλέον, η προγονική σχέση για οποιοδήποτε ζεύγος (x_v, x_u), $x_v \in X'$ και $x_u \in X''$ είναι προφανής αφού είναι σε διαφορετικά υποδένδρα και δεν έχουν κάποια σχέση μεταξύ τους. Όμως, στην περίπτωση αυτή η μέγιστη προτεραιότητα μεταξύ των v και u ανήκει στη ρίζα x_m. Το ίδιο αποδεικνύεται επαγωγικά για όλα τα ζεύγη που βρίσκονται εξ ολοκλήρου είτε στο X' είτε στο X''. ∎

Μία άμεση συνέπεια αυτού του λήμματος είναι ένα ανάλογο *λήμμα κοινών προγόνων*.

Λήμμα 10.4. *Έστω T ένα ΔΑΣ για το X και έστω $1 \leq \ell, m, i \leq n$ όπου $\ell < m$. Έστω ότι p_v είναι η προτεραιότητα του στοιχείου x_v. Τότε, αν οι προτεραιότητες είναι διακριτές μεταξύ τους, τότε το x_i είναι κοινός πρόγονος των x_ℓ και x_m στο T αν και μόνο αν:*

- $p_i = \max\{p_v | i \leq v \leq m\}$ *αν* $1 \leq i \leq \ell$
- $p_i = \max\{p_v | \ell \leq v \leq m\}$ *αν* $\ell \leq i \leq m$
- $p_i = \max\{p_v | \ell \leq v \leq i\}$ *αν* $m \leq i \leq n$

Τα δύο τελευταία λήμματα διευκολύνουν τον υπολογισμό των πιθανοτήτων $a_{i,j}$ και $c_{i,\ell,m}$.

Λήμμα 10.5. *Σε ένα ΔΑΣ, το x_i είναι πρόγονος του x_j με πιθανότητα $1/(|i-j|+1)$, δηλαδή:*

$$a_{i,j} = \frac{1}{(|i-j|+1}$$

Απόδειξη. Σύμφωνα με το λήμμα των προγόνων θα πρέπει η προτεραιότητα του x_i να είναι η μεγαλύτερη ανάμεσα σε όλα τα $|i-j|+1$ στοιχεία μεταξύ των x_i και x_j. Αφού αυτές οι προτεραιότητες είναι ανεξάρτητες και ίδιας κατανομής συνεχείς τυχαίες μεταβλητές, αυτό συμβαίνει με πιθανότητα $1/(|i-j|+1)$. ∎

Με αντίστοιχο τρόπο αποδεικνύεται το επόμενο συμπέρασμα.

Λήμμα 10.6. *Έστω ότι $1 \leq \ell < m \leq n$. Η μέση τιμή για κοινό πρόγονο είναι:*

$$c_{i;\ell,m} = \begin{cases} 1/(m-i+1) & \text{αν} \quad 1 \leq i \leq \ell \\ 1/(m-\ell+1) & \text{αν} \quad \ell \leq i \leq m \\ 1/(i-\ell+1) & \text{αν} \quad m \leq i \leq n \end{cases}$$

Από τη συζήτηση που προηγήθηκε μπορούμε να υπολογίσουμε τις μέσες τιμές των μεγεθών που μας ενδιαφέρουν. Ο υπολογισμός αυτός εμπεριέχει αρμονικούς αριθμούς (δες Κεφάλαιο 2.1).

Λήμμα 10.7. *Έστω $1 \leq \ell < m \leq n$. Σε ένα ΔΑΣ ισχύουν τα εξής:*

1. $E[D(x_\ell)] = H_\ell + H_{n+1-\ell} - 1 < 1 + 2\ln n$
2. $E[SL(x_\ell)] = 1 - \frac{1}{\ell}$
3. $E[SR(x_\ell)] = 1 - \frac{1}{n+1-\ell}$

Απόδειξη. Η απόδειξη προκύπτει με απλή αντικατάσταση των τιμών από τα Λήμματα 10.5 και 10.6 στο Λήμμα 10.2. ∎

Στη συνέχεια προχωρούμε στη ανάλυση των βασικών πράξεων σε ΔΑΣ. Μία επιτυχής αναζήτηση για το στοιχείο x σε ένα ΔΑΣ T ξεκινά από τη ρίζα του T και χρησιμοποιώντας το κλειδί του x ακολουθεί το μονοπάτι μέχρι τον κόμβο που περιέχει το x. Ο απαιτούμενος χρόνος είναι ανάλογος του μήκους του μονοπατιού ή αλλιώς του βάθους του συγκεκριμένου κόμβου που περιέχει το

x. Επομένως, με βάση το Λήμμα 10.7 ο μέσος χρόνος για μία πράξη επιτυχούς αναζήτησης είναι $\mathrm{O}(\log n)$.

Σε περίπτωση ανεπιτυχούς αναζήτησης για ένα ανύπαρκτο κλειδί μεταξύ των κλειδιών των στοιχείων x^- και x^+, ο μέσος χρόνος είναι $\mathrm{O}(\log n)$ αφού αυτή η διαδικασία θα πρέπει να σταματήσει είτε στο x^- είτε στο x^+ (θεωρούμε ότι τα x^- και x^+ είναι υπαρκτά).

Ένα στοιχείο εισάγεται στο ΔΑΣ αφού πρώτα το εισάγουμε σε ένα νέο κόμβο που προσάπτεται σε ένα φύλλο που βρέθηκε κατά την ανεπιτυχή αναζήτηση. Έπειτα το στοιχείο αυτό ανεβαίνει στο μονοπάτι προς τη ρίζα με απλές περιστροφές μέχρι να μην παραβιάζεται η διάταξη σωρού. Ο αριθμός των περιστροφών είναι το πολύ ίσος με το μήκος του μονοπατιού που διασχίσθηκε κατά την ανεπιτυχή αναζήτηση. Επομένως, ο χρόνος για την εισαγωγή είναι ίσος με το χρόνο μίας ανεπιτυχούς αναζήτησης, ο οποίος στη μέση περίπτωση είναι $\mathrm{O}(\log n)$.

Η πράξη της διαγραφής απαιτεί στην ουσία τον ίδιο χρόνο αφού είναι η ακριβώς αντίστροφη διαδικασία της εισαγωγής. Καταρχήν εντοπίζεται το προς διαγραφή στοιχείο με βάση το κλειδί του και έπειτα εκτελούνται απλές περιστροφές μέχρι αυτό το κλειδί να γίνει φύλλο, οπότε και το διαγράφουμε. Αν θα γίνει δεξιά ή αριστερή περιστροφή αποφασίζεται με βάση τις προτεραιότητες, αφού πάντα ο κόμβος με τη μεγαλύτερη προτεραιότητα θα περάσει προς τα επάνω.

Κάτι που είναι επίσης πολύ ενδιαφέρον είναι ο μέσος αριθμός περιστροφών ανά πράξη ενημέρωσης (εισαγωγής ή διαγραφής). Γενικώς, η περιστροφή είναι μία ακριβή πράξη σε σχέση με την προσπέλαση ενός κόμβου και θα επιθυμούσαμε το πλήθος τους να είναι κατά το δυνατόν μικρότερος. Αφού η διαγραφή είναι συμμετρική της εισαγωγής, θα μελετήσουμε μόνο την περίπτωση της διαγραφής. Έστω x ένα στοιχείο του ΔΑΣ T, το οποίο πρόκειται να διαγραφεί. Παρότι δεν μπορούμε να συμπεράνουμε από τη δομή του T ποιές περιστροφές θα εκτελεσθούν, ωστόσο μπορούμε να συμπεράνουμε πόσες θα είναι αυτές. Πιο συγκεκριμένα, το πλήθος τους θα είναι ίσο με το άθροισμα των μηκών της δεξιάς ράχης του αριστερού υποδένδρου του x και της αριστερής ράχης του δεξιού υποδένδρου του x. Η ορθότητα του επιχειρήματος φαίνεται από το γεγονός ότι μία αριστερή περιστροφή του x έχει το αποτέλεσμα της μείωσης κατά 1 του μήκους της αριστερής ράχης του δεξιού υποδένδρου, ενώ μία δεξιά περιστροφή του x μειώνει το μήκος κατά 1 της δεξιάς ράχης του αριστερού υποδένδρου. Επομένως, από το Λήμμα 10.7 προκύπτει ότι ο μέσος αριθμός περιστροφών είναι μικρότερος από 2.

10.7 Φίλτρα Bloom

Ένα φίλτρο Bloom είναι μία δομή δεδομένων που χρησιμοποιείται για να απαντήσει προσεγγιστικά ερωτήματα συμμετοχής ενός αριθμού σε ένα σύνολο αριθμών S. Ένα φίλτρο Bloom αποτελείται από έναν πίνακα A με m bits, που δεικτοδοτούνται από $A[0]$ μέχρι $A[m-1]$, όπου αρχικά $A[i] = 0, 0 \leq i \leq m-1$. Το φίλτρο χρησιμοποιεί k διαφορετικές συναρτήσεις κατακερματισμού στο διάστημα $[0, \ldots, m-1]$, το οποίο αντιστοιχεί στις m θέσεις του πίνακα. Υποθέτουμε ότι οι συναρτήσεις κατακερματισμού επιλέγουν κάθε κελί με ίδια πιθανότητα.

Η πρόσθεση ενός στοιχείου y στο σύνολο S επιτυγχάνεται εξάγοντας k θέσεις στον πίνακα από τις k διαφορετικές συναρτήσεις κατακερματισμού οι οποίες εφαρμόζονται στο συγκεκριμένο στοιχείο y. Έπειτα θέτουμε σε 1 τα κελιά που δεικτοδοτούνται από αυτούς τους k αριθμούς. Η ερώτηση αν ένα στοιχείο x συμμετέχει στο σύνολο S που ήδη έχει αποθηκευτεί σε ένα Bloom φίλτρο γίνεται εφαρμόζοντας τις k συναρτήσεις κατακερματισμού στο x. Έπειτα ελέγχουμε τις k θέσεις του πίνακα A και αν όλες είναι 1 τότε απαντάμε ότι το x ανήκει στο S αλλιώς αν έστω και ένα κελί έχει 0 τότε το στοιχείο x δεν ανήκει στο S.

Σε περίπτωση που η απάντηση συμμετοχής του x στο σύνολο S είναι αρνητική τότε το x σίγουρα δεν υπάρχει στο S που αποθηκεύεται στο φίλτρο Bloom. Αυτό γιατί αν είχε εισαχθεί στο φίλτρο Bloom τότε όλες οι αντίστοιχες θέσεις του πίνακα θα ήταν 1. Αν η απάντηση συμμετοχής είναι θετική τότε υπάρχει η περίπτωση το στοιχείο x να μην ανήκει στο S αλλά όλες οι θέσεις να έχουν γίνει 1 εξαιτίας της εισαγωγής άλλων στοιχείων στο S. Σε αυτή την περίπτωση έχουμε *ψευδή θετική απόκριση* στην ερώτηση για συμμετοχή του στοιχείου x.

Αν και υπάρχει η πιθανότητα ψευδούς θετικής απόκρισης, τα φίλτρα Bloom έχουν ένα εξαιρετικό πλεονέκτημα ως προς τη χρησιμοποίηση χώρου σε σχέση με άλλες δομές δεδομένων (όπως δέντρα, λίστες κοκ). Όλες αυτές οι δομές απαιτούν την αποθήκευση ολόκληρου του στοιχείου που δεικτοδοτείται, το οποίο μπορεί να είναι από μερικά bits (ακέραιος) μέχρι αρκετές εκατοντάδες ψηφιακών λέξεων (αλφαριθμητικά). Τα φίλτρα Bloom από την άλλη πλευρά με πιθανότητα ψευδούς θετικής απόκρισης 1% και βέλτιστη τιμή του k, απαιτούν περίπου 9.6 bits ανά στοιχείο ανεξαρτήτως μεγέθους του στοιχείου. Γενικά την πιθανότητα ψευδούς θετικής απόκρισης μπορούμε να την υποδεκαπλασιάζουμε απλά αυξάνοντας το χώρο ανά στοιχείο κατά 4.8 bits. Επιπλέον, το κόστος για ένθεση ενός στοιχείου είναι συνάρτηση μόνο του k και όχι του μεγέθους του στοιχείου ή του μεγέθους του συνόλου. Όμως ένα σημαντικό πρόβλημα που έχουν τα φίλτρα Bloom είναι ότι είναι δύσκολο να υποστηριχθεί η διαγραφή ενός στοιχείου.

Θεώρημα 10.1. *Η πιθανότητα ψευδούς θετικής απόκρισης είναι* $0.6185^{m/n}$.

Απόδειξη. Έστω ότι η συνάρτηση κατακερματισμού επιλέγει κάθε θέση του πίνακα με ίδια πιθανότητα. Αν m είναι ο αριθμός των bits στον πίνακα A, η πιθανότητα ότι κάποιο συγκεκριμένο bit δεν θα γίνει ίσο με 1 κατά την εισαγωγή ενός στοιχείου είναι:

$$1 - \frac{1}{m}$$

Η πιθανότητα ότι δεν θα τεθεί από καμία από τις k συναρτήσεις κατακερματισμού θα είναι:

$$\left(1 - \frac{1}{m}\right)^k$$

Αν ενθέσουμε n στοιχεία τότε η πιθανότητα να είναι 0 είναι:

$$\left(1 - \frac{1}{m}\right)^{kn} \approx e^{-kn/m}$$

και άρα η πιθανότητα αυτό να είναι 1 είναι:

$$1 - \left(1 - \frac{1}{m}\right)^{kn} \approx 1 - e^{-kn/m}$$

Έστω ότι ελέγχουμε την συμμετοχή στο S ενός στοιχείου x που δεν ανήκει σε αυτό. Τότε η πιθανότητα να έχουμε ψευδής θετική απόκρισης είναι:

$$\left(1 - e^{-kn/m}\right)^k$$

Έστω ότι $f = \left(1 - e^{-kn/m}\right)^k$. Η πιθανότητα ψευδούς θετικής απόκρισης μειώνεται όταν αυξάνεται το m και αυξάνεται καθώς το n αυξάνει. Για δοσμένα m και n, η τιμή του k που ελαχιστοποιεί αυτή την πιθανότητα προκύπτει από την ελαχιστοποίηση της συνάρτησης $g = k \ln\left(1 - e^{kn/m}\right)$, ενώ ισχύει ότι $f = e^g$. Άρα η καλύτερη τιμή για το k προκύπτει από την ελαχιστοποίηση της g ως προς k. Παίρνοντας 1η παράγωγο της g ως προς k βρίσκουμε ότι η συνάρτηση ελαχιστοποιείται για $k = \ln 2\frac{m}{n}$. Αντικαθιστώντας το k βρίσκουμε την πιθανότητα ψευδούς απόκρισης:

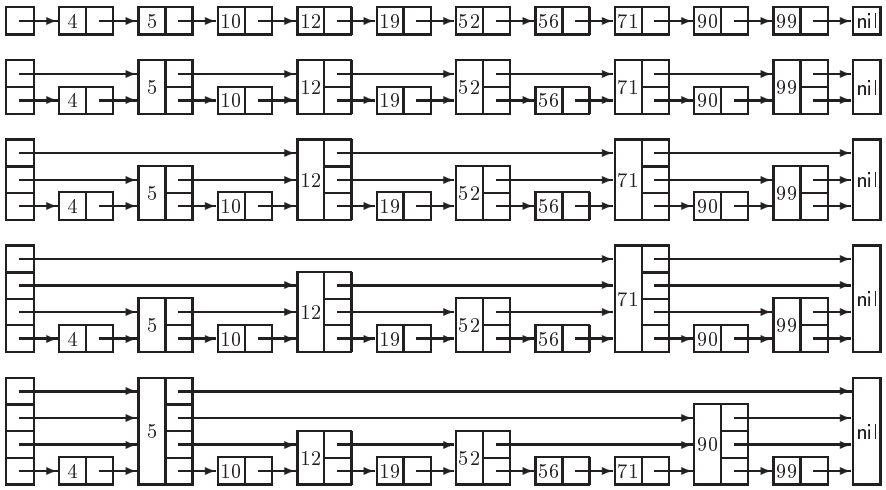
$$\left(1 - e^{-\ln 2m/n \cdot n/m}\right)^{\ln 2m/n} = \left(\frac{1}{2}\right)^{\ln 2m/n} \approx 0.6185^{m/n}$$

∎

10.8 Λίστες Παράλειψης

Η μέθοδος των **λιστών παράλειψης** (skip lists) προτάθηκε από τον Pugh το 1990. Η δομή αυτή είναι πολύ αποτελεσματική σε όλες τις σχετικές λειτουργίες, δηλαδή αναζήτηση, εισαγωγή και διαγραφή. Για την επεξήγηση της δομής πρέπει να γίνει μία μικρή εισαγωγή με ένα παράδειγμα.

Ας θεωρηθεί κατ'αρχήν μία κλασική απλή λίστα που περιέχει ταξινομημένα κλειδιά, όπως φαίνεται στην πρώτη δομή του επόμενου σχήματος. Είναι γνωστό ότι στη χειρότερη περίπτωση για μία επιτυχή αναζήτηση θα διασχισθεί η λίστα μέχρι τέλους. Στη δεύτερη δομή του σχήματος κάθε δεύτερος κόμβος της λίστας περιέχει δύο δείκτες αντί ένα: ο πρώτος δείκτης δείχνει στον επόμενο κόμβο, ενώ ο δεύτερος δείχνει στο μεθεπόμενο κόμβο της λίστας. Φαίνεται διαισθητικά ότι στη χειρότερη περίπτωση για μία επιτυχή αναζήτηση θα προσπελασθούν $\lceil n/2 \rceil + 1$ κόμβοι, όπου n είναι το μήκος της λίστας. Κατά παρόμοιο τρόπο στην τρίτη δομή του σχήματος, όπου κάθε τέταρτος κόμβος περιέχει τέσσερις δείκτες, μία επιτυχής αναζήτηση θα απαιτήσει στη χειρότερη περίπτωση την προσπέλαση $\lceil n/4 \rceil + 2$ κόμβων της λίστας.



Σχήμα 10.4: Συνδεδεμένες λίστες με επιπλέον δείκτες.

Στην τέταρτη δομή του σχήματος υπάρχουν κόμβοι με ένα δείκτη σε ποσοστό 50%, κόμβοι με δύο δείκτες σε ποσοστό 25%, κόμβοι με τρεις δείκτες σε ποσοστό 12,5% και τέλος κόμβοι με τέσσερις δείκτες σε ποσοστό 12,5%. Λέγεται ότι ο κόμβος με i δείκτες βρίσκεται στο i-οστό **επίπεδο** (level). Ακολουθώντας

το σκεπτικό αυτό είναι πλέον ευνόητο τι κατανομή κόμβων υπάρχει σε μία λίστα με κόμβους μέχρι i επίπεδα.

Σε μία λίστα παράλειψης με κόμβους σε i επίπεδα η κατανομή των κόμβων ακολουθεί τη μέθοδο αυτή, αλλά η αλληλουχία τους είναι τυχαία. Δείγμα δομής λιστών παράλειψης είναι η τελευταία περίπτωση του προηγούμενου σχήματος. Στη δομή αυτή ένας πίνακας δεικτών παίζει το ρόλο του δείκτη header των κλασικών συνδεδεμένων λιστών. Όλοι οι δείκτες αρχικοποιούνται με τιμή nil. Η αναζήτηση ενός κλειδιού σε μία τέτοια δομή γίνεται αρχικά ακολουθώντας την αλυσίδα των δεικτών του μεγαλυτέρου επιπέδου, μέχρι να προσπελασθεί κόμβος με κλειδί ίσο ή μεγαλύτερο από το αναζητούμενο. Στην πρώτη περίπτωση η διαδικασία τερματίζεται, ενώ στη δεύτερη η διαδικασία συνεχίζεται ακολουθώντας την αλυσίδα των δεικτών του χαμηλότερου επιπέδου. Έτσι η διαδικασία συνεχίζεται διαδοχικά θεωρώντας τις αλυσίδες των μικρότερων επιπέδων. Για παράδειγμα, κατά την αναζήτηση του κλειδιού 71 στην τελική δομή του σχήματος προσπελάζονται οι κόμβοι με κλειδιά 5, 90, 12, 52, 56 και τελικά 71.

Η επόμενη διαδικασία `search` περιγράφει την αναζήτηση σε μία λίστα παράλειψης του κλειδιού `searchkey`. Με τη μεταβλητή `level` συμβολίζεται το μέγιστο επίπεδο της λίστας παράλειψης, ενώ με `forward` συμβολίζονται οι δείκτες των διαδοχικών επιπέδων.

```
     procedure search(searchkey)
1.   x <-- header
2.   for  i <-- level downto 1 do
3.      while x.forward[i].key < searchkey
4.         do x <-- x.forward[i]
5.   x <-- x.forward[1]
6.   if x.key=searchkey then print(successful)
7.   else print(unsuccessful)
```

Η διαδικασία εισαγωγής ενός κλειδιού αρχίζει σαν τη διαδικασία της αναζήτησης, ώστε να εντοπισθεί το σημείο, όπου θα πρέπει να δημιουργηθεί ο νέος κόμβος. Ο αριθμός των δεικτών του νέου κόμβου προσδιορίζεται με ένα ψευδοτυχαίο αριθμό που υπακούει την κατανομή των κόμβων ανά επίπεδο. Για την ολοκλήρωση της διαδικασίας απαιτείται η σύνδεση του νέου κόμβου με τους υπόλοιπους της δομής. Αυτό επιτυγχάνεται με τη χρήση ενός βοηθητικού πίνακα που περιέχει τους δεξιότερους δείκτες όλων των επιπέδων που βρίσκονται αριστερά από το σημείο όπου θα γίνει η εισαγωγή. Ανάλογη διαχείριση απαιτείται και κατά τη διαγραφή ενός κόμβου, ώστε να γίνει η σύνδεση των κόμβων της λίστας. Κατά την υλοποίηση απαιτείται κάποια προσοχή στη διαχείριση των δεικτών επειδή η

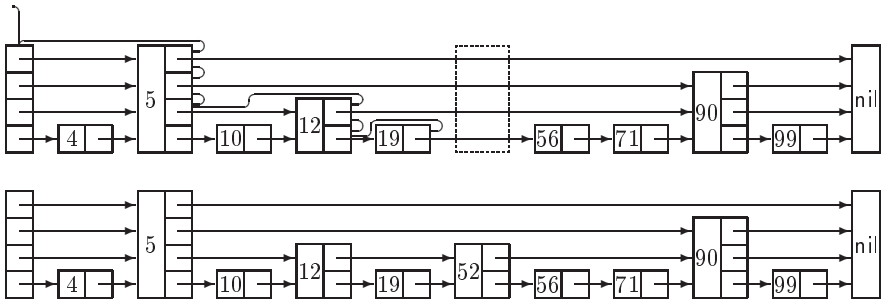
εισαγωγή ή η διαγραφή μπορεί να αφορά σε κόμβο οποιουδήποτε επιπέδου. Η επόμενη διαδικασία `insert` αποτυπώνει αυτήν την περιγραφή. Ο πίνακας `update` χρειάζεται για τη σύνδεση των κόμβων της δομής. Πιο συγκεκριμένα στη θέση `update[i]` τοποθετείται ο κατάλληλος δείκτης προς το δεξιότερο κόμβο επιπέδου i ή υψηλότερου, ο οποίος κόμβος βρίσκεται στα αριστερά του σημείου όπου θα εισαχθεί ο νέος κόμβος μέσω της `makenode`. Οι μεταβλητές `value` και `newvalue` αναφέρονται σε ένα βοηθητικό πεδίο.

```
     procedure insert(searchkey,newvalue)
1.   x <-- header
2.   for  <-- level downto 1 do
3.      while x.forward[i].key < searchkey
4.         do x <-- x.forward[i]
5.      update[i] <-- x
6.   x <-- x.forward[1]
7.   if x.key=searchkey then x.value <-- newvalue
8.   else
9.      newlevel <-- randomlevel
10.     if newlevel>level then
11.        for i <-- level+1 to newlevel do
12.           update[i] <-- header
13.        level <-- newlevel
14.     x <-- makenode(newlevel,searchkey,value)
15.     for i <-- 1 to newlevel do
16.        x.forward[i] <-- update[i].forward[i]
17         update[i].forward[i] <-- x
```

Στην πρώτη δομή του επόμενου σχήματος παρουσιάζεται η διαδικασία αναζήτησης της θέσης όπου θα πρέπει να εισαχθεί το κλειδί 52, ενώ στο κάτω μέρος του σχήματος δίνεται η τελική κατάσταση της δομής.

Στη διαδικασία `insert` σημαντικό ρόλο παίζει η διαδικασία `randomlevel`, που με μια λογική Las Vegas προσδιορίζει με τυχαίο τρόπο το επίπεδο του προς εισαγωγή κόμβου. Η διαδικασία `randomlevel` που δίνεται στη συνέχεια, είναι γενική καθώς με `p` συμβολίζεται το ποσοστό των κόμβων με δείκτες στο i-οστό επίπεδο, οι οποίοι έχουν δείκτες και στο $(i+1)$-οστό επίπεδο.

```
     procedure randomlevel
1.   newlevel <-- 1
2.   while random()<p do newlevel <-- newlevel+1
3.   return min(newlevel,maxlevel)
```



Σχήμα 10.5: Εισαγωγή σε λίστα παράλειψης.

Για $p = 1/2$, η διαδικασία μοιάζει με τη συνεχή ρίψη κερμάτων μέχρι να έρθουν 'γράμματα'. Η πιθανότητα ένα δεδομένο στοιχείο να αποθηκευθεί σε κόμβο του i-οστού επιπέδου είναι ίση με την πιθανότητα ρίχνοντας το κέρμα να βγάλουμε $i-1$ φορές 'κορώνα' και κατόπιν 'γράμματα'. Αυτό μπορεί να συμβεί με πιθανότητα $1/2^i$. Η πιθανότητα ένα στοιχείο να ξεπεράσει τα $c \log n$ επίπεδα είναι $1/2^{c \log n} = 1/n^c$. Άρα η πιθανότητα οποιοδήποτε στοιχείο να βρίσκεται σε παραπάνω από $c \log n$ επίπεδα δεν θα είναι μεγαλύτερη από $1/n^{c-1}$. Για παράδειγμα, το ύψος h είναι μεγαλύτερο από $3 \log n$ με πιθανότητα:

$$\frac{1}{n^2}$$

Γενικώς, δοθείσης σταθεράς $c > 1$, το ύψος h είναι μεγαλύτερο από $c \log n$ με πιθανότητα το πολύ $1/n^{c-1}$. Συνεπώς με πολύ μεγάλη πιθανότητα το ύψος είναι $\mathrm{O}(\log n)$. Ωστόσο, θεωρητικά η τιμή του ύψους μπορεί να γίνει οσοδήποτε μεγάλη. Για το λόγο αυτό αποφασίζεται από την αρχή η μέγιστη τιμή του επιτρεπτού πλήθους επιπέδων `maxlevel` να ισούται με $\log_{1/p} n = \frac{1}{\log 1/p} \log n$, και άρα $c = \frac{1}{\log 1/p}$, $p > 2$.

Στη συνέχεια θα ασχοληθούμε με την ανάλυση της πολυπλοκότητας της διαδικασίας της αναζήτησης. Η πολυπλοκότητα της διαδικασίας της εισαγωγής και της διαγραφής είναι παρόμοια, καθώς το επιπλέον κόστος είναι συνάρτηση του μέγιστου αριθμού των επιπέδων και όχι του πλήθους των στοιχείων της δομής. Για τους σκοπούς της ανάλυσης αυτής θα μελετήσουμε τα βήματα της αναζήτησης κατά την αντίστροφη φορά. Για παράδειγμα, έστω ότι στο Σχήμα 10.5 αναζητούμε το κλειδί 56. Ο Πίνακας 10.2 δείχνει τις εκτελούμενες ενέργειες κατά την αντίστροφη φορά.

Κόμβος	Επίπεδο	Κίνηση
56	0	πίσω
52	0	επάνω
52	1	πίσω
12	1	πίσω
5	1	επάνω
5	2	επάνω
header	3	πίσω

Πίνακας 10.2: Εκτελούμενες ενέργειες σε λίστα παράλειψης.

Πρόταση.
Η πολυπλοκότητα της αναζήτησης σε λίστα παράλειψης είναι λογαριθμική.

Απόδειξη
Η πολυπλοκότητα της αναζήτησης εξαρτάται από τον αριθμό των βημάτων προς τα επάνω και προς τα πίσω. Εκ κατασκευής η πιθανότητα από κάποιο σημείο να κινηθούμε προς τα πίσω είναι $1-p$, ενώ η πιθανότητα να κινηθούμε προς τα επάνω είναι p. Έστω ότι με $C(k)$ συμβολίζουμε τη μέση τιμή του μήκους του μονοπατιού αναζήτησης που ανεβαίνει k επίπεδα σε μία δομή με άπειρα επίπεδα. Ως αρχική συνθήκη ισχύει $C(0) = 0$ και γενικώς:

$$\begin{aligned} C(k) &= (1-p)\ (1+C(k))\ +\ p\ (1+C(k-1)) \\ &= 1/p\ +\ C(k-1)\ =\ k/p \end{aligned}$$

Με απλά λόγια, ένα μονοπάτι που ανεβαίνει k επίπεδα έχει μήκος $2k$ για $p = 1/2$. Ο μέγιστος αριθμός επιπέδων είναι $\log n$. Επομένως η δομή έχει λογαριθμική πολυπλοκότητα $\mathrm{O}(\log n)$. □

Πρόταση.
Η χωρική πολυπλοκότητα της λίστας παράλευψης είναι γραμμική.

Απόδειξη
Ως προς τη χωρική πολυπλοκότητα παρατηρούμε τα εξής υποθέτοντας ότι $p = 1/2$. Με το προηγούμενο σκεπτικό γνωρίζουμε ότι η πιθανότητα ένα στοιχείο να βρίσκεται στο i-οστό επίπεδο είναι ίση με την πιθανότητα ρίχνοντας το κέρμα να βγάλουμε $i-1$ φορές 'κορώνα' και κατόπιν 'γράμματα'. Αυτό μπορεί να συμβεί με πιθανότητα $1/2^i$. Επομένως η μέση τιμή των στοιχείων στο i-οστό επίπεδο

είναι $n/2^i$. Άρα η προσδοκητή τιμή του πλήθους των δεικτών είναι:

$$\sum_{i=0}^{h} \frac{n}{2^i} \;=\; n \sum_{i=1}^{h} \frac{1}{2^i} \;=\; n\, \frac{1-(\frac{1}{2})^{h+1}}{1-\frac{1}{2}} \;=\; n\, \left(2-(1/2)^h\right)$$

όπου $h = \log_{1/p} n$ είναι ο αριθμός των επιπέδων της δομής. Συνεπώς ασυμπτωτικά προκύπτει ότι η χωρική πολυπλοκότητα είναι $\Theta(n)$. □

10.9 Γρήγορη Ταξινόμηση

Μελετήσαμε τη γρήγορη ταξινόμηση στο Κεφάλαιο 9.3. Η επίδοση της γρήγορης ταξινόμησης εξαρτάται από την επιλογή του άξονα (pivot). Στη κλασική έκδοση του αλγορίθμου ως άξονας επιλέγεται το πρώτο στοιχείο του πίνακα. Αν η τιμή του άξονα είναι το μεσαίο στοιχείο του τελικώς ταξινομημένου πίνακα, τότε ο αρχικός πίνακας χωρίζεται σε δύο ίσους υποπίνακες. Σε αντίθεση με αυτή την καλύτερη περίπτωση, στη χειρότερη περίπτωση η τιμή του άξονα μπορεί να είναι η μικρότερη ή η μεγαλύτερη τιμή των στοιχείων του πίνακα, οπότε ο πίνακας θα υποδιαιρεθεί σε δύο υποπίνακες μεγέθους 0 και $n-1$.

Στη βιβλιογραφία έχουν προταθεί αρκετές παραλλαγές της γρήγορης ταξινόμησης με σκοπό την επιλογή ενός άξονα με τιμή που να οδηγεί σε καλύτερο διαμερισμό. Μία παραλλαγή είναι η λεγόμενη '**μεσαίος των τριών**'(median-of-three), όπου από το πρώτο, το τελευταίο και το μεσαίο στοιχείο του πίνακα επιλέγεται ως άξονας το στοιχείο με τη μεσαία τιμή μεταξύ των τριών. Μία άλλη παραλλαγή είναι η **ταξινόμηση με τη μέση τιμή** (meansort), όπου ως άξονας επιλέγεται η μέση τιμή των στοιχείων του πίνακα. Έχει αποδειχθεί ότι στη χειρότερη περίπτωση οι τεχνικές αυτές δεν αποφεύγουν την τετραγωνική πολυπλοκότητα $\Theta(n^2)$.

Σκοπός μας, λοιπόν, είναι ένας 'δίκαιος' (δηλαδή, αποτελεσματικός) διαμερισμός. Για το σκοπό αυτό προτείνεται η επιλογή ενός τυχαίου στοιχείου του πίνακα ως άξονα. Η επόμενη διαδικασία σκιαγραφεί την τεχνική αυτή τύπου Las Vegas.

```
     procedure quicksortrand(left,right);
1.   if left<right then
2.      i <-- left; j <-- right+1;
3.      k <-- random(left,right);
4.      swap(A[k],A[left]); pivot <-- A[left];
5.      repeat
6.         repeat i <-- i+1 until A[i]>=pivot;
7.         repeat j <-- j-1 until A[j]<=pivot;
```

```
8.          if i<j then swap(A[i],A[j]);
9.       until j<=i;
10.      swap(A[left],A[j]);
11.      quicksort(left,j-1);
12.      quicksort(j+1,right)
```

Στη συνέχεια θα προχωρήσουμε στην παρουσίαση δύο ανεξάρτητων αλλά ισοδύναμων αποδείξεων σχετικά με την πολυπλοκότητα της τυχαίας γρήγορης ταξινόμησης. Η πρώτη έχει ως εξής.

Πρόταση.
Η πολυπλοκότητα της γρήγορης ταξινόμησης είναι $\Theta(n \log n)$ στη μέση περίπτωση.

Απόδειξη 1η
Δεδομένου ενός συνόλου S, με S_i συμβολίζουμε τον i-οστό μικρότερο αριθμό. Ορίζουμε την τυχαία μεταβλητή X_{ij} ώστε να έχει τιμή 1 (0) αν τα στοιχεία S_i και S_j (δεν) συγκριθούν κατά την εκτέλεση της τυχαίας γρήγορης ταξινόμησης. Συνεπώς για την προσδοκητή τιμή του συνολικού αριθμού συγκρίσεων που εκτελούνται από τον αλγόριθμο ισχύει:

$$\mathrm{E}\left[\sum_{i=1}^{n-1}\sum_{j\geq i} X_{ij}\right] \;=\; \sum_{i=1}^{n-1}\sum_{j\geq i} \mathrm{E}[X_{ij}]$$

Καθώς η τυχαία μεταβλητή παίρνει τιμές 0 και 1, ουσιαστικά δηλώνει την πιθανότητα τα στοιχεία S_i και S_j να συγκριθούν. Έστωσαν τα στοιχεία $S_i, S_{i+1}, \ldots, S_j$. Τα δύο στοιχεία S_i και S_j θα συγκριθούν αν το ένα από τα δύο έχει επιλεγεί ως άξονας. Αν δεν έχει επιλεγεί το ένα από τα δύο ως άξονας, τότε θα έχει επιλεγεί ένα εκ των $S_{i+1}, \ldots, S_{j-1}$, γεγονός που θα στείλει τα δύο στοιχεία S_i και S_j σε διαφορετικούς υποπίνακες, οπότε δεν πρόκειται να συγκριθούν. Συνεπώς η πιθανότητα τα δύο αυτά στοιχεία να συγκριθούν είναι $2/(j-i+1)$. Άρα:

$$\begin{aligned}\sum_{i=1}^{n-1}\sum_{j\geq i} \mathrm{E}[X_{ij}] \;&=\; \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \frac{2}{j-i+1} \;=\; \sum_{i=1}^{n-1}\sum_{k=1}^{n-i} \frac{2}{k+1} \\ &<\; \sum_{i=1}^{n-1}\sum_{k=1}^{n-i} \frac{2}{k} \;=\; \sum_{i=1}^{n-1} \mathrm{O}(\log n) \;=\; \mathrm{O}(n \log n)\end{aligned}$$

□

Απόδειξη 2η
Η δεύτερη εναλλακτική απόδειξη στηρίζεται στις αναδρομικές εξισώσεις και τον

ορισμό της πολυπλοκότητας. Για την ανάλυσή μας, για τη θέση j του pivot θα θεωρήσουμε δύο ισοπίθανες περιπτώσεις (δηλαδή πιθανότητα 1/2):

- τον 'καλό' διαμερισμό' όπου ισχύει:

$$left + \frac{right - left + 1}{4} \leq j \leq right - \frac{right - left + 1}{4}$$

- τον 'κακό διαμερισμό' όπου ισχύει:

$$left \leq j < left + \frac{right - left + 1}{4}$$

ή

$$right > j \geq right - \frac{right - left + 1}{4}$$

Με απλά λόγια, καλός θεωρείται ο διαχωρισμός όταν ο τυχαία επιλεγόμενος άξονας οδηγεί σε διαμερισμό όπου ο μικρότερος υποπίνακας έχει μέγεθος το ελάχιστο $(right - left + 1)/4$, ενώ ο μεγαλύτερος υποπίνακας έχει μέγεθος το πολύ $3(right - left + 1)/4$.

Από το Κεφάλαιο 9.3 γνωρίζουμε ότι γενικώς ισχύει η επόμενη αναδρομική εξίσωση για $n \geq 2$:

$$T(n) \;=\; n + \frac{1}{n} \sum_{k=0}^{n-1} \; (T(k) + T(n-k-1))$$

Αν με κάποιο τρόπο μπορούσαμε να εγγυηθούμε ότι κάθε διαμερισμός θα είναι καλός, τότε θα ίσχυε η επόμενη αναδρομική εξίσωση:

$$\begin{aligned}
T_g(n) &= n + \frac{2}{n} \sum_{k=n/4}^{3n/4-1} (T_g(k) + T_g(n-k-1)) \\
&= n + \frac{4}{n} \sum_{k=n/4}^{n/2-1} (T_g(k) + T_g(n-k-1)) \\
&\leq n + \frac{4}{n} \sum_{k=n/4}^{n/2-1} (T_g(3n/4) + T_g(n/4)) \\
&= n + \frac{4}{n} \; (n/2 - n/4) \;\; (T_g(3n/4) + T_g(n/4)) \\
&= n + T_g(3n/4) + T_g(n/4)
\end{aligned}$$

Στην ανωτέρω επεξεργασία χρησιμοποιήσαμε το γεγονός ότι η μέγιστη τιμή της ποσότητας $T_g(k)+T_g(n-k+1)$ στο διάστημα $n/4 \leq j \leq 3n/4-1$ λαμβάνεται στα άκρα του διαστήματος[3].

Τώρα θέλουμε να αποδείξουμε ότι $T_g(n) \leq cn\log n$. Θα το αποδείξουμε με επαγωγή. Η αλήθεια της πρότασης ισχύει για μικρά n και για $n' < n$. Θα αποδείξουμε ότι ισχύει και για κάθε n. Διαδοχικά έχουμε:

$$\begin{aligned}
T_g(n) &\leq n + T_g(3n/4) + T_g(n/4) \\
&\leq n + c\,\frac{3n}{4}\,\log\frac{3n}{4} + c\,\frac{n}{4}\,\log\frac{n}{4} \;=\; n + \frac{c\,n}{4}\left(3\log\frac{3n}{4} + \log\frac{n}{4}\right) \\
&= n + \frac{c\,n}{4}\,(3\log n + 3\log 3 - 3\log 4 + \log n - \log 4) \\
&= n + \frac{c\,n}{4}\,(4\log n + 3\log 3 - 8) \;=\; cn\log n - n\left(2c - 1 - \frac{3c\log 3}{4}\right) \\
&= cn\log n - n\,(0.81c - 1) \;\leq\; cn\log n
\end{aligned}$$

όπου η αλήθεια της τελευταίας ανισοϊσότητας προκύπτει για $c \geq 2$.

Από την άλλη πλευρά υπάρχει και ο κακός διαμερισμός για τον οποίο ισχύει:

$$\begin{aligned}
T_b(n) &= n \;+\; \frac{4}{n}\sum_{k=3n/4}^{n-1}\,(T_b(k) + T_b(n-k-1)) \\
&\leq n \;+\; \frac{4}{n}\sum_{k=3n/4}^{n-1}\,(T_b(n-1) + T_b(0)) \\
&= n \;+\; \frac{4}{n}\sum_{k=3n/4}^{n-1} T_b(n-1) \;=\; n \;+\; T_b(n-1)
\end{aligned}$$

Από την τελευταία σχέση καταλήγουμε ότι: $T_b(n) = \Theta(n^2)$. Όμως με την όποια πιθανότητα θα συμβεί ένας κακός διαμερισμός με την ίδια πιθανότητα θα συμβεί

[3]Για λόγους ευκολίας τα όρια του αθροίσματος δεν ταυτίζονται με τα όρια που προσδιορίσθηκαν προηγουμένως στις περιπτώσεις του καλού και του κακού διαχωρισμού.

και ένας καλός διαμερισμός. Άρα στη γενική περίπτωση ισχύει:

$$\begin{aligned}
T(n) &= n + \frac{1}{n}\sum_{k=0}^{n-1}(T(k)+T(n-k-1)) \\
&= n + \frac{2}{n}\sum_{k=n/2}^{n-1}(T(k)+T(n-k-1)) \\
&= n + \frac{2}{n}\sum_{k=n/2}^{3n/4-1}(T(k)+T(n-k-1)) \;+\; \frac{2}{n}\sum_{k=3n/4}^{n-1}(T(k)+T(n-k-1)) \\
&\leq n + \frac{2}{n}\sum_{k=n/2}^{3n/4-1}(T(3n/4)+T(n/4)) \;+\; \frac{2}{n}\sum_{k=3n/4}^{n-1}(T(n-1)+T(0)) \\
&\leq n + \frac{2}{n}\,\frac{n}{4}\,(T(3n/4)+T(n/4)) \;+\; \frac{2}{n}\,\frac{n}{4}\,T(n-1) \\
&= n + \frac{1}{2}\,(T(3n/4)+T(n/4)+T(n-1))
\end{aligned}$$

Όπως και πριν, θα αποδείξουμε επαγωγικά ότι $T(n) \leq cn\log n$ για κάποια σταθερά c. Η αλήθεια της πρότασης ισχύει για μικρά n και για $n' < n$. Θα αποδείξουμε ότι ισχύει και για n. Διαδοχικά έχουμε:

$$\begin{aligned}
T(n) &\leq n + \frac{1}{2}\,(T(3n/4)+T(n/4)+T(n-1)) \\
&\leq n + \frac{1}{2}\left(c\frac{3n}{4}\log\frac{3n}{4} \;+\; c\frac{n}{4}\log\frac{n}{4} \;+\; c(n-1)\log(n-1)\right) \\
&= n + \left(\frac{3cn}{8}\log n \;+\; \frac{3cn}{8}\log\frac{3}{4} \;+\; \frac{cn}{8}\log n + \frac{cn}{8}\log\frac{1}{4} \;+\; \frac{c(n-1)}{2}\log(n-1)\right) \\
&= n + \left(\frac{cn}{2}\log n \;-\; cn \;+\; \frac{3cn\log 3}{8} \;+\; \frac{c(n-1)}{2}\log(n-1)\right) \\
&\leq n + \left(\frac{cn}{2}\log n \;-\; cn \;+\; \frac{3cn\log 3}{8} \;+\; \frac{cn}{2}\log n\right) \\
&= cn\log n \;-\; n\,(0.41c-1) \;\leq\; cn\log n
\end{aligned}$$

όπου η αλήθεια της τελευταίας ανισοϊσότητας προκύπτει για $c \geq 3$. □

Διαπιστώνουμε, λοιπόν, ότι και πάλι καταλήγουμε στο ίδιο συμπέρασμα σε σχέση με την επίδοση της μέσης περίπτωσης. Ωστόσο, ενώ στην ανάλυση

της κλασικής γρήγορης ταξινόμησης θεωρούμε το σύνολο των διαφορετικών διατάξεων που μπορούν να εισαχθούν προς ταξινόμηση, στην παρούσα περίπτωση θεωρούμε το σύνολο των δυνατών αποτελεσμάτων της τυχαίας γεννήτριας. Στη συμβατική γρήγορη ταξινόμηση μπορούσαμε να ισχυρισθούμε ότι: Αν τα δεδομένα είναι τυχαία, τότε η πολυπλοκότητα της μέσης περίπτωσης της γρήγορης ταξινόμησης είναι $\Theta(n \log n)$'. Αντίθετα τώρα μπορούμε να ισχυρισθούμε ότι: 'Με πολύ μεγάλη πιθανότητα η τυχαία γρήγορη ταξινόμηση εκτελείται σε χρόνο $\Theta(n \log n)$'.

10.10 Έλεγχος Πρώτων Αριθμών

Όπως γνωρίζουμε από την Άσκηση 1.12, πρώτος λέγεται ένας ακέραιος αριθμός, αν διαιρείται μόνο από τη μονάδα και τον εαυτό του. Το 1 δεν θεωρείται πρώτος, ενώ οι 5 μικρότεροι αριθμοί είναι οι 2, 3, 5, 7 και 11. Εκτός από τους αρχαίους Έλληνες (όπως ο Ευκλείδης και ο Ερατοσθένης), με το πρόβλημα είχαν ασχοληθεί και οι αρχαίοι Κινέζοι που πίστευαν ότι αν ο αριθμός $2^n - 2$ διαρείται ακριβώς δια του n, τότε ο n είναι πρώτος. Ωστόσο, η εικασία αυτή δεν είναι ορθή. Αντιπαράδειγμα αποτελεί ο αριθμός $n = 341$. Το πρόβλημα του ελέγχου αν ένας αριθμός είναι πρώτος ή σύνθετος (primality testing), καθώς και το πρόβλημα της παραγοντοποίησης (factorization) βρίσκουν σημαντική χρήση στην κρυπτογραφία, μία γνωστική περιοχή που βασίζεται στο γεγονός ότι δεν υπάρχουν αποτελεσματικοί αλγόριθμοι για την παραγοντοποίηση μεγάλων αριθμών.

Σύμφωνα με την Άσκηση 1.12, αν n είναι ένας θετικός σύνθετος ακέραιος, τότε ο n έχει έναν πρώτο αριθμό που δεν ξεπερνά το $\sqrt{n}$. Επομένως προκειμένου να διαπιστώσουμε αν ένας ακέραιος n είναι πρώτος αρκεί να εκτελέσουμε $\sqrt{n}$ δοκιμές στη χειρότερη περίπτωση και να απαντήσουμε με βεβαιότητα κατά πόσο ο αριθμός είναι πρώτος. Στην επόμενη διαδικασία `prime1` υποθέτουμε ότι ο `n` είναι περιττός αριθμός.

```
     procedure prime1(n)
1.   for i <-- 3 to n step 2 do
2.      if n mod i = 0 then return false;
3.   return true
```

Λέγεται ότι ένας αλγόριθμος **αποτελεσματικός χρονο-πολυωνυμικά** (polytime efficient) αν εκτελείται σε χρόνο πολυωνυμικό ως προς το μήκος του μεγέθους της αναπαράστασης του αριθμού σε bits, δηλαδή να ισχύει $O((\log n)^c)$, για κάποια σταθερά c. Προφανώς, η πολυπλοκότητα του ανωτέρω αιτιοκρατικού

αλγορίθμου είναι $O(\sqrt{n})$=$O(2^{\frac{1}{2}\log n})$ και επομένως ο αλγόριθμος δεν είναι αποτελεσματικός χρονο-πολυωνυμικά.

Στη συνέχεια θα εξετάσουμε τυχαίους αλγορίθμους που ανήκουν στην κατηγορία Monte Carlo και θα καταλήξουμε σε έναν πολύ αποτελεσματικό τυχαίο αλγόριθμο. Για παράδειγμα, έστω ότι δίνεται ο αριθμός $n = 2773 = 47 \times 59$, που προφανώς δεν είναι πρώτος. Θα μπορούσαμε να ακολουθήσουμε τη λογική του προβλήματος των επαναλαμβανόμενων στοιχείων και να δοκιμάσουμε αν το 2773 διαιρείται με κάποιον ακέραιο που επιλέγεται τυχαία στο διάστημα $[2, \lfloor\sqrt{2773}\rfloor] = [2, 52]$. Επομένως με πιθανότητα περίπου 2% θα επιλέγαμε το 47 και θα καταλήγαμε ότι το 2773 δεν είναι πρώτος αριθμός, όμως με περίπου 98% θα καταλήγαμε σε λάθος συμπέρασμα. Βέβαια κατά τον έλεγχο αριθμών με πολλά ψηφία (πράγμα που κατ'εξοχήν συμβαίνει στην κρυπτογραφία) η πιθανότητα για ορθή απάντηση θα είναι ακόμη μικρότερη. Θα μπορούσαμε να εκτελέσουμε το πείραμα αυτό περισσότερες φορές, ώστε να αυξήσουμε την πιθανότητα αποκλεισμού του λάθους, όπως φαίνεται στην επόμενη διαδικασία `prime2`. Η διαδικασία αυτή εκτελεί ένα μεγάλο αριθμό πειραμάτων (`large`), ώστε η πιθανότητα της ορθής απάντησης να είναι της τάξης $1 - n^{-a}$, όπως πρέπει να συμβαίνει στους αλγορίθμους που στηρίζονται σε μία λογική Monte Carlo.

```
     procedure prime2(n)
1.   for k <-- 1 to large do
2.      i <-- random(1,sqrt(n));
3.      if (n mod i = 0) then return false;
3.   return true
```

Στο αλγόριθμο αυτό οι αρνητικές απαντήσεις (δηλαδή, ότι ο αριθμός δεν είναι πρώτος) είναι με βεβαιότητα ορθές, ενώ οι θετικές απαντήσεις (δηλαδή, ότι ο αριθμός είναι πρώτος) είναι ορθές αλλά με μικρή πιθανότητα. Επίσης, η πιθανότητα του λάθους δεν είναι τόσο συνάρτηση του ελεγχόμενου αριθμού αλλά του αποτελέσματος της γεννήτριας τυχαίων αριθμών. Στη συνέχεια θα εξετάσουμε έναν εναλλακτικό τυχαίο αλγόριθμο που στηρίζεται στο επόμενο θεώρημα του Fermat, του οποίου την απόδειξη μπορούμε να βρούμε σε βιβλία Θεωρίας Αριθμών.

Θεώρημα.
Αν ο αριθμός n είναι πρώτος, τότε για $1 \leq a < n$ ισχύει: $a^{n-1} \bmod n = 1$.

Απόδειξη.

Μία πρώτη απόδειξη είναι επαγωγική ως προς a και θα αποδείξει την ισοδύναμη πρόταση: $a^n \bmod n = a$. Είναι προφανές ότι πάντοτε ισχύει η βασική συνθήκη για $a = 1$ (ακόμη και αν ο n δεν είναι πρώτος). Δεχόμαστε ότι ισχύει $a^n \bmod n = a$. Θα αποδείξουμε ότι $(a+1)^n \bmod n = a+1$. Έχουμε ότι:

$$(a+1)^n \;=\; \sum_{i=0}^{n} \binom{n}{i} a^{n-i}$$

Ο συντελεστής $\binom{n}{i}$ είναι διαιρετός δια n εκτός από τις περιπτώσεις $i = 0$ και $i = n$. Επομένως ισχύει:

$$(a+1)^n \bmod n \;=\; (a^n + 1) \bmod n \;=\; (a+1) \bmod n$$

Μία δεύτερη απόδειξη στηρίζεται στην εξής ιδέα. Έστωσαν οι αριθμοί $1, 2, \ldots, n-1$, που τους πολλαπλασιάσουμε επί a, οπότε προκύπτουν οι αριθμοί $a, 2a, \ldots, (n-1)a$. Οι αριθμοί αυτοί είναι διακριτοί ως προς $\bmod n$ και υπάρχουν $n-1$ τέτοιοι αριθμοί. Άρα οι αριθμοί αυτοί είναι ίδιοι με τους προηγούμενους αλλά με μία άλλη σειρά. Άρα ισχύει:

$$\begin{aligned} 1 \times 2 \times \ldots \times (n-1) \mod n &= a \times 2a \times \ldots \times (n-1)a \mod n \\ &= a^{n-1} \times 1 \times 2 \times \ldots \times (n-1) \bmod n \end{aligned}$$

Συνεπώς ισχύει η πρόταση. □

Ας δοκιμάσουμε το θεώρημα για τον πρώτο αριθμό $n = 5$. Πράγματι ισχύει: $4^4 \bmod 5 = 1$, $3^4 \bmod 5 = 1$ και $2^4 \bmod 5 = 1$. Ενώ για αντιπαράδειγμα, ο $n = 4$ δεν είναι πρώτος αριθμός και ισχύει: $3^3 \bmod 4 = 3$ και $2^3 \bmod 4 = 0$. Ωστόσο, το θεώρημα περιγράφει μία ικανή συνθήκη αλλά όχι και αναγκαία, δηλαδή δεν ισχύει και κατά την αντίστροφη σειρά. Έτσι υπάρχουν μη πρώτοι αριθμοί για τους οποίους ισχύει η συνθήκη μερικές φορές. Για παράδειγμα, για το μη πρώτο αριθμό $n = 9$ ισχύει: $a^8 \bmod 9 \neq 1$ για $1 < a < 8$ αλλά $8^8 \bmod 9 = 1$. Έτσι το 9 λέγεται **8-ψευδοπρώτος** (8-pseudoprime). Οι k-ψευδοπρώτοι αριθμοί είναι πιο σπάνιοι αριθμοί από τους πρώτους. Για παράδειγμα, το 9 είναι ο μικρότερος ψευδοπρώτος ακέραιος, ενώ ο μικρότερος 2-ψευδοπρώτος είναι ο 341 (δηλαδή ισχύει: $2^{340} \bmod 341 = 1$).

Με βάση το θεώρημα μπορούμε να σχεδιάσουμε έναν τυχαίο αλγόριθμο, όπου για δεδομένο περιττό αριθμό n, με τυχαίο τρόπο θα επιλέγουμε τιμές για το $1 < a < n$, ώστε να δοκιμάζουμε αρκετές φορές τη συνθήκη του θεωρήματος (και να μην πέσουμε εύκολα στην παγίδα των ψευδοπρώτων). Η επόμενη διαδικασία `prime3` αποτυπώνει αυτή την περιγραφή. Παρατηρούμε ότι στην περίπτωση αυτή

ο αλγόριθμος θα έχει την αντίστροφη λογική από τη λογική του αλγορίθμου prime2. Έτσι, οι αρνητικές απαντήσεις (δηλαδή, ότι ο αριθμός δεν είναι πρώτος) είναι με βεβαιότητα ορθές, ενώ οι θετικές απαντήσεις (δηλαδή, ότι ο αριθμός είναι πρώτος) είναι ορθές με μεγάλη πιθανότητα. Σημειώνεται επίσης ότι ο διπλός βρόχος while είναι μία ακόμη εκδοχή ύψωσης σε δύναμη (δες Κεφάλαιο 5.4).

```
     procedure prime3(n)
1.   q <-- n-1;
2.   for i <-- 1 to large do
3.      m <-- q; y <-- 1;
4.      a <-- random(2,n-1); z <-- a;
5.      while (m>0) do
6.         while (m mod 2 = 0) do
7.            z <-- (z*z) mod n; m <-- int(m/2);
8.         m <-- m-1; y <-- (y*z) mod n;
9.      if (y<>1) then rerurn false;
110.  return true
```

Η διαδικασία prime3 είναι σημαντικά αποτελεσματικότερη σε σχέση με την prime2 αλλά και πάλι δεν έχουμε λύσει το πρόβλημα καθ'ολοκληρία γιατί όπως δείξαμε στο προηγούμενο παράδειγμα το θεώρημα ισχύει κατά τη μία μόνο κατεύθυνση. Ωστόσο, όπως αναφέρεται στη βιβλιογραφία χειρότερη είναι η σύμπτωση των λεγόμενων αριθμών **Carmichael**, οι οποίοι δεν είναι πρώτοι αλλά πάντοτε ικανοποιούν το θεώρημα του Fermat. Αν και οι αριθμοί αυτοί είναι σπάνιοι, εντούτοις το σύνολό τους είναι άπειρο. Για παράδειγμα, αναφέρεται ότι υπάρχουν 255 τέτοιοι αριθμοί μέχρι το 100.000.000, ενώ οι εννέα μικρότεροι αριθμοί Carmichael είναι οι 561, 1105, 1729, 2465, 2821, 6601, 8911, 10585 και 15841, που περνούν όλες τις συνθήκες για κάθε θετικό a μικρότερο του αντίστοιχου αριθμού.

Πρόταση.
Ο αριθμός n είναι Carmichael αν και μόνο αν ισχύει $n = p_1 \times p_2 \times \ldots \times p_k$, όπου οι $p_1, p_2, \ldots, p_k$ είναι πρώτοι αριθμοί και ισχύει: $(n-1) \bmod (p_i - 1) = 0$ για κάθε i. □

Για παράδειγμα, ισχύει: $561 = 3 \times 11 \times 17$, ενώ $560 \bmod 2 = 0$, $560 \bmod 10 = 0$, και $560 \bmod 16 = 0$. Εν πάσει περιπτώσει, η προσέγγιση της διαδικασίας prime3 απειλείται καθώς ο αλγόριθμος αυτός θα δίνει (σπανίως βέβαια) λάθος συμπέρασμα, που θα ισχύει για τις ίδιες πάντοτε περιπτώσεις (δηλαδή, ανεξαρτήτως

του αποτελέσματος της γεννήτριας τυχαίων αριθμών). Όμως λύση υπάρχει και στηρίζεται στην εξής πρόταση.

Πρόταση.
Αν ο n είναι πρώτος αριθμός και ισχύει $1 \leq x < n$, τότε η εξίσωση $x^2 \bmod n = 1$ έχει μόνο 2 λύσεις, τις $x = 1$ και $x = n - 1$.

Απόδειξη.
Η εξίσωση αυτή σημαίνει ότι $(x^2 - 1) \bmod n = 0 \Rightarrow (x+1)(x-1) \bmod n = 0$. Εφ'όσον το n είναι πρώτος αριθμός και $1 \leq x \leq n$, έπεται ότι το n πρέπει να διαιρεί είτε το $x - 1$ είτε το $x + 1$. □

Από την πρόταση αυτή συνάγεται ότι αν υπάρχουν και άλλες ρίζες πλην των $x = 1$ και $x = n-1$, τότε ο αριθμός n δεν είναι πρώτος. Πράγματι, σε σχέση με το 561, το μικρότερο αριθμό Carmichael, ισχύει ότι η εξίσωση: $x^2 \bmod 561 = 1$ έχει τις δύο προφανείς λύσεις $x = 1$ και $x = 560$ καθώς και μερικές ακόμα όπως $x = 494$, $x = 373$ κοκ. Άρα λοιπόν ο 561 δεν είναι πρώτος. Στην πρόταση αυτή, λοιπόν, στηρίζεται η επόμενη διαδικασία `prime4` που βελτιώνει τη διαδικασία `prime3`. Μάλιστα, η εντολή PrimeQ του Mathematica υλοποιεί αυτόν τον αλγόριθμο, που φέρει το όνομα των Miller-Rabin.

```
     procedure prime2(n,alpha)
1.   q <-- n-1;
2.   for i <-- 1 to alpha*log(n) do
3.      m <-- q; y <-- 1;
4.      a <-- random(2,n-1); z <-- a;
5.      while (m>0) do
6.         while (m mod 2 = 0) do
7.            x <-- z; z <-- (z*z) mod n;
8.            if ((z=1) and (x<>1) and (x<>q)) then
9 .              return false;
10.           m <-- int(m/2);
11.        m <-- m-1; y <-- (y*z) mod n;
12.     if (y<>1) then rerurn false;
13.  return true
```

Σύμφωνα, λοιπόν, με τη διαδικασία αυτή, αν το n είναι πρώτος, τότε και θα ικανοποιείται το θεώρημα του Fermat αλλά και δεν θα υπάρχει κάποια μη τετριμμένη λύση σύμφωνα με το προηγούμενο θεώρημα. Έτσι ο αλγόριθμος

αυτός δίνει πάντοτε ορθή απάντηση για τους πρώτους αριθμούς και τους αριθμούς Carmichael. Ωστόσο, μπορεί να δώσει λάθος αποτέλεσμα αν πέσει στην παγίδα των ψευδοπρώτων αριθμών (δηλαδή, ο αριθμός να μην είναι πρώτος αλλά στην έξοδο να δοθεί σαν πρώτος). Η αξιοπιστία του αλγορίθμου στην περίπτωση αυτή εξαρτάται από τη γεννήτρια των τυχαίων αριθμών και τις τιμές της παραμέτρου a (εντολή 4), οι οποίες θα επιλεγούν για τη δοκιμή. Ας ονομάσουμε **μάρτυρα** (witness) την τιμή του a, η οποία αποκαλύπτει τη συνθετότητα του n. Η πιθανότητα επιλογής μίας τιμής μάρτυρα για το a δίνεται (χωρίς απόδειξη) από την επόμενη πρόταση, η οποία αποτελεί τη βάση για να φθάσουμε στην πολυπλοκότητα της `prime4`.

Λήμμα 10.8. *Αν ο περιττός αριθμός n δεν είναι πρώτος, τότε υπάρχουν τουλάχιστον $(n-1)/2$ τιμές του a, οι οποίες είναι μάρτυρες.*

Λήμμα 10.9. *Για κάθε μονό αριθμό $n > 2$ και ακέραιο k, η πιθανότητα λάθους της* `prime4` *είναι 2^{-k}.*

Απόδειξη. Με βάση το Λήμμα 10.8 η εκτέλεση κάθε επανάληψης του αλγορίθμου δίνει μάρτυρα a για την συνθετότητα του ακεραίου n με πιθανότητα $\geq 1/2$. Ο αλγόριθμος δεν θα ανακαλύψει ότι ο n είναι σύνθετος αν είναι τόσο άτυχος ώστε σε κάθε επανάληψη από τις συνολικά k επαναλήψεις να μην βρίσκει μάρτυρα. Η πιθανότητα αυτή είναι τουλάχιστον 2^{-k}. ∎

Θεώρημα 10.2. *Η πολυπλοκότητα της* `prime4` *είναι λογαριθμική.*

Απόδειξη. Αν το n είναι πρώτος, τότε πάντοτε λαμβάνουμε ορθή απάντηση, οπότε αυτή η περίπτωση δεν μας απασχολεί. Ας υποθέσουμε ότι το n δεν είναι πρώτος. Από το Λήμμα 10.9 προκύπτει ότι η πιθανότητα κανείς από τους επιλεγόμενους $a \times \log n$ τυχαίους αριθμούς να μην είναι μάρτυρας είναι $\leq (\frac{1}{2})^{a \log n} = n^{-a}$. Δηλαδή, η διαδικασία `prime4` θα δώσει λανθασμένη απάντηση με πιθανότητα $\leq n^{-a}$, οπότε έτσι ικανοποιείται η απαίτηση των αλγορίθμων Monte Carlo για επιτυχία με μεγάλη πιθανότητα ίση με $1 - n^{-a}$. Καθώς κάθε βρόχος `while` συνεπάγεται $O(\log n)$ επαναλήψεις, έπεται ότι η πολυπλοκότητα του αλγορίθμου είναι $O(\log^2 n)$. ∎

Μερικά επιλογικά σχόλια αναφορικά με το πρόβλημα του ελέγχου πρώτων αριθμών, για το οποίο μόλις μελετήσαμε μία αποτελεσματική λύση. Μέχρι πρόσφατα θεωρούνταν ότι το πρόβλημα αυτό δεν υπήρχε πολυωνυμικός αιτιοκρατικός αλγόριθμος. Στις 6 Αυγούστου του 2002, τρεις ερευνητές του Institute of Technology in Kampur, οι M. Agrawal, N. Kayal και N Saxena, δημοσίευσαν στο

διαδίκτυο μία μελέτη που λύνει το πρόβλημα σε πολυωνυμικό χρόνο. Έτσι ανοίγει μία νέα περίοδος στην έρευνα των γρήγορων μαθηματικών αλγορίθμων της Θεωρίας Αριθμών.

10.11 Στατιστικά Διάταξης

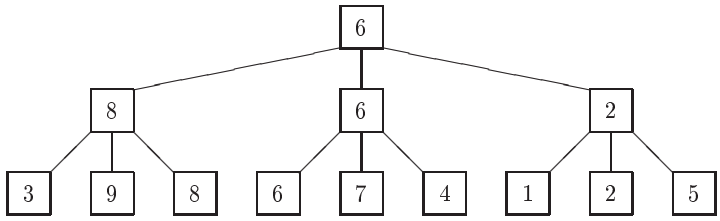
Στο Κεφάλαιο 9.4 είχαμε εξετάσει το πρόβλημα των στατιστικών διάταξης σε συνδυασμό με τη γρήγορη ταξινόμηση. Με παρόμοιο τρόπο, μπορούμε να διατυπώσουμε μία τυχαιοποιημένη εκδοχή της συνάρτησης για την επιλογή του k-οστού στοιχείου ενός αταξινόμητου πίνακα υιοθετώντας κάθε φορά μία τυχαία επιλογή του άξονα. Με παρόμοιο επίσης τρόπο θα μπορούσαμε να διατυπώσουμε την αντίστοιχη ανάλυση, που θα κατέληγε στο ίδιο συμπέρασμα, δηλαδή πολυπλοκότητα $\mathrm{O}(n)$. Η προσέγγιση αυτή είναι σχετικά εύκολη υπόθεση για τον αναγνώστη και για το λόγο αυτό στην παρούσα παράγραφο θα ασχοληθούμε με ένα νέο πρόβλημα: την εύρεση του **προσεγγιστικού μεσαίου** (approximate median). Τυπικά το πρόβλημα ορίζεται ως εξής.

Ορισμός.
Δεδομένου ενός πίνακα A με n αριθμούς, τότε ϵ-μεσαίος λέγεται ένας αριθμός q, τέτοιος ώστε $|A[i] < q| \leq (1-\epsilon)n$ και $|A[i] > q| \leq (1-\epsilon)n$. □

Για την ευκολία της επίλυσης του προβλήματος θεωρούμε ότι οι n τιμές του πίνακα είναι οι αριθμοί $0, \frac{1}{n}, \frac{2}{n}, \ldots, \frac{n-1}{n}$, ενώ επίσης δεχόμαστε ότι το n είναι δύναμη του 3, δηλαδή ισχύει $n = 3^k$. Θεωρούμε τη δομή ενός πλήρους τριαδικού δένδρου με $L = n$ φύλλα. Λαμβάνουμε μία τυχαία διάταξη από τις $n!$ συνολικά και θεωρούμε ότι τα στοιχεία αυτής της διάταξης αποτελούν τα φύλλα του δένδρου. Συνεχίζουμε την κατασκευή του δένδρου από το επίπεδο των φύλλων προς το επίπεδο της ρίζας αντιστοιχώντας 3 φύλλα σε ένα κόμβο-πατέρα με περιεχόμενο ίσο προς το μεσαίο των 3 στοιχείων των κόμβων-παιδιών. Η διαδικασία συνεχίζεται αναδρομικά μέχρι το επίπεδο της ρίζας. Στο σημείο αυτό, μία πρώτη παρατήρηση είναι ότι η τάξη του στοιχείου της ρίζας θα είναι $n/2$ στη μέση περίπτωση. Το Σχήμα 10.6 απεικονίζει ένα παράδειγμα με $n = 9$.

Έστω ότι με $P^h_{<x}$ συμβολίζουμε την πιθανότητα η έξοδος του αλγορίθμου να μην είναι μεγαλύτερη από x, δεδομένου ενός δένδρου με ύψος h. Για το επίπεδο των φύλλων ισχύει η βασική συνθήκη $P^0_{<x} = x$. Θα υπολογίσουμε την πιθανότητα $P^h_{<x}$ με αναδρομικό τρόπο.

Έστω η ρίζα του δένδρου, η οποία έχει 3 παιδία με τιμές c_1, c_2, c_3. Η μεσαία τιμή των τριών αυτών τιμών μπορεί να είναι λιγότερο από x αν και μόνον αν



Σχήμα 10.6: Πλήρες τριαδικό δένδρο με $n = 9$.

δύο από τις τιμές αυτές είναι μικρότερη από x. Αυτό μπορεί να συμβεί σε δύο περιπτώσεις:

- ακριβώς δύο από τις τιμές των παιδιών είναι λιγότερο από x.
- και οι τρεις τιμές είναι λιγότερο από x.

Έτσι προκύπτουν οι σχέσεις:

$$\begin{aligned} P^h_{<x} &= \binom{3}{2}(P^{h-1}_{<x})^2(1-P^{h-1}_{<x}) + (P^{h-1}_{<x})^3 \\ &= 3\,(P^{h-1}_{<x})^2 - 2\,(P^{h-1}_{<x})^3 \;\leq\; 3\,(P^{h-1}_{<x})^2 \end{aligned}$$

Με διαδοχικές αντικαταστάσεις έχουμε:

$$\begin{aligned} P^h_{<x} &\leq 3\,(P^{h-1}_{<x})^2 \;\leq\; 3\,3^2\,3^4\ldots 3^{2^{h-1}}\,(P^0_{<x})^{2^h} \\ &\leq 3\,3^2\,3^4\ldots 3^{2^{h-1}}\,x^{2^h} \;=\; 3^{1+2+4+\ldots+2^{h-1}}\,x^{2^h} \\ &= 3^{2^h-1}\,x^{2^h} \;=\; \frac{1}{3}\,3^{2^h}\,x^{2^h} \;=\; \frac{1}{3}\,(3x)^{2^h} \end{aligned}$$

Έτσι φθάνουμε στο συμπέρασμα ότι με ένα τυχαίο αλγόριθμο μπορούμε να βρούμε τον προσεγγιστικό μεσαίο με πολύ μεγάλη πιθανότητα.

Ασκήσεις

1. Τα φίλτρα Bloom μπορούν να χρησιμοποιηθούν για να υπολογίσουμε προσεγγιστικά τη διαφορά συνόλων. Έστω δύο σύνολα X και Y καθένα από τα οποία έχει n στοιχεία. Δημιουργούμε Bloom φίλτρα και για τα δύο σύνολα χρησιμοποιώντας τον ίδιο αριθμό από bits m και πλήθος συναρτήσεων κατακερματισμού k. Να καθορίσετε το αναμενόμενο πλήθος bits όπου τα δύο φίλτρα διαφέρουν μεταξύ τους σαν συνάρτηση των m, n, k και $|X \cap Y|$.

2. Να εισάγετε την παρακάτω ακολουθία αριθμών με την σειρά που εμφανίζεται σε (α) μία λίστα παράλειψης ($p = 1/3$), (β) σε ένα δέντρο αναζήτησης σωρού και (γ) σε ένα φίλτρο Bloom ($m = 30, k = 2$ - επιλέξτε εσείς τις συναρτήσεις κατακερματισμού).

   $63, 30, 36, 31, 12, 50, 35, 5, 27, 59, 43, 17$

   Έπειτα διαγράψτε τα στοιχεία $59, 5$ από τη λίστα παράλειψης και από το δέντρο αναζήτησης σωρού.

3. Τα φίλτρα Bloom δεν έχουν την δυνατότητα διαγραφής ενός στοιχείου. Μπορείτε να αλλάξετε τα φίλτρα Bloom ώστε να είναι σε θέση να υποστηρίζουν την πράξη της διαγραφής; Πόσος είναι ο χώρος που θα χρησιμοποιούν σε αυτή την περίπτωση;

4. Έστω ϵ, δ πραγματικοί αριθμοί έτσι ώστε: $0 < \epsilon < \delta < 1$. Έστω A ένας τυχαίος αλγόριθμος που υπολογίζει μία συνάρτηση F όπου:

   $P(A(x) = F(x)) \geq \epsilon$ και $P(A(x) =?) = 1 - P(A(x) = F(x))$

   Έστω ότι για κάθε $k \in N, k \geq 2$, A_k είναι ο τυχαίος αλγόριθμος ώστε για κάθε είσοδο x ο A εκτελείται k φορές στο x. Η έξοδος του A_k είναι ? αν και μόνο αν και οι k εκτελέσεις του A στο x έδωσαν έξοδο ? (έξοδος ? σημαίνει ότι η έξοδος είναι λανθασμένη). Σε κάθε άλλη περίπτωση ο αλγόριθμος A_k υπολογίζει το σωστό αποτέλεσμα $F(x)$. Να υπολογίσετε το μικρότερο k ώστε ώστε $P(A_k(x) = F(x)) \geq \delta$.

5. Έστω ότι A είναι ένας Monte Carlo αλγόριθμος έτσι ώστε για κάθε είσοδο x, υπολογίζει το σωστό αποτέλεσμα $F(x)$ με πιθανότητα $\frac{1}{2} + \epsilon_x$, όπου ϵ_x εξαρτάται από το μέγεθος του x (αριθμός bits, έστω $|x|$). Έστω δ μία σταθερά, $0 < \delta < 1/2$. Πόσες επαναλήψεις $k = k(|x|)$ του A στο x πρέπει να γίνουν έτσι ώστε να επιτύχουμε $P(A_k(x) = F(x))) \geq 1 - \delta$, αν $\epsilon_x = \frac{1}{|x|}$.

6. Έστω ότι A είναι ένας τυχαίος αλγόριθμος που υπολογίζει μία συνάρτηση F με $P(A(x) = F(x)) \geq 1/3$ για κάθε x. Επίσης υποθέστε ότι ξέρετε ότι

$P(A(x) = \alpha) \leq 1/4$ για κάθε λανθασμένο αποτελέσμα α (η πιθανότητα υπολογισμού λανθασμένης τιμής είναι το πολύ $1/4$). Μπορείτε να χρησιμοποιήσετε αυτή την πληροφορία ώστε να σχεδιάσετε έναν χρήσιμο τυχαίο αλγόριθμο για τη συνάρτηση F?

7. n σφαίρες ρίχνονται σε n κάδους σε μία επαναληπτική διαδικασία. Σε κάθε επανάληψη κάθε σφαίρα επιλέγει να πέσει σε ένα κάδο ισοπίθανα. Αν μία σφαίρα πέσει μόνη της σε έναν κάδο τότε η σφαίρα αυτή αφαιρείται και η διαδικασία επαναλαμβάνεται με τις υπόλοιπες σφαίρες (αυτό το παιχνίδι μοντελοποιεί ένα απλό σύστημα όπου n διεργασίες χρησιμοποιούν τυχαία n συσκευές επικοινωνίας και κάθε συσκευή εξυπηρετεί μόνο αν της πέσει μία και μόνο μία διεργασία αλλιώς τις απορρίπτει όλες).

   Αν υπάρχουν b σφαίρες στην αρχή της επανάληψης, ποιό είναι το αναμενόμενο πλήθος σφαιρών στο τέλος μίας επανάληψης?

8. Δοθέντος ενός δικαίου νομίσματος (πιθανότητα για Γράμματα (Γ) ή Κορώνα (Κ) είναι $1/2$) να εξομοιώσετε ένα κίβδηλο νόμισμα όπου η πιθανότητα εμφάνισης Κ είναι p και η πιθανότητα εμφάνισης Γ είναι $1-p$. Προσπαθείστε να ελαχιστοποιήσετε το μέσο αριθμό πειραμάτων (στρίψιμο νομίσματος) του δικαίου νομίσματος.

9. Θεωρείστε τον παρακάτω αλγόριθμο εύρεσης ελαχίστου στοιχείου σε έναν μη ταξινομημένο πίνακα:

```
     procedure random(A[1,...,n])
1.   min=Infinity;
2.   for i <-- 1 to n in random order do
3.      if (A[i]<min)
4.       min<--A[i];
5.      return min
```

   (α) Στη χειρότερη περίπτωση, πόσες φορές θα εκτελεστεί η γραμμή 4?

   (β) Ποιά είναι η πιθανότητα η γραμμή 4 να εκτελεστεί κατά την n-ιοστή επανάληψη?

   (γ) Ποιος είναι ο ακριβής αναμενόμενος αριθμός εκτελέσεων της γραμμής 4?

10. Έστω S ένα σύνολο n σημείων στο επίπεδο. Ένα σημείο p στο S καλείται Pareto-βέλτιστο αν δεν υπάρχει άλλο σημείο που να είναι πιο πάνω και δεξιά από το p.

(α) Σχεδιάστε και περιγράψτε έναν αλγόριθμο που υπολογίζει τα Pareto-βέλτιστα σημεία του S σε χρόνο $O(n \log n)$.

(β) Έστω ότι κάθε σημείο του S επιλέγεται ανεξάρτητα και ισοπίθανα από το μοναδιαίο τετράγωνο $[0,1] \times [0,1]$. Ποιο είναι το αναμενόμενο πλήθος Pareto-βέλτιστων σημείων στο S?

11. Έστω ότι παρακολουθούμε μία ροή (stream) από δεδομένα τα οποία τα επεξεργαζομαστε ένα κάθε φορά. Από τη στιγμή που ένα στοιχείο x της ροής επεξεργαστεί και έρθει το επόμενο του, το x χάνεται. Θέλουμε να πάρουμε ένα τυχαίο δείγμα k διαφορετικών στοιχείων από τη ροή με τους παρακάτω περιορισμούς:

    - Μπορούμε να αποθηκεύσουμε μόνο k στοιχεία κάθε χρονική στιγμή (το μέγεθος της μνήμης χωράει k στοιχεία).
    - Δεν γνωρίζουμε το συνολικό αριθμό πακέτων στη ροή.
    - Αν αποφασίσουμε να μην αποθηκεύσουμε ένα στοιχείο την στιγμή που φτάνει σε εμάς, τότε αυτό χάνεται για πάντα.

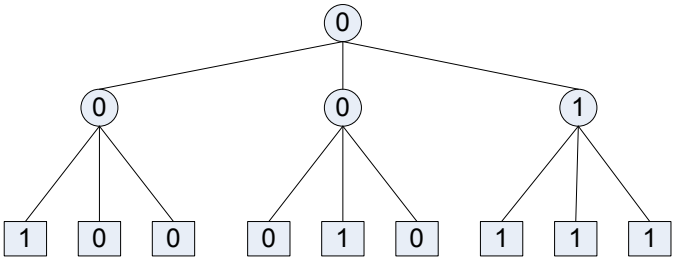
    Σχεδιάστε έναν αλγόριθμο ώστε όταν η ροή τερματιστεί, θα έχουμε αποθηκεύσει ένα υποσύνολο της ροής με k στοιχεία επιλεγμένα με ίση πιθανότητα από όλα τα στοιχεία της ροής. Αν ο συνολικός αριθμός στοιχείων της ροής είναι μικρότερος του k, τότε θα πρέπει να αποθηκεύσετε όλα τα πακέτα.

12. Μας δίνεται ένα κίβδηλο νόμισμα με πιθανότητα εμφάνισης Κορώνας (Κ) p και Γραμμάτων (Γ) $1-p$. Πως θα εξομοιώσετε ένα δίκαιο νόμισμα (πιθανότητα Κ και Γ είναι $1/2$) χρησιμοποιώντας το κίβδηλο νόμισμα?

13. Έστω ότι ο $X[1 \ldots n]$ είναι ένας πίνακας πραγματικών αριθμών (διαφορετικών μεταξύ τους), και έστω $N[1 \ldots n]$ ένας πίνακας δεικτών για τον πίνακα X με την εξής ιδιότητα: Αν το $X[i]$ είναι το μεγαλύτερο στοιχείο του X, τότε το $X[N[i]]$ είναι το μικρότερο στοιχείο του X. Διαφορετικά το $X[N[i]]$ είναι το μικρότερο στοιχείο του X μεγαλύτερο από το $X[i]$. Για παράδειγμα:

i	1	2	3	4	5	6	7	8	9
$X[i]$	83	54	16	31	45	99	78	62	27
$N[i]$	6	8	9	5	2	3	1	7	4

    Περιγράψτε και αναλύστε έναν τυχαίο αλγόριθμο που αποφασίζει αν ένας αριθμός x εμφανίζεται στον πίνακα X σε αναμενόμενο χρόνο $O(\sqrt{n})$. Ο



Σχήμα 10.7: Ένα τριαδικό δέντρο πλειοψηφίας με βάθος $d = 2$.

αλγόριθμος που θα προτείνετε δεν πρέπει να μεταβάλει τους πίνακες X και N.

14. Ένα δέντρο πλειοψηφίας είναι ένα πλήρες δέντρο βάθους d, όπου κάθε φύλλο έχει τιμή 0 ή 1. Η τιμή ενός εσωτερικού κόμβου είναι η πλειοψηφούσα τιμή των παιδιών του. Θα ασχοληθούμε με το πρόβλημα εύρεσης της τιμής της ρίζας ενός τριαδικού πλειοψηφικού δέντρου δοθείσης της ακολουθίας των 3^d φύλλων σαν είσοδο. Για παράδειγμα, αν $d = 2$ και τα φύλλα έχουν τιμή $1, 0, 0, 0, 1, 0, 1, 1, 1$ η ρίζα έχει τιμή 0 (Σχήμα 10.7). Περιγράψτε και αναλύστε έναν τυχαίο αλγόριθμο που βρίσκει την τιμή της ρίζας σε αναμενόμενο χρόνο $O(c^d)$ για κάποια σταθερά $c < 3$.

   [Υπόδειξη: Κοιτάξτε την απλή περίπτωση $d = 1$. Έπειτα εφαρμόστε αναδρομικά την διαδικασία αυτή.]

15. Έστω ότι θέλουμε να φτιάξουμε ένα αλγόριθμο $RP(n)$ που βρίσκει μία αναδιάταξη των ακεραίων από $\{1, \ldots, n\}$, όπου κάθε δυνατή αναδιάταξη έχει ίδια πιθανότητα εμφάνισης. Ο παρακάτω αλγόριθμος είναι μία τέτοια υλοποίηση:

```
     procedure RP(n)
1.   for i <-- 1 to n
2.      π[i]<--NULL; //Το π είναι ο πίνακας που θα περιέχει
                     //την αναδιάταξη
3.   for i <-- 1 to n
4.      j<--Random(n); //Επιστρέφει ένα τυχαίο ακέραιο στο
                       //διάστημα [1,n] ισοπίθανα
5.      while(π[j]!=NULL)
6.         j<--Random(n);
7.      π[j]=i;
```

```
8.   return(π);
```

Απόδείξτε ότι ο αλγόριθμος είναι σωστός (να δείξετε δηλαδή ότι πράγματι παράγει μία αναδιάταξη ισοπίθανα). Ποιός είναι ο αναμενόμενος χρόνος εκτέλεσης του συγκεκριμένου αλγόριθμου?

11

Επιμερισμένη Ανάλυση

Περιεχόμενα Κεφαλαίου

11.1 Επιμερισμένη Ανάλυση

Η επιμερισμένη ανάλυση αποτελεί ένα ισχυρό εργαλείο για την ανάλυση δομών δεδομένων και αλγορίθμων. Η **επιμερισμένη ανάλυση** (amortized analysis) χρησιμοποιείται για να δείξει ότι το μέσο κόστος μίας πράξης είναι μικρό, όταν αυτό υπολογίζεται σαν ο μέσος όρος μίας ακολουθίας πράξεων, παρότι μπορεί να υπάρχουν επιμέρους πράξεις με μεγάλο κόστος. Οι υλοποιήσεις με καλές επιμερισμένες πολυπλοκότητες συχνά είναι απλούστερες και αποδοτικότερες σε σχέση με τις υλοποιήσεις πολυπλοκοτήτων χειρότερης περίπτωσης. Προσοχή, η επιμερισμένη ανάλυση απλά αποτελεί ένα εργαλείο προσδιορισμού της απόδοσης μιας δομής ή ενός αλγόριθμου και όχι σχεδιαστικό εργαλείο. Βεβαίως, ενίοτε η ανάδραση από αυτή την ανάλυση μπορεί να δώσει κατευθύνσεις για καλύτερο σχεδιασμό.

Η ανάλυση μέσης περίπτωσης και η πιθανοτική ανάλυση δεν έχουν σχέση με την επιμερισμένη ανάλυση. Στην ανάλυση μέσης περίπτωσης, υπολογίζουμε το μέσο όρο σε όλες τις δυνατές εισόδους ενώ στην πιθανοτική υπολογίζουμε το μέσο όρο σε όλες τις τυχαίες επιλογές που κάνουμε. Στην επιμερισμένη ανάλυση υπολογίζουμε το μέσο όρο μίας ακολουθίας πράξεων. Η επιμερισμένη ανάλυση υποθέτει ακολουθία πράξεων χειρότερης περίπτωσης και δεν χρησιμοποιεί τυχαιότητα.

Η τεχνική αυτή ανάλυσης απαιτεί τον καθορισμό των διαφορετικών ακολουθιών πράξεων που μπορεί να συμβούν. Αυτή είναι η συνηθισμένη περίπτωση σε δομές δεδομένων, όπου η κατάσταση της δομής παραμένει αμετάβλητη μεταξύ δύο διαδοχικών πράξεων σε μία τέτοια ακολουθία. Η βασική ιδέα είναι ότι μία ακριβή πράξη θα αλλάξει την κατάσταση έτσι ώστε οι επόμενες πράξεις θα είναι φθηνότερες, και επομένως μπορούμε να επιμερίσουμε το κόστος. Ας θεωρήσουμε ένα παράδειγμα ώστε να κατανοήσουμε τη διαφορά μεταξύ επιμερισμένης πολυπλοκότητας και πολυπλοκότητας χειρότερης περίπτωσης.

> Ο Γιάννης αγοράζει ένα επαγγελματικό αυτοκίνητο και κλείνει ένα ειδικό συμβόλαιο για τη συντήρησή του με διάρκεια ενός χρόνου. Δίνει 50 ευρώ κάθε μήνα εκτός των Μαρτίου, Ιουνίου, Σεπτεμβρίου και Δεκεμβρίου (καλύπτει αλλαγή λαδιών), 100 ευρώ κάθε Μάρτιο, Ιούνιο και Σεπτέμβριο (καλύπτει αλλαγή λαδιών και ένα μικρό service) και 200 ευρώ κάθε Δεκέμβριο (καλύπτει αλλαγή λαδιών και ένα μεγάλο service). Ποιό είναι το κόστος ανά μήνα?
>
> Στην περίπτωση της πολυπλοκότητας χειρότερης περίπτωσης, μπορούμε να φράξουμε το κόστος ανά μήνα από το μέγιστο κόστος, δηλαδή τα 200 ευρώ για το Δεκέμβριο. Επομένως, για 12 μήνες το κόστος θα

είναι 2400 ευρώ. Στην περίπτωση της επιμερισμένης πολυπλοκότητας, παρατηρούμε ότι μόνο μία φορά το χρόνο πληρώνουμε 200 ευρώ, 3 φορές το χρόνο πληρώνουμε 100 ευρώ και τέλος 8 φορές το χρόνο πληρώνουμε 50 ευρώ. Επομένως, το κόστος ανά μήνα θα είναι $\frac{200+3\cdot 100+8\cdot 50}{12} = 75$ ευρώ. Το επιμερισμένο κόστος λοιπόν ανά μήνα θα είναι 75 ευρώ ενώ το χειρότερο κόστος θα είναι 200 ευρώ.

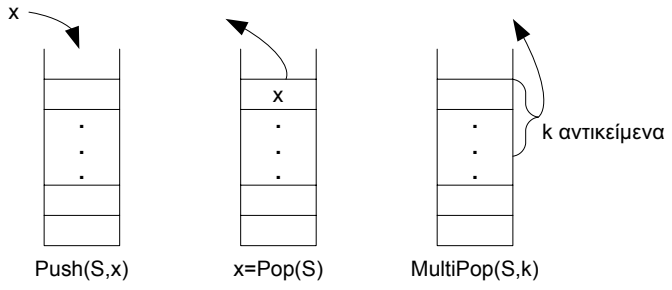
Η επιμερισμένη πολυπλοκότητα μίας δομής είναι πάντα μικρότερη ή ίση από την πολυπλοκότητα χειρότερης περίπτωσης. Η εξίσωση των δύο πολυπλοκοτήτων σε μία δομή δεδομένων συνήθως την καθιστά πολυπλοκότερη και μάλιστα μερικές φορές είναι αδύνατη.

11.2 Τεχνικές Επιμερισμένης Ανάλυσης

Στη συνέχεια θα αναφερθούμε σε τρεις τεχνικές που χρησιμοποιούνται στην επιμερισμένη ανάλυση: στην τεχνική του αθροίσματος, στην τεχνική του λογιστή και στην τεχνική του φυσικού [158]. Η τεχνική του φυσικού και η τεχνική του λογιστή έχουν ίδια αναλυτική δύναμη αλλά κάθε φορά ενδείκνυται η χρήση μίας εκ των δύο αναλόγως με το πρόβλημα. Αυτές οι τρεις τεχνικές θα δοθούν σε συνδυασμό με ένα απλό παράδειγμα.

Έστω ένας σωρός $\mathcal{S}$ που υποστηρίζει τις εξής πράξεις (δες Σχήμα 11.1):

- Push$(\mathcal{S}, x)$: Βάζει στην κορυφή του σωρού $\mathcal{S}$ το στοιχείο x.
- Pop$(\mathcal{S})$: Επιστρέφει το στοιχείο στην κορυφή του σωρού $\mathcal{S}$, εφόσον ο σωρός δεν είναι άδειος, ενώ ταυτοχρόνως το διαγράφει.



Σχήμα 11.1: Οι τρεις πράξεις που υποστηρίζονται από το σωρό.

- MultiPop($\mathcal{S}, k$): Επιστρέφει τα πρώτα k στοιχεία του σωρού $\mathcal{S}$, ή λιγότερα αν ο σωρός δεν περιέχει τόσα πολλά, και τα διαγράφει από το σωρό.

Έστω ότι κάθε πράξη Push και Pop έχει μοναδιαίο κόστος και αποτελούν τις στοιχειώδεις πράξεις που μετρούμε για τον προσδιορισμό της αποδοτικότητας του σωρού. Η πράξη MultiPop($\mathcal{S}, k$) έχει κόστος χειρότερης περίπτωσης ίσο με k, δηλαδή στη χειρότερη περίπτωση το κόστος της πράξης αυτής είναι $\mathrm{O}(n)$ αν στο σωρό βρίσκονται n στοιχεία, καθώς εκτελεί n πράξεις Pop. Είναι όμως αυτό το επιμερισμένο της κόστος? Δηλαδή, ποιό είναι το συνολικό κόστος μίας οποιασδήποτε ακολουθίας πράξεων $P = p_1, p_2, \ldots, p_n$ επάνω στο σωρό $\mathcal{S}$?

Θα χρησιμοποιήσουμε τις τρεις αυτές τεχνικές για να δείξουμε ότι το κόστος αυτό είναι εξαιρετικά μικρό.

Η Τεχνική του Αθροίσματος

Η Τεχνική του Αθροίσματος υπολογίζει ένα άνω όριο $T(n)$ στο συνολικό κόστος μίας ακολουθίας n πράξεων (ένα άθροισμα), και έπειτα υπολογίζει το μέσο κόστος $\frac{T(n)}{n}$.

Στην περίπτωση του σωρού, έστω ότι $T(n)$ είναι το κόστος της ακολουθίας των n πράξεων P. Παρατηρούμε ότι κάθε φορά που εισάγεται ένα στοιχείο στο σωρό με μία πράξη Push μπορεί να εξαχθεί από αυτόν με μία πράξη Pop ή MultiPop. Αυτό σημαίνει ότι σε μία ακολουθία n πράξεων μπορούμε να έχουμε το πολύ n πράξεις Push μαζί με το πολύ n πράξεις Pop (είτε αυτές είναι από μία πράξη Pop είτε από μία πράξη MultiPop). Επομένως, το κόστος για την ακολουθία των n πράξεων είναι $T(n)$=O(n). Άρα, το επιμερισμένο κόστος και για τις τρεις πράξεις είναι $\frac{T(n)}{n}$=O(1).

Αυτή η τεχνική αν και φαίνεται απλή, χρησιμοποιείται μόνο σε πολύ απλές περιπτώσεις αφού δεν είναι πάντα εφικτό να μετρήσουμε το ακριβές κόστος όλων των ενδιάμεσων σταδίων μέσω των οποίων διέρχεται η δομή κατά τη διάρκεια της ακολουθίας πράξεων. Για αυτό το λόγο δεν θα επεκταθούμε περαιτέρω σε αυτή.

Η Τεχνική του Λογιστή

Η Τεχνική του Λογιστή προσδίδει ένα συγκεκριμένο κόστος, το *επιμερισμένο κόστος*, σε κάθε πράξη ανεξάρτητα από το πραγματικό τους κόστος. Όταν το πραγματικό κόστος μιας πράξης είναι μικρότερο από το επιμερισμένο κόστος, τότε η διαφορά δίνεται σε συγκεκριμένα στοιχεία μέσα στη δομή δεδομένων ως *πίστωση*. Όταν το πραγματικό κόστος είναι μεγαλύτερο από το επιμερισμένο κόστος, τότε χρησιμοποιείται η αποθηκευμένη πίστωση από κάποια από τα στοιχεία της δομής για να καλυφθεί το κόστος. Προφανώς θα πρέπει το επιμερισμένο

κόστος μίας πράξης να επιλεγεί έτσι ώστε οι πιστώσεις να μην γίνονται ποτέ αρνητικές. Έτσι εγγυόμαστε ότι το επιμερισμένο κόστος μίας οποιασδήποτε ακολουθίας πράξεων δεν θα είναι ποτέ μικρότερο από το πραγματικό της κόστος και επομένως το συνολικό επιμερισμένο κόστος μίας ακολουθίας πράξεων φράσσει το πραγματικό της κόστος.

Το επιμερισμένο κόστος μιας πράξης δίνεται από το άθροισμα του πραγματικού κόστους της πράξης αυξημένο κατά την πίστωση που καταθέτει ή αποσύρει από κάποια στοιχεία μέσα από τη δομή.

Εφαρμόζοντας τη συγκεκριμένη τεχνική στο σωρό, αφού το κάθε στοιχείο μπορεί να λάβει μέρος μόνο σε δύο πράξεις, μίας εισαγωγής στο σωρό και μίας εξαγωγής από το σωρό, το επιμερισμένο κόστος ac των πράξεων καθορίζεται ως εξής:

- $ac(\text{Push}(\mathcal{S}, x)) = 2$
- $ac(\text{Pop}(\mathcal{S})) = 0$
- $ac(\text{MultiPop}(\mathcal{S}, k)) = 0$

Στην ουσία, χρεώνουμε την πράξη Push τόσο για την εισαγωγή του στοιχείου όσο και για τη διαγραφή του από μία Pop ή MultiPop. Επομένως, οι δύο τελευταίες πράξεις έχουν μηδενική χρέωση.

Όταν εισάγουμε ένα στοιχείο με Push στο σωρό, τότε το επιμερισμένο κόστος είναι 2, από το οποίο πληρώνεται το πραγματικό κόστος που είναι 1, ενώ πιστώνεται το στοιχείο που εισήχθηκε με 1 μονάδα. Όταν το στοιχείο αυτό θα εξαχθεί με Pop ή MultiPop, τότε από την πίστωσή του πληρώνεται η πράξη εξαγωγής. Επομένως, με βάση την προηγούμενη κατανομή του επιμερισμένου κόστους, καλύπτεται το πραγματικό κόστος όλων των πράξεων. Οι πιστώσεις ποτέ δεν γίνονται αρνητικές, και επομένως το επιμερισμένο κόστος μίας ακολουθίας πράξεων φράσσει το πραγματικό κόστος αυτής της ακολουθίας. Άρα, για n πράξεις το επιμερισμένο κόστος είναι το πολύ $2n$ και τελικά το επιμερισμένο κόστος κάθε πράξης είναι $\frac{2n}{n}$=2=O(1).

Η δυσκολία αυτής της τεχνικής έγκειται στη σωστή αντιστοίχιση των επιμερισμένων κοστών σε κάθε πράξη.

Η Τεχνική του Φυσικού

Στην Τεχνική του Λογιστή οι πιστώσεις αντιστοιχούσαν σε προπληρωμένη εργασία που έχει γίνει για κάθε στοιχείο της δομής δεδομένων. Στη Τεχνική του Φυσικού, οι πιστώσεις αντιστοιχούν στο δυναμικό της δομής δεδομένων ώστε να πληρώσει μελλοντικές πράξεις. Αν μία πράξη είναι ακριβή, τότε το δυναμικό

θα μειωθεί αρκετά ώστε να καλυφθεί το κόστος της, ενώ αν είναι φθηνή, τότε το δυναμικό θα αυξηθεί ώστε να καλύψει τις μελλοντικές ακριβές πράξεις. Το δυναμικό λοιπόν προσάπτεται σε όλη τη δομή δεδομένων και όχι σε κάθε στοιχείο της δομής ξεχωριστά και υποδηλώνει την ευχέρεια της δομής να εκτελέσεις ακριβές πράξεις.

Πιο συγκεκριμένα, έστω η ακολουθία πράξεων $P = p_1, p_2, \ldots, p_n$ και έστω ότι με $\mathcal{D}_i$ αναπαριστούμε τη δομή έπειτα από την εφαρμογή της πράξης p_i. Έστω $\mathcal{D}_0$ η αρχική κατάσταση της δομής. Η συνάρτηση δυναμικού Φ αντιστοιχεί κάθε δομή $\mathcal{D}_i$ σε έναν πραγματικό αριθμό $\Phi(\mathcal{D}_i)$ που είναι το δυναμικό της δομής δεδομένων $\mathcal{D}_i$.

$$\Phi : \mathcal{D}_i \to \Re$$

Το επιμερισμένο κόστος ac_i της i-οστής πράξης ισούται με το πραγματικό της κόστος c_i αυξημένο κατά τη μεταβολή του δυναμικού της δομής. Δηλαδή:

$$ac_i = c_i + \Phi(\mathcal{D}_i) - \Phi(\mathcal{D}_{i-1}) \tag{11.1}$$

Το συνολικό επιμερισμένο κόστος για n πράξεις είναι:

$$\sum_{i=1}^{n} ac_i = \sum_{i=1}^{n} (c_i + \Phi(\mathcal{D}_i) - \Phi(\mathcal{D}_{i-1}))$$

και αφού το άθροισμα των Φ είναι τηλεσκοπικό:

$$\sum_{i=1}^{n} ac_i = \sum_{i=1}^{n} c_i + \Phi(\mathcal{D}_n) - \Phi(\mathcal{D}_0)$$

το οποίο σημαίνει ότι το επιμερισμένο κόστος είναι ίσο με το πραγματικό κόστος αυξημένο κατά τη μεταβολή του δυναμικού από την $\mathcal{D}_0$ στην $\mathcal{D}_n$. Αν θέλουμε το επιμερισμένο κόστος να φράσσει το πραγματικό κόστος, θα πρέπει να ισχύει $\Phi(\mathcal{D}_n) \geq \Phi(\mathcal{D}_0)$. Μάλιστα, γενικώς θα πρέπει να ισχύει $\Phi(\mathcal{D}_i) \geq \Phi(\mathcal{D}_0)$, αφού γενικά δεν γνωρίζουμε το μήκος της ακολουθίας πράξεων. Συνήθως $\Phi(\mathcal{D}_0) = 0$ και επομένως θα πρέπει απλώς να ισχύει $\Phi(\mathcal{D}_i) \geq 0$.

Ας δούμε πως εφαρμόζεται η συγκεκριμένη τεχνική στην περίπτωση του σωρού. Ορίζουμε τη συνάρτηση $\Phi(\mathcal{S})$ να είναι ίση με τον αριθμό των στοιχείων που είναι αποθηκευμένα στο $\mathcal{S}$. Αρχικά ο σωρός $\mathcal{S}$ είναι κενός, οπότε $\Phi(\mathcal{S}_0) = 0$. Αν ο σωρός $\mathcal{S}_i$ έχει (μετά την πράξη p_i) ℓ στοιχεία, τότε το δυναμικό είναι θετικό αφού $\Phi(\mathcal{S}_i) = \ell > 0$ και άρα ισχύει $\Phi(\mathcal{S}_i) \geq 0$ για κάθε i.

Προχωρούμε στην ανάλυση των πράξεων. Έστω ότι πριν την εκτέλεση της i-οστής πράξης υπάρχουν ℓ στοιχεία μέσα στο σωρό $\mathcal{S}_{i-1}$. Αν η i-οστή πράξη

είναι η $\text{Push}(\mathcal{S}_{i-1}, x)$, τότε η μεταβολή του δυναμικού είναι:

$$\Phi(\mathcal{S}_i) - \Phi(\mathcal{S}_{i-1}) = \ell + 1 - \ell = 1$$

και επομένως το επιμερισμένο κόστος ac_i^{pu} της $\text{Push}(\mathcal{S}, x)$ θα είναι:

$$ac_i^{pu} = c_i^{pu} + 1 = 1 + 1 = 2$$

όπου c_i^{pu} είναι το πραγματικό της κόστος. Το επιμερισμένο κόστος λοιπόν είναι $\text{O}(1)$.

Αντίστοιχα για την $\text{Pop}(\mathcal{S})$ η μεταβολή του δυναμικού είναι:

$$\Phi(\mathcal{S}_i) - \Phi(\mathcal{S}_{i-1}) = \ell - 1 - \ell = -1$$

και επομένως το επιμερισμένο κόστος ac_i^{po} της $\text{Pop}(\mathcal{S})$ θα είναι:

$$ac_i^{po} = c_i^{po} - 1 = 1 - 1 = 0$$

όπου c_i^{po} είναι το πραγματικό της κόστος. Άρα, το επιμερισμένο κόστος είναι $\text{O}(1)$.

Τέλος, για την $\text{Multipop}(\mathcal{S}, k)$ η μεταβολή του δυναμικού είναι:

$$\Phi(\mathcal{S}_i) - \Phi(\mathcal{S}_{i-1}) = \ell - k - \ell = -k$$

αφού από την $\mathcal{S}_{i-1}$ αφαιρούμε k στοιχεία συνολικά για να καταλήξουμε στην $\mathcal{S}_i$, και επομένως το επιμερισμένο κόστος ac_i^{mp} της $\text{Multipop}(\mathcal{S}, k)$ θα είναι:

$$ac_i^{mp} = c_i^{mp} - k = k - k = 0$$

όπου c_i^{mp} είναι το πραγματικό της κόστος. Συνεπώς, το επιμερισμένο κόστος είναι $\text{O}(1)$.

Επομένως αποδείξαμε και με τη μέθοδο του φυσικού ότι όλες οι πράξεις στο σωρό $\mathcal{S}$ έχουν επιμερισμένο κόστος $\text{O}(1)$, παρότι κάποιες από αυτές τις πράξεις μπορούν να έχουν πολύ μεγαλύτερο κόστος.

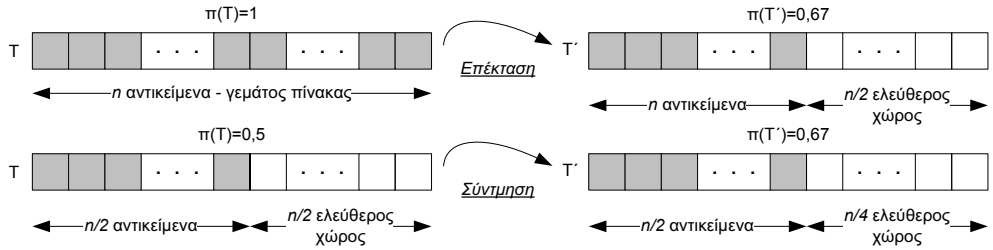
11.3 Δυναμικοί Πίνακες

Σε πολλές περιπτώσεις (για παράδειγμα, στον κατακερματισμό με ανοικτή διεύθυνση, δες Κεφάλαιο 8.9) δεν γνωρίζουμε εκ των προτέρων το πλήθος των στοιχείων που πρέπει να αποθηκευθούν σε έναν πίνακα. Αυτό γίνεται γιατί υποστηρίζονται πράξεις εισαγωγής νέων στοιχείων στον πίνακα, καθώς και πράξεις διαγραφής

στοιχείων από τον πίνακα. Προσοχή όμως γιατί δεν μας ενδιαφέρουν οι λεπτομέρειες υλοποίησης των δύο αυτών πράξεων, παρά μόνο οι επιπτώσεις τους σε σχέση με τον ελεύθερο χώρο του πίνακα. Το μέγεθος του πίνακα εξ ορισμού είναι στατικό και δεν μεταβάλλεται αφού το λειτουργικό σύστημα δεσμεύει συνεχόμενες θέσεις μνήμης για αυτόν και δεν προσφέρει καμία εγγύηση όσον αφορά στη μεταβολή του μεγέθους του. Θεωρούμε, λοιπόν, ότι το κόστος δέσμευσης μνήμης είναι $O(1)$. Για αυτό ακριβώς το λόγο θα πρέπει να υποστηρίξουμε ένα μηχανισμό αντιγραφής ενός πίνακα σε ένα νέο πίνακα είτε όταν ο αρχικός πίνακας δεν έχει άλλες θέσεις είτε όταν είναι πολύ κενός.

Στην προκειμένη περίπτωση έχουμε δύο διαφορετικά φαινόμενα που πρέπει να ισορροπήσουμε. Από τη μία, αν επιλέξουμε μεγάλο μέγεθος πίνακα ώστε να έχουμε πολύ χώρο ελεύθερο, τότε η σπατάλη χώρου είναι μεγάλη. Από την άλλη, αν ο πίνακας είναι πάντοτε πλήρης, τότε η εισαγωγή ενός νέου στοιχείου απαιτεί τη δημιουργία ενός νέου πίνακα με μέγεθος κατά 1 μεγαλύτερο, ώστε να χωρέσει το νέο στοιχείο. Σε αυτή τη περίπτωση δεν έχουμε σπατάλη χώρου αλλά σε κάθε εισαγωγή νέου στοιχείου υποχρεούμαστε να αντιγράφουμε όλο τον πίνακα.

Η διαδικασία *επέκτασης* ενός πίνακα T επιτάσσει τη δημιουργία ενός νέου μεγαλύτερου πίνακα T' όπου αντιγράφονται τα στοιχεία του T. Έπειτα από τη διαδικασία επέκτασης ο T απορρίπτεται, ενώ ο T' θα έχει χώρο για νέα στοιχεία. Η διαδικασία *σύντμησης* ενός πίνακα T επιτάσσει τη δημιουργία ενός νέου μικρότερου πίνακα T' όπου αντιγράφονται τα στοιχεία του T. Έπειτα από τη διαδικασία σύντμησης ο T απορρίπτεται ενώ ο T' δεν θα έχει τόσο μεγάλη σπατάλη χώρου όπως ο T. Στο Σχήμα 11.2 φαίνονται οι διαδικασίες επέκτασης και σύντμησης. Τα γεμάτα κελιά είναι σκιασμένα, ενώ τα λευκά είναι κενά. Επιπλέον, επάνω από κάθε πίνακα φαίνεται η πληρότητά του.



Σχήμα 11.2: Οι πράξεις επέκτασης και σύντμησης σε έναν πίνακα.

Η λύση σε αυτό το πρόβλημα είναι να συνδυάσουμε αυτές τις δύο ακραίες λύσεις. Έστω ένας πίνακας T με μέγεθος n, ο οποίος περιέχει m στοιχεία.

Ορίζουμε την *πληρότητα* $\pi(T)$ του πίνακα T ως το λόγο των στοιχείων στον πίνακα T προς το συνολικό του χώρο, δηλαδή $\pi(T) = \frac{m}{n}$ (δες Σχήμα 11.2). Απαιτούμε η πληρότητα ενός πίνακα να βρίσκεται μέσα σε ένα διάστημα, ώστε να εξασφαλίσουμε ότι ο πίνακας ούτε θα είναι πολύ κενός αλλά δεν θα είναι και πολύ πλήρης. Επομένως θα πρέπει:

$$\alpha \leq \pi(T) \leq \beta$$

όπου $\alpha < \beta \leq 1$. Αν $\pi(T) = \beta$ τότε ο πίνακας έχει υπερχειλίσει και τον επεκτείνουμε δημιουργώντας ένα νέο πίνακα T' μεγέθους $\frac{2\beta}{\alpha+\beta}n$. Αντιστοίχως, όταν $\pi(T) = \alpha$, τότε ο πίνακας T' είναι αραιός και θα πρέπει να συντμηθεί σε έναν πίνακα T' μεγέθους $\frac{2\alpha}{\alpha+\beta}n$. Η επιλογή αυτή ως προς το νέο μέγεθος του πίνακα γίνεται ώστε η πληρότητα του νέου πίνακα να ισαπέχει από τις δύο ακραίες καταστάσεις που αναπαρίστανται από τα α και β, δηλαδή να ισχύει $\pi(T') = \frac{\alpha+\beta}{2}$ αμέσως μετά την επέκταση ή τη σύντμηση.

Η πολυπλοκότητα χειρότερης περίπτωσης για κάθε πράξη στον πίνακα (εισαγωγή ή διαγραφή στοιχείου) ως προς τη διαχείριση του πίνακα είναι $O(n)$ αφού στη χειρότερη περίπτωση θα πρέπει να αντιγράψουμε έναν πίνακα μεγέθους $O(n)$ σε κάποιον άλλον. Ποιο είναι όμως το επιμερισμένο κόστος? Θα χρησιμοποιήσουμε τις τεχνικές του λογιστή και του φυσικού για να το υπολογίσουμε για την περίπτωση όπου $\alpha = 0,5$ και $\beta = 1$, δηλαδή για την περίπτωση όπου ο πίνακας μπορεί να είναι από μισοάδειος μέχρι τελείως γεμάτος.

Η Τεχνική του Λογιστή

Ξεκινώντας με την Τεχνική του Λογιστή, η ιδέα είναι ότι ακριβώς πριν από κάθε πράξη σύντμησης ή επέκτασης κάθε κελί θα έχει πίστωση που στο σύνολό της θα είναι ικανή να καλύψει αυτήν την ακριβή πράξη. Αυτή η πίστωση σε κάθε κελί του πίνακα T προκύπτει από τις πράξεις διαγραφής ή εισαγωγής ενός στοιχείου. Συγκεκριμένα, θέτουμε τα εξής επιμερισμένα κόστη για κάθε πράξη:

- $ac(\text{Delete}) = 3$
- $ac(\text{Insert}) = 5$

Πώς προέκυψαν όμως αυτά τα κόστη? Η ιδέα βασίζεται στο γεγονός ότι ο πίνακας T μεγέθους n που προήλθε από μία σύντμηση ή επέκταση θα χρειασθεί τουλάχιστον $\frac{n}{4}$ πράξεις εισαγωγής ή διαγραφής πριν επανασυντμηθεί ($m = \frac{n}{2}$) ή επαναεπεκταθεί ($m = n$). Επομένως, σε αυτό το διάστημα θα πρέπει η συνολική πίστωση να μπορεί να καλύψει τη σύντμηση ή την επέκταση.

Για την πράξη της εισαγωγής, διακρίνουμε δύο περιπτώσεις. Στην πρώτη περίπτωση, μετά την εισαγωγή ισχύει $m < n$, και επομένως δεν απαιτείται

επέκταση. Σε αυτήν την περίπτωση το πραγματικό κόστος της πράξης είναι 1 και απομένει πίστωση για την εισαγωγή ίση με 4, από την οποία 1 προσάπτεται στο κελί που έγινε η εισαγωγή, ενώ τα υπόλοιπα 3 προσάπτονται σε κελιά που έχουν μηδενική πίστωση εισαγωγής (ανεξάρτητα αν έχουν στοιχεία ή όχι). Όταν $m = n$, τότε απαιτείται να γίνει επέκταση του πίνακα σε ένα νέο πίνακα T' μεγέθους $\frac{4n}{3}$. Το κόστος για την αντιγραφή του T στον T' είναι ίσο με n (το πλήθος των στοιχείων του T) και καλύπτεται από τις πιστώσεις που έχουν τα στοιχεία του T. Οι πιστώσεις φθάνουν αφού απαιτούνται τουλάχιστον $\frac{n}{4}$ πράξεις εισαγωγής όπου η κάθε μία παρέχει πίστωση 4, που σημαίνει συνολική πίστωση ίση με n (όσο δηλαδή και το κόστος της επέκτασης). Το σύνολο των πιστώσεων στον T' είναι 0. Επομένως, οι πιστώσεις ποτέ δεν γίνονται αρνητικές, και άρα το επιμερισμένο κόστος μίας ακολουθίας πράξεων φράσσει το πραγματικό κόστος αυτής της ακολουθίας.

Για την πράξη της διαγραφής, διακρίνουμε πάλι δύο περιπτώσεις. Όταν $m > \frac{n}{2}$, από το επιμερισμένο κόστος πληρώνουμε το πραγματικό κόστος που είναι ίσο με 1 και το υπόλοιπο 2 κατανέμεται στα πρώτα (από αριστερά) κελιά του πίνακα T που έχουν μηδενική πίστωση. Όταν $m = \frac{n}{2}$, τότε δημιουργούμε ένα νέο πίνακα T' μεγέθους $\frac{3n}{4}$ και αντιγράφουμε όλα τα στοιχεία του T στον T'. Το κόστος της σύντμησης είναι $\frac{n}{2}$ (όσα και τα στοιχεία του T) και καλύπτονται χρησιμοποιώντας τις πιστώσεις των πρώτων $\frac{n}{2}$ κελιών. Ο νέος πίνακας T' έχει σύνολο πιστώσεων ίσο με 0. Επομένως, οι πιστώσεις ποτέ δεν γίνονται αρνητικές, και άρα το επιμερισμένο κόστος μίας ακολουθίας πράξεων φράσσει το πραγματικό κόστος αυτής της ακολουθίας.

Προηγουμένως, θεωρήσαμε ότι οι ακολουθίες χειρότερης περίπτωσης είναι όταν έχουμε είτε συνεχόμενες εισαγωγές είτε συνεχόμενες διαγραφές. Αυτό είναι λογικό συμπέρασμα αν κανείς λάβει υπόψη του ότι σε μία ακολουθία που κυριαρχείται από διαγραφές, οι εισαγωγές απλώς καθυστερούν τη σύντμηση και δίνουν επιπλέον πιστώσεις, ενώ το ίδιο ισχύει και στην περίπτωση ακολουθίας που κυριαρχείται από εισαγωγές. Επομένως, ακολουθίες που περιέχουν και τις δύο πράξεις είναι σίγουρα βολικότερες σε σχέση με ακολουθίες που αποτελούνται μόνο από εισαγωγές ή διαγραφές.

Συμπερασματικά, για μία ακολουθία n πράξεων, το μέγιστο επιμερισμένο κόστος είναι $\max\{3, 5\}n = 5n$, και άρα το επιμερισμένο κόστος για κάθε πράξη είναι $\frac{5n}{n}$=O(1).

Η Τεχνική του Φυσικού

Θα αποδώσουμε μία συνάρτηση δυναμικού στον πίνακα T που να αναπαριστά τη δυνατότητα του T να πληρώσει ακριβές πράξεις. Πρώτα θα συζητήσουμε

τη διαίσθηση πίσω από την επιλογή αυτής της συνάρτησης. Θα θέλαμε αυτή η συνάρτηση να είναι 0, όταν ο πίνακας μεγέθους n έχει πληρότητα ακριβώς ίση με $\frac{3}{4}$. Αν η πληρότητα τείνει είτε προς το $\frac{1}{2}$ είτε προς το 1, τότε η συνάρτηση να έχει τέτοια τιμή που να μπορεί να καλύψει το κόστος για την επέκταση ή τη σύντμηση του πίνακα.

Έστω ότι με T_i αναπαριστούμε τον πίνακα έπειτα από την i-οστή πράξη σε αυτόν και έστω ότι m_i είναι το πλήθος των στοιχείων που αποθηκεύει. Η επόμενη συνάρτηση $\Phi(\mathcal{D}_i)$ για τον πίνακα T_i εγγυάται ότι το επιμερισμένο κόστος πάντα θα καλύπτει το πραγματικό κόστος – από εδώ και στο εξής θα αναπαριστούμε με Φ_i το $\Phi(\mathcal{D}_i)$:

$$\Phi_i = \begin{cases} 4 \cdot \left(m_i - \frac{3n}{4}\right) & \text{αν } m_i > \frac{3n}{4} \\ 2 \cdot \left(\frac{3n}{4} - m_i\right) & \text{αν } m_i < \frac{3n}{4} \\ 0 & \text{αν } m_i = \frac{3n}{4} \end{cases}$$

Η συνάρτηση δυναμικού ποτέ δεν γίνεται αρνητική και επομένως το επιμερισμένο κόστος που θα υπολογίσουμε αποτελεί άνω φράγμα στο πραγματικό κόστος των πράξεων.

Θα μιλήσουμε ξεχωριστά για κάθε πράξη. Αν η i-οστή πράξη είναι η εισαγωγή και το επιμερισμένο κόστος της είναι ac_i^{ins}, ενώ το πραγματικό της κόστος είναι c_i^{ins}, τότε διακρίνουμε δύο περιπτώσεις:

1. $m_i < n$. Σε αυτή την περίπτωση ο πίνακας δεν χρειάζεται επέκταση. Αν $m_i > \frac{3n}{4}$, τότε το επιμερισμένο κόστος είναι:

   $$ac_i^{ins} = c_i^{ins} + \Phi_i - \Phi_{i-1} = 1 + 4\left(m_{i-1} + 1 - \frac{3n}{4}\right) - 4\left(m_{i-1} - \frac{3n}{4}\right) = 1 + 4 = 5$$

   ενώ αν $m_i < \frac{3n}{4}$, τότε το επιμερισμένο κόστος είναι:

   $$ac_i^{ins} = c_i^{ins} + \Phi_i - \Phi_{i-1} = 1 + 2\left(\frac{3n}{4} - m_{i-1} - 1\right) - 2\left(\frac{3n}{4} - m_{i-1}\right) = 1 - 2 = -1$$

   Το αρνητικό επιμερισμένο κόστος δηλώνει ότι αν $m_i < \frac{3n}{4}$, τότε μία εισαγωγή απομακρύνει από τη σύντμηση και άρα είναι βολικό για τον πίνακα.

2. $m = n$. Σε αυτή την περίπτωση ο πίνακας χρειάζεται επέκταση. Το επιμερισμένο κόστος είναι:

   $$ac_i^{ins} = c_i^{ins} + \Phi_i - \Phi_{i-1} = (n+1) + 4\left(n - \frac{3}{4}\frac{4n}{3}\right) - 4\left(n - 1 - \frac{3n}{4}\right) = 5$$

όσο δηλαδή και το επιμερισμένο κόστος μίας απλής εισαγωγής χωρίς επέκταση.

Υποθέτοντας ότι η i-οστή πράξη είναι διαγραφή θεωρούμε ότι το επιμερισμένο κόστος είναι ac_i^{del} και ότι το πραγματικό είναι c_i^{del}. Και σε αυτή την περίπτωση διακρίνουμε δύο περιπτώσεις:

1. $m_i > \frac{n}{2}$. Σε αυτή την περίπτωση ο πίνακας δεν χρειάζεται μείωση. Αν $m_i < \frac{3n}{4}$, τότε το επιμερισμένο κόστος είναι:

$$ac_i^{del} = c_i^{del} + \Phi_i - \Phi_{i-1} = 1 + 2\left(\frac{3n}{4} - (m_{i-1} - 1)\right) - 2\left(\frac{3n}{4} - m_{i-1}\right) = 3$$

ενώ αν $m_i > \frac{3n}{4}$, τότε το επιμερισμένο κόστος είναι:

$$ac_i^{del} = c_i^{del} + \Phi_i - \Phi_{i-1} = 1 + 4\left(m_{i-1} - 1 - \frac{3n}{4}\right) - 4\left(m_{i-1} - \frac{3n}{4}\right) = -3$$

Το επιμερισμένο κόστος είναι αρνητικό αφού μία διαγραφή όταν $m_i > \frac{3n}{4}$ απομακρύνει τον πίνακα από την ακραία περίπτωση όπου είναι τελείως πλήρης.

2. $m_i = \frac{n}{2}$. Σε αυτή την περίπτωση ο πίνακας χρειάζεται σύντμηση. Το επιμερισμένο κόστος είναι:

$$ac_i^{del} = c_i^{del} + \Phi_i - \Phi_{i-1} = \left(\frac{n}{2} + 1\right) + 2\left(\frac{3}{4}\frac{2n}{3} - \frac{n}{2}\right) - 2\left(\frac{3n}{4} - \left(\frac{n}{2} + 1\right)\right) = 3$$

όσο δηλαδή και το επιμερισμένο κόστος μίας απλής διαγραφής χωρίς σύντμηση.

Αφού η συνάρτηση δυναμικού δεν είναι ποτέ αρνητική, προκύπτει ότι το επιμερισμένο κόστος μίας ακολουθίας n πράξεων φράσσει το πραγματικό κόστος της ίδιας ακολουθίας. Επομένως, το μέγιστο κόστος αυτής της ακολουθίας θα είναι $5n$ - αφού στη χειρότερη περίπτωση είναι όλες εισαγωγές με το μεγαλύτερο επιμερισμένο κόστος. Άρα, το επιμερισμένο κόστος κάθε πράξης είναι $\frac{5n}{n}$=O(1).

Επομένως, αν και αυτή η υλοποίηση των δυναμικών πινάκων έχει κόστος χειρότερης περίπτωσης ίσο με $O(n)$ το επιμερισμένο κόστος είναι μόλις $O(1)$ και αυτό γιατί οι ακριβές πράξεις είναι πολύ λιγότερες σε σχέση με το πλήθος των φθηνών πράξεων σε οποιαδήποτε ακολουθία πράξεων.

11.4 Αυτοργανούμενες Δομές Δεδομένων

Σε αυτή την ενότητα θα εξετάσουμε μια κατηγορία δομών που καλούνται αυτοργανούμενες δομές δεδομένων (self-organizing). Οι αυτοργανούμενες δομές αποτελούν κομψές λύσεις στο πρόβλημα του λεξικού (εύρεση - διαγραφή - εισαγωγή), όταν τα στοιχεία του συνόλου S έχουν συχνότητες πρόσβασης οι οποίες μπορούν να μεταβάλλονται κατά τη διάρκεια της εκτέλεσης του αλγορίθμου (on-line). Επειδή τα στοιχεία έχουν συγκεκριμένες συχνότητες πρόσβασης, οι αυτοργανούμενες δομές φροντίζουν να οργανώνουν την αποθήκευση των στοιχείων του S με τρόπο ώστε τα στοιχεία που έχουν μεγάλη συχνότητα πρόσβασης να βρίσκονται κοντά στο σημείο πρόσβασης. Συγκεκριμένα, πρόσβαση στο στοιχείο x_i θα προκαλέσει μετακίνησή του στο σημείο πρόσβασης. Αυτή η τακτική κάνει φθηνότερες τις επόμενες προσβάσεις στο x_i.

Οι αυτοργανούμενες δομές δεν διατηρούν κάποια σαφή πληροφορία σχετικα με τα στοιχεία, όπως η συχνότητα πρόσβασης. Αυτό από τη μια παρέχει μικρότερη επιβάρυνση χώρου αλλά έχει σαν συνέπεια να μην παρέχονται εγγυήσεις για την πολυπλοκότητα κάποιας πράξης στη χειρότερη περίπτωση. Παρόλ" αυτά, η επιμερισμένη πολυπλοκότητα των πράξεων είναι ανταγωνιστική των συνηθισμένων στατικών δομών. Επίσης οι αυτοργανούμενες δομές μπορούν να προσαρμόζονται στις αλλαγές των ιδιοτήτων της εισόδου, έχοντας όμως σαν μειονέκτημα ότι κάθε τους πράξη επιβαρύνεται από τα έξοδα αναδιοργάνωσης της δομής (μετακίνηση στοιχείων στο σημείο πρόσβασης). Ένα βασικό χαρακτηριστικό των αυτοργανούμενων δομών είναι ότι είναι εύκολες στην υλοποίηση.

Να σημειωθεί ότι οι δομές που εξετάζονται, υποστηρίζουν πράξεις λεξικών. Σε αυτή την ενότητα θα εξετάσουμε δύο αυτοργανούμενες δομές: την *αυτοργανούμενη γραμμική λίστα* και τα *αρθρωμένα δέντρα*, μαζί με την επιμερισμένη ανάλυση των πράξεων σε αυτές.

11.4.1 Αυτοργανούμενες Γραμμικές Λίστες

Πρόκειται για πολύ απλές γραμμικές λίστες που αποθηκεύουν στοιχεία του συνόλου $S = \{x_1, \ldots, x_n\}$, υποκείμενα στις πράξεις του λεξικού, $access(x)$, $ins(x)$ και $del(x)$. Οι λίστες μπορούν να υλοποιηθούν με οποιοδήποτε γνωστό τρόπο, με συνδεδεμένη λίστα ή με τη χρήση πίνακα.

Για παράδειγμα, έστω, ότι σε μία λίστα γίνονται αναζητήσεις με διαφορετική πιθανότητα προσπέλασης σε κάθε κόμβο και ότι η πιθανότητα προσπέλασης του i-οστού κόμβου, p_i, είναι γνωστή εκ των προτέρων. Είναι ευνόητο ότι η επίδοση θα βελτιωθεί αν οι κόμβοι διαταχθούν κατά φθίνουσα πιθανότητα επίσκεψης.

Για παράδειγμα, έστω ότι η πιθανότητα επίσκεψης του i-οστού κόμβου μίας

λίστας ισούται με $p_i = 1/2^i$. Τότε με κατάλληλη άλγεβρα προκύπτει ότι η μέση τιμή του αριθμού των συγκρίσεων για μία επιτυχή αναζήτηση είναι:

$$\sum_{i=1}^{n} i \times p_i \;=\; 2$$

Αν υποτεθεί ότι οι κόμβοι αυτοί τοποθετούνται στη λίστα με αύξουσα πιθανότητα προσπέλασης, τότε αποδεικνύεται ότι η μέση τιμή του αριθμού των συγκρίσεων για μία επιτυχή αναζήτηση είναι:

$$\sum_{i=1}^{n} i \times p_{n+1-i} \;=\; n - 1 + \frac{1}{2^{n+1}}$$

Το παράδειγμα αυτό αποδεικνύει το σημαντικό πλεονέκτημα της πρώτης εκδοχής. Μάλιστα αν το μήκος της λίστας είναι μεγάλο, τότε η διαφορά είναι συντριπτική. Ωστόσο, η ερώτηση που τίθεται είναι: "Πώς πρέπει να διαταχθούν οι κόμβοι μίας λίστας, όταν δεν είναι γνωστές εκ των προτέρων οι πιθανότητες προσπέλασης των κόμβων;"

Ορίζουμε ως $pos(i)$ τη θέση του x_i στη λίστα, δηλαδή των αριθμό των στοιχείων που παρεμβάλλονται από το αριστερό άκρο της λίστας, ως το x_i. Καθεμία πράξη προσπελαύνει γραμμικά τα στοιχεία της λίστας από το αριστερό άκρο, μέχρι την εύρεση του κατάλληλου στοιχείου ή το τέλος της λίστας. Συνεπώς, οι πράξεις $access(x_i)$ και $del(x_i)$, κοστίζουν $pos(x_i)$, ενώ η πράξη $ins(x_i)$, έχει κόστος $|S|+1$, μιας και πρέπει να εξασφαλιστεί ότι $x_i \notin S$. Παράλληλα με κάθε πράξη υλοποιείται και μια στρατηγική αυτοργάνωσης της λίστας. Δύο από τις πιο γνωστές στρατηγικές αυτοργάνωσης είναι οι:

Κανόνας Μετακίνησης στην Αρχή: (Move to Front Rule – MFR). Καθεμία από τις πράξεις $access(x)$, $ins(x)$, μετακινεί το x στην αρχή της λίστας χωρίς να μεταβάλλει τη σειρά των υπολοίπων στοιχείων, ενώ η πράξη $del(x)$ διαγράφει το στοιχείο από τη λίστα.

Κανόνας Αντιμετάθεσης: (Transposition Rule – TR). Η πράξη $access(x)$ εναλλάσει το x με το προηγούμενό του στοιχείο, η πράξη $ins(x)$ κάνει το x προτελευταίο στοιχείο της λίστας και η πράξη $del(x)$ διαγράφει το στοιχείο από τη λίστα.

Στη συνέχεια θα αναλύσουμε τη συμπεριφορά του κανόνα μετακίνησης στη αρχή στον οποίο θα αναφερόμαστε για συντομία ως MFR. Ας δούμε μια ακολουθία πράξεων σε λίστα με την MFR στρατηγική οργάνωσης.

$$1\ \ 3\ \ 4 \xrightarrow{\text{ins}(2)} 2\ \ 1\ \ 3\ \ 4 \xrightarrow{\text{access}(3)} 3\ \ 2\ \ 1\ \ 4 \xrightarrow{\text{del}(1)} 3\ \ 2\ \ 4$$

Το κόστος αυτής της ακολουθίας είναι $4 + 3 + 3 = 10$.

Όπως έγινε προφανές από το παράδειγμα, με την αυτοργανούμενη λίστα κάθε πράξη μπορεί να στοιχίσει μέχρι $O(|S|)$. Θα δούμε σε επόμενο Κεφάλαιο ότι η στρατηγική MFR είναι ιδιαίτερα αποτελεσματική ως προς την συνολική πολυπλοκότητα μιας ακολουθίας πράξεων και ότι μάλιστα είναι *το πολύ δύο φορές αργότερη*, από τη βέλτιστη στατική οργάνωση της λίστας. Αυτός ο τύπος ανάλυσης καλείται ανταγωνιστική ανάλυση.

11.4.2 Τα Αρθρωμένα δέντρα

Τα αρθρωμένα δέντρα είναι η δεύτερη περίπτωση αυτοργανούμενης δομής που εξετάζουμε. Πρόκειται για μια αυτοργανούμενη δομή δυαδικού κομβοπροσανατολισμένου δέντρου, η οποία δίνει λύση στο πρόβλημα του *βεβαρυμένου λεξικού* (weighted dictionary). Η ιδιαιτερότητα των αρθρωμένων δέντρων έγκειται στο ότι αποτελούν κομψές λύσεις για το βεβαρυμένο λεξικό, επιδεικνύοντας πολύ καλή επιμερισμένη πολύπλοκότητα για κάθε πράξη χωρίς να αποθηκεύουν τα βάρη κάθε στοιχείου.

Ως στρατηγική επανοργάνωσης επιλέγεται η *εξάρθρωση* ως παραλλαγή της στρατηγικής *Μετακίνησης στη Ρίζα*. Κατά τη διάρκεια κάθε πράξης, το εμπλεκόμενο στοιχείο μεταφέρεται στη ρίζα με διαδοχικές περιστροφές, όπως θα δούμε στη συνέχεια. Αυτή η στρατηγική έχει ως συνέπεια οι επόμενες πράξεις που εμπλέκουν αυτό το στοιχείο να είναι πιο φθηνές γιατί αναμένεται το στοιχείο να βρεθεί κοντά στη ρίζα. Ο χειρότερος χρόνος μιας επί μέρους πράξης μπορεί να είναι $O(|S|)$ αλλά κάθε ο συνολικός χρόνος για μια ακολουθία πράξεων είναι ανταγωνιστικός με στατικές δομές που αποθηκεύουν πληροφορία συχνότητας. Στη συνέχεια θα εξετάσουμε πιο αναλυτικά τη στρατηγική αυτοργάνωσης και θα περιγράψουμε καθεμία από τις επί μέρους πράξεις.

```
TYPE nodeptr=^node;
     node=RECORD
             data:INTEGER; left,right,parent:nodeptr
          END;

PROCEDURE Splay(current:nodeptr);
VAR father:nodeptr;
BEGIN
   father:=current^.parent;
```

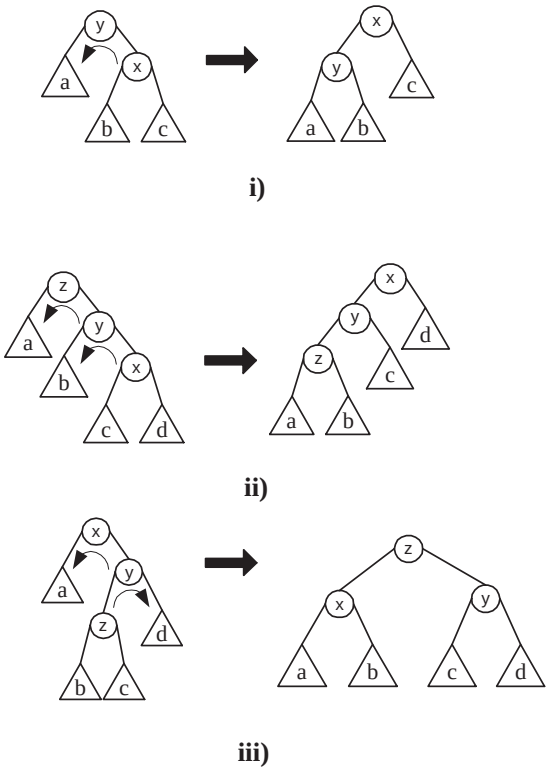
```
   WHILE father<>NILL DO
      BEGIN
         IF father^.parent=NILL THEN SingleRotate(current)
         ELSE DoubleRotate(current)
         father:=current^.parent
      END
END;

PROCEDURE SingleRotate(x:nodeptr);
BEGIN
   IF x^.parent^.left=x THEN ZigLeft(x) ELSE ZigRight(x)
END;

PROCEDURE ZigLeft(x:nodeptr);
VAR p,q:nodeptr;
BEGIN
   p:=x^.parent; q:=x^.right; x^.right:=p; x^.parent:=NIL;
   IF q<>NIL THEN q^.parent:=p;
   p^.left:=q; p^.parent:=x
END;

PROCEDURE ZigZigLeft(x:nodeptr);
VAR p1,q1,p2,q2,gp2:nodeptr;
BEGIN
   p1:=x^.parent; p2:=p1^.parent; q1:=x^.right; q2:=p1^.right;
   gp2:=p2^.parent; x^.right:=p1; p1^.parent:=x;
   p1^.right:=p2; p2^.parent:=p1;
   IF q1<>NIL THEN q1^.parent:=p1; p1^.left:=q1;
   IF q2<>NIL THEN q2^.parent:=p2; p2^.left:=q2;
   x^.parent:=gp2;
   IF gp2<>NIL THEN
      IF gp2^.left=p2 THEN gp2^.left:=x ELSE gp2^.right:=x
END;
```

Έστω ένα στοιχείο $x \in S$. Για τη μεταφορά του x στη ρίζα του δέντρου, επαναλαμβάνουμε συνεχώς τον μετασχηματισμό εξάρθρωσης, μέχρι το x να καταστεί ρίζα. Οι περιπτώσεις που περιγράφονται, απεικονίζονται και στο Σχήμα 11.3. Οι συμμετρικές περιπτώσεις δεν απεικονίζονται.



Σχήμα 11.3: Εφαρμογή εξάρθρωσης στον κόμβο x. **i)** Zig: Τερματική απλή περιστροφή στον x, **ii)** Zig-Zig: Δύο απλές περιστροφές μια στον x και μία στον z, **iii)** Zig-Zag: Διπλή περιστροφή στον x.

ΕΞΑΡΘΡΩΣΗ

Περίπτωση 1(zig): Αν ο $p(x)$, ο πατέρας του x, είναι ρίζα τότε κάνουμε μια απλή περιστροφή στο $p(x)$ και φέρνουμε τον x στη ρίζα. Αυτή η περίπτωση είναι τερματική.

Περίπτωση 2(zig – zig): Αν ο $p(x)$ δεν είναι ρίζα, και οι $p(x)$ και x είναι και οι δύο αριστερά ή δεξιά παιδιά των γονέων τους, τότε εκτελούμε μια *απλή περιστροφή* στον $p(x)$ με τον πατέρα του, $p(p(x))$, ακολουθούμενη από μια *δεύτερη απλή περιστροφή*, της ίδιας φοράς με την πρώτη, στην ακμή

που συνδέει τον x με τον $p(x)$. Η περίπτωση αυτή δεν είναι αναγκαία τερματική.

Περίπτωση 3(zig − zag): Ο $p(x)$ δεν είναι ρίζα, και οι $p(x)$ και x δεν είναι του ίδιου είδους παιδιά, δηλαδή ο $p(x)$ είναι αριστερό παιδί και ο x δεξιό ή το αντίστροφο. Τότε εκτελούμε μια *διπλή περιστροφή* στον x. Η περίπτωση αυτή δεν είναι αναγκαία τερματική.

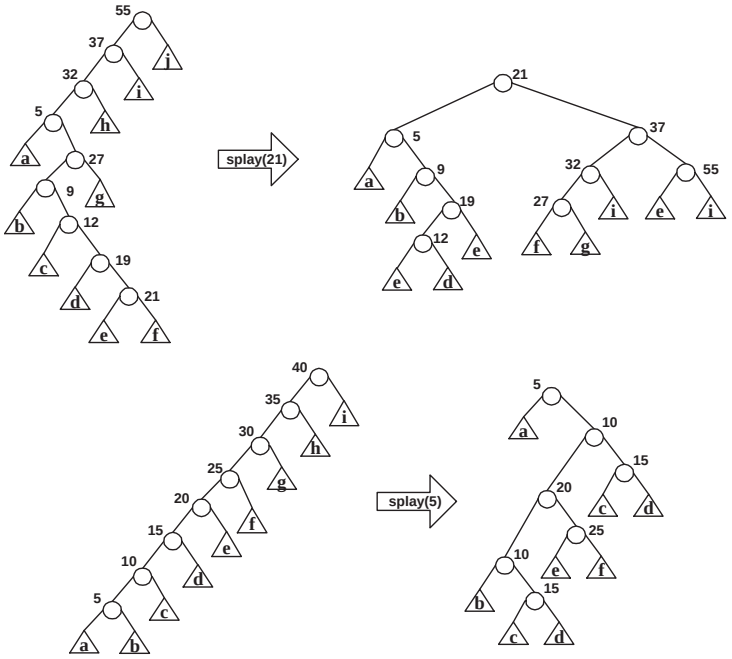
Όλες οι επαναλήψεις του μετασχηματισμού εξάρθρωσης χρειάζονται χρόνο $\Theta(d)$, όπου d είναι το βάθος του κόμβου x. Επιπλέον μετά από κάθε εξάρθρωση, το βάθος κάθε κόμβου στο μονοπάτι που οδηγούσε στον x, μειώνεται περίπου στο μισό, γεγονός που κάνει τις υπόλοιπες προσβάσεις σε όλους τους κόμβους του μονοπατιού αυτού, πιο φθηνές. Παραδείγματα εφαρμογής πράξεων εξάρθρωσης απεικονίζονται στο Σχήμα 11.4.

Θα μελετήσουμε την επιμερισμένη πολυπλοκότητα της πράξης εξάρθρωσης. Γι' αυτό το σκοπό θα χρησιμοποιήσουμε τη μέθοδο του Φυσικού και θα ορίσουμε συνάρτηση δυναμικού για ολόκληρη τη δομή. Θεωρούμε ότι κάθε κόμβος έχει ένα θετικό βάρος $w(i)$. Επίσης για κάθε κόμβο x, ορίζουμε το *μεγεθός* του, $s(x)$, ως το άθροισμα των βαρών, όλων των κόμβων στο υπόδεντρό του και την *τάξη* του, $r(x) = \log s(x)$. Τέλος, ως δυναμικό του δέντρου ορίζουμε το άθροισμα των τάξεων όλων των κόμβων του, δηλαδή $\Phi(T) = \sum_{\forall x \in T} r(x)$. Μιας και οι κυρίαρχες πράξεις που λαμβάνουν χώρα κατά την εξάρθρωση είναι οι περιστροφές, ορίζουμε ως πολυπλοκότητα της πράξης, τον αριθμό των περιστροφών που γίνονται.

Για να πάρουμε μία ιδέα για την επιλογή της συνάρτησης δυναμικού, υπολογίζουμε την Φ σε δύο ακραίες περιπτώσεις. Πρώτα να θυμηθούμε ότι $\int \ln x = x \ln x - x$ και επομένως $\int \log_2 x = x \log_2 x - x/\ln 2$. Στην ακραία μη ζυγισμένη περίπτωση όπου η τάξη του i-οστού κόμβου είναι $2\lfloor \log_2 i \rfloor$ και το δυναμικό του δέντρου T είναι

$$\Phi(T) = 2\sum_{i=1}^{n} \lfloor \log_2 i \rfloor = 2n \log_2 n - O(n)$$

Στην περίπτωση που είναι ζυγισμένο το δέντρο T, φράσσουμε την συνάρτηση Φ από την ποσότητα $2U(n)$, όπου $U(n) = 2U(n/2) + \log_2 n$. Η επίλυση αυτής της αναδρομικής δίνει $U(n) = O(n)$ και επομένως $\Phi(T) = O(n)$.



Σχήμα 11.4: Παραδείγματα εφαρμογής εξάρθρωσης. **i)** Έξάρθρωση στον κόμβο 21. **ii)** Ακραία περίπτωση εξάρθρωσης όπου εφαρμόζονται συνεχώς zig-zig.

Λήμμα 11.1. *Ο επιμερισμένος χρόνος της εφαρμογής της πράξης εξάρθρωσης σε κόμβο x και σε δέντρο με ρίζα t είναι το πολύ* $3(r(t) - r(x)) + 1 = O(\log(s(t)/s(x)))$.

Proof. Υπενθυμίζεται ότι το επιμερισμένο κόστος μιας πράξης σύμφωνα με τη μέθοδο του Φυσικού, δίνεται από τη Σχέση 11.1 και είναι $ac_i = c_i + \Phi(\mathcal{D}_i) - \Phi(\mathcal{D}_{i-1})$.

Έστω τώρα κάποιος κόμβος $v \in T$. Με $s(v)$, $r(v)$ και $s'(v)$, $r'(v)$ θα συμβολίζουμε το μέγεθος και την τάξη του v, πριν και μετά την εξάρθρωση, αντίστοιχα. Έστω y ο πατέρας του x και z ο πατέρας του y (αν υπάρχει) πριν την εξάρθρωση στον x. Αναλύουμε το κόστος καθεμίας περίπτωσης της εξάρθρωσης ξεχωριστά. Χρεώνουμε το πραγματικό κόστος μιας περιστροφής ως ένα.

Περίπτωση 1: Έχουμε μια περίπτωση, οπότε το επιμερισμένο κόστος αυτού του βήματος είναι: $AC = T + \Delta\Phi$.

$$\begin{array}{lll} 1 + r'(x) + r'(y) - r(x) - r(y) & & \text{γιατί αλλάζουν μόνο οι τάξεις των } x \text{ και } y \\ \leq & 1 + r'(x) - r(x) & \text{γιατί } r(y) \geq r'(y) \\ \leq & 1 + 3(r'(x) - r(x)) & \text{γιατί } r'(x) \geq r(x) \Rightarrow r'(x) - r(x) \leq 3(r'(x) - r(x)) \end{array}$$

Περίπτωση 2: Εκτελούνται δύο περιστροφές οπότε το επιμερισμένο κόστος είναι:

$$\begin{array}{lll} 2 + r'(x) + r'(y) + r'(z) & & \\ \quad -r(x) - r(y) - r(z) & & \text{γιατί αλλάζουν μόνο οι τάξεις των } x, y \text{ και } z \\ & & \\ = & 2 + r'(y) + r'(z) - r(x) - r(y) & \text{γιατί } r'(x) = r(z) \\ \leq & 2 + r'(x) + r'(z) - 2r(x) & \text{γιατί } r'(x) \geq r(y) \text{ και } r(y) \geq r(x). \end{array}$$

και θα δείξουμε ότι $2 + r'(x) + r'(z) - 2r(x) \leq 3(r'(x) - r(x)) \Rightarrow 2r'(x) - r(x) - r'(z) \geq 2$. Έχουμε:

$$r'(x) - r(x) + r'(x) - r'(z) \geq 2 \Rightarrow \log \frac{s'(x)}{s(x)} + \log \frac{s'(x)}{s'(z)} \geq 2 \Rightarrow \quad (11.2)$$

Όμως από το Σχήμα 11.3 ισχύει ότι: $s'(x) = s'(z) + s(x) + w(y) \Rightarrow s'(x) \geq s'(z) + s(x)$. Άρα η Σχέση 11.2 γίνεται:

$$\log \frac{s'(z) + s(x)}{s(x)} + \log \frac{s'(z) + s(x)}{s'(z)} \geq 2 \Rightarrow \log \left(1 + \frac{s'(z)}{s(x)}\right) + log\left(1 + \frac{s(x)}{s'(z)}\right) \geq 2 \Rightarrow$$

$$\log \left(1 + \frac{s'(z)}{s(x)} + 1 + \frac{s(x)}{s'(z)}\right) \geq 2 \Rightarrow \log \left(2 + \frac{s'^2(z) + s^2(x)}{s'(z)s(x)}\right) \geq 2$$

Επομένως, αρκεί να δείξουμε ότι $\frac{s'^2(z)+s^2(x)}{s'(z)s(x)} \geq 2$. Πράγματι:

$$s'^2(z) + s^2(x) \geq 2s'(z)s(x) \Rightarrow (s'(z) - s(x))^2 \geq 0$$

που ισχύει πάντα. Άρα, έχουμε αποδείξει ότι το επιμερισμένο κόστος της εξάρθρωσης στη δεύτερη περίπτωση είναι **το πολύ** $3(r'(x) - r(x))$.

$$\begin{aligned}
&2 + r'(x) + r'(y) + r'(z) \\
&\quad -r(x) - r(y) - r(z) \\
&\quad \leq 2 + r'(y) + r'(z) - 2r(x) \quad \text{γιατί } r'(x) = r(z) \text{ και } r(x) \leq r(y).
\end{aligned}$$

Περίπτωση 3: Το επιμερισμένο κόστος σε αυτή την περίπτωση είναι:

όπου θα δείξουμε ότι $2+r'(y)+r'(z)-2r(x) \leq 2(r'(x)-r(x)) \Rightarrow 2r'(x)-r'(y)-r'(z) \geq 2$. Αυτό προκύπτει με τον ίδιο τρόπο που εργαστήκαμε στην Περίπτωση 2, λόγω της ανισότητας $s'(y) + s'(z) \leq s'(x)$. Η επιμερισμένη πολυπλοκότητα και σε αυτή την περίπτωση είναι **το πολύ** $2(r'(x)-r(x)) \leq 2(r'(x) - r(x))$.

Μέχρι στιγμής έχουμε δώσει φράγμα για το επιμερισμένο κόστος κάθε βήματος εξάρθρωσης. Το επιμερισμένο κόστος για ολόκληρη την μετακίνηση του x από τα φύλλα στη ρίζα δίνεται από το άθροισμα του επιμερισμένου κόστους κάθε τοπικού μετασχηματισμού. Ας δούμε δύο διαδοχικούς τέτοιους μετασχηματισμούς, τον k και τον $k+1$. Παρατηρούμε ότι $r'_k(x) = r_{k+1}(x)$, δηλαδή ότι η τάξη του x μετά την εφαρμογή του $k-$στού μετασχηματισμού, ισούται με την τάξη του x πριν την εφαρμογή της $(k+1)$-στης πράξης.

Η ακολουθία με το μέγιστο συνολικό επιμερισμένο κόστος είναι, ακολουθία zig-zig και zig-zag με οποιαδήποτε σειρά, τερματιζόμενη με ένα zig. Τότε το συνολικό επιμερισμένο κόστος θα είναι:

$$\begin{aligned}
ac_{spl}(x) &= 3(r'_0(x) - r_0(x)) + 3(r'_1(x) - r_1(x)) + \cdots + 3(r'_k(x) - r_k(x)) + 1 \\
&= 3(r_1(x) - r_0(x)) + 3(r_2(x) - r_1(x)) + \cdots + 3(r_{k+1}(x) - r_k(x)) + 1 \\
&\quad \cdots (\text{τηλεσκοπικό άθροισμα}) \\
&= 3(r_{k+1}(x) - r_0(x)) + 1
\end{aligned}$$

όπου στην τελευταία γραμμή το $r_0(x)$ αντιστοιχεί στην τάξη του x στην αρχική του θέση ενώ το $r_{k+1}(x)$ αντιστοιχεί στην τάξη του x αφού έχει μεταφερθεί στη ρίζα, οπότε η τελευταία γραμμή μπορεί να γραφεί ως $3(r'(t) - r(x)) + 1$, ολοκληρώνοντας την απόδειξη. □

Ας δούμε τώρα τις πράξεις που υποστηρίζει το αρθρωμένο δέντρο με τις πολυπλοκότητές τους.

$access(x, t)$:

Πρόσβαση του x στο t. Υλοποιείται με συνήθη προσπέλαση από τη ρίζα προς τον κόμβο x ακολουθούμενη από εξάρθρωση που μεταφέρει τον x στη ρίζα. Αν το x δεν υπάρχει στο δέντρο, τότε εκτελείται εξάρθρωση στον κόμβο y, τον τελευταίο κόμβο που επισκέφθηκε η αναζήτηση, πριν καταλήξει στα φύλλα, οπότε συμπεραίνουμε ότι το x δεν υπάρχει στο δέντρο.

$Join(t_1, t_2)$: (Συνένωση δέντρων)

Συνένωση των δέντρων t_1 και t_2, σε ένα ενιαίο δέντρο, καταστρέφοντας τα παλιά δέντρα. Η πράξη αυτή υλοποιείται ως εξής:

- Έστω m το μεγαλύτερο στοιχείο του t_1. Εκτελείται $access(m, t_1)$ (ουσιαστικά ακολουθείται το δεξιότερο μονοπάτι από τη ρίζα στα φύλλα). Μετά την πρόσβαση το t_1 έχει τον m στη ρίζα και ο m δεν έχει δεξί παιδί.
- Η ρίζα του t_2 γίνεται δεξιό παιδί του m.

$Split(i, t)$: (Διαχωρισμός)

Γίνεται διαχωρισμός του t σε δύο δέντρα t_1 και t_2, με το t_1 να περιέχει όλα τα μικρότερα ή ίσα του i και το t_2, τα μεγαλύτερα του i, στοιχεία. Η πράξη υλοποιείται ως εξής:

- Εκτελείται $access(i, t)$. Το i μπορεί να μην υπάρχει στο δέντρο, οπότε η πρόσβαση μπορεί να φέρει στη ρίζα του t ένα στοιχείο μεγαλύτερο του i.
- Αν το στοιχείο που είναι στη ρίζα είναι το i ή μικρότερο του i, το t_1 αποτελείται από την τρέχουσα ρίζα και το αριστερό υπόδεντρό της και το t_2 από το δεξί υπόδεντρο της ρίζας, διαφορετικά το t_1 αποτελείται μόνο από το αριστερό υπόδεντρο και το υπόλοιπο κομμάτι αποτελεί το t_2.

$ins(i, t)$:

Υλοποιείται ως εξής:

- Πρώτα εκτελείται $Split(i, t)$ του t σε t_1 και t_2
- Στη συνέχεια δημιουργείται ένας νέος κόμβος, λαμβάνει την τιμή i και το νέο δέντρο έχει ως ρίζα τον i, αριστερό υπόδεντρο το t_1 και δεξί το t_2.

$del(i, t)$:

Υλοποιείται ως εξής:

- Πρώτα εκτελείται $access(i,t)$ και ο i έρχεται στη ρίζα.
- Κατόπιν διαγράφεται η ρίζα του t και προκύπτουν δύο υπόδεντρα t_1 και t_2
- Το νέο δέντρο προκύπτει μετά την πράξη $Join(t_1,t_2)$.

Στη συνέχεια θα δούμε τις πολυπλοκότητες κάθε πράξης. Έστω στοιχείο i στο δέντρο και $i-$, $i+$ τα στοιχεία που βρίσκονται πριν και μετά από το i σε συμμετρική διάταξη. Το παρακάτω Θεώρημα συνοψίζει την απόδοση των πραξεων στα αρθρωμένα δέντρα.

Θεώρημα 11.1. *Έστω W το άθροισμα των βαρών, όλων των στοιχείων του δέντρου. Τότε οι επιμερισμένες πολυπλοκότητες για κάθε πράξη είναι οι εξής:*

$$Access(i,t): \begin{cases} 3\log\left(\frac{W}{w(i)}\right)+1 & \text{αν το } i \text{ βρίσκεται στο } t \\ 3\log\left(\frac{W}{\min\{w(i-),w(i+)\}}\right)+1 & \text{αν το } i \text{ δεν βρίσκεται στο } t. \end{cases}$$

$$Join(t_1,t_2): \quad 3\log\left(\frac{W}{w(i)}\right)+O(1), \qquad \text{όπου } i \text{ είναι το μεγαλύτερο στοιχείο του } t_1.$$

$$Split(i,t): \begin{cases} 3\log\left(\frac{W}{w(i)}\right)+O(1) & \text{αν το } i \text{ βρίσκεται στο } t \\ 3\log\left(\frac{W}{\min\{w(i-),w(i+)\}}\right)+O(1) & \text{αν το } i \text{ δεν βρίσκεται στο } t. \end{cases}$$

$$ins(i,t): \qquad 3\log\left(\frac{W}{\min\{w(i-),w(i+)\}}\right)+\log\left(\frac{W}{w(i)}\right)+O(1).$$

$$del(i,t): \qquad 3\log\left(\frac{W}{w(i)}\right)+\log\left(\frac{W-w(i)}{w(i-)}\right)+O(1).$$

Proof. Οι πολυπλοκότητες προκύπτουν από το Λήμμα 11.1, σαν επακόλουθο του ότι κάθε πράξη περιέχει και έναν μετασχηματισμό εξάρθρωσης. Ας δούμε πρώτα την *access*. Από το Λήμμα 11.1 έχουμε ότι η επιμερισμένη πολυπλοκότητα είναι το πολύ $3(\log(s(t)/s(x)))+1$. Επιπλέον ισχύουν, $s(t)=W$ και $s(x)\geq w(i)$, λόγω του γεγονότος ότι ο κόμβος x μπορεί να έχει μη κενά υπόδεντρα. Αν πάλι δεν υπάρχει η τιμή i στο t θα είναι $s(x)\geq\min\{w(i-),w(i+)\}$, όπου αυτή τη φορά ο x δεν περιέχει την τιμή i αλλά την $i-$ ή την $i+$, ανάλογα που γίνεται εξάρθρωση. Προκύπτει εύκολα λοιπόν ο χρόνος για *access*, ενώ και για την *Split*, ισχύουν ανάλογα.

Σε ότι αφορά την $Join$, η πολυπλοκότητα προκύπτει από αυτή της $Access$, αλλά προκύπτει επιπλέον αύξηση του δυναμικού μετά την συνένωση. Αυτή η αύξηση είναι $\log\big(s(t_1)+s(t_2)\big)-\log s(t_1) \leq 3\log\left(\frac{W}{s(t_1)}\right)$, όπου $W = s(t_1)+s(t_2)$ οπότε η συνολική επιμερισμένη πολυπλοκότητα της πράξης θα είναι <κόστος εξάρθρωσης> + <μεταβολή δυναμικού κατά την ένωση> + <πραγματικό κόστος Join> $= 3\log\left(\frac{s(t_1)}{w(i)}\right) + 3\log\left(\frac{W}{s(t_1)}\right) + O(1) = 3\log\left(\frac{W}{w(i)}\right) + O(1)$.

Για την πράξη ins θα θεωρήσουμε ότι $W = s(t) + w(i)$, οπότε ο πρώτος όρος προκύπτει από το κόστος της πράξης εξάρθρωσης που θα χρειαστεί και ο δεύτερος όρος λόγω της αύξησης του δυναμικού του δέντρου, μετά την προσθήκη του κόμβου με τιμή i στη ρίζα. Αυτή η αύξηση ισούται με $\log\big(s(t)+w(i)\big) - \log w(i) = \log\frac{W}{w(i)}$. □

Αξίζει να σημειωθεί ότι τα βάρη χρησιμοποιούνται μόνο για τους σκοπούς της ανάλυσης και ότι το αρθωρμένο δέντρο δεν έχει καμία γνώση των βαρών. Συνεπώς, το Θεώρημα 11.1 είναι αληθές για **οποιαδήποτε** ανάθεση θετικών βαρών. Θα δούμε κάποια Θεωρήματα που αποδεικνύουν την καλή απόδοση των αρθρωμένων δέντρων.

Θεώρημα 11.2. *Έστω ακολουθία m προσβάσεων σε αρθρωμένο δέντρο n κόμβων. Το συνολικό κόστος της ακολουθίας είναι $O((m+n)\log n + m)$.*

Θεώρημα 11.3. *Έστω τυχαίο αρθρωμένο δέντρο με n κόμβους, οι κόμβοι του οποίου είναι αριθμένοι με συμμετρική διάταξη. Το κόστος της προσπέλασης όλων των στοιχείων του βάσει της συμμετρικής διάταξης είναι μόλις $O(n)$.*

Έστω τώρα ότι χρησιμοποιούμε το αρθρωμένο δέντρο σαν διπλοουρά. Οι διπλοουρές υποστηρίζουν εισαγωγές και διαγραφές στοιχείων από τα δύο άκρα τους. Οι διπλοουρές μπορούν να προσομοιωθούν από αρθρωμένα δέντρα αντιστοιχίζοντας το $i-$στό απο αριστερά στοιχείο της διπλοουράς, στον $i-$στό βάσει συμμετρικής διάταξης, κόμβο του δέντρου.

Θεώρημα 11.4. *Έστω ένα αρθρωμένο δέντρο με n κόμβους, το οποίο προσομοιώνει μια διπλοουρά. Μια ακολουθία από m πράξεις διπλοουράς στο δέντρο, χρειάζεται χρόνο $O(n+m)$.*

11.5 Βιβλιογραφική Συζήτηση

Στο [158] μπορείτε να βρείτε την πρωτογενή παρουσίαση των τεχνικών του λογιστή και του φυσικού με την μορφή που τις δώσαμε προηγουμένως. Επίσης

κλασικά παραδείγματα εφαρμογής αυτής της ανάλυσης μπορείτε να βρείτε στα [111, 110, 151]. Το κλασικό βιβλίο [27] περιέχει μία εκτενή αναφορά στην επιμερισμένη ανάλυση.

Επιπλέον, στο [119] οι τεχνικές επιμερισμένης ανάλυσης έχουν επεκταθεί ώστε να λαμβάνεται υπόψη και η διαχρονικότητα. Μία δομή δεδομένων είναι διαχρονική [33] όταν επιτρέπει την δεικτοδότηση όχι μόνο των στοιχείων που περιέχει μία συγκεκριμένη χρονική στιγμή αλλά τη δεικτοδότηση και όλων των εκδοχών της με βάση τις πράξεις ενημέρωσης που έχουν γίνει σε αυτή.

Οι δομές δεδομένων με καλή επιμερισμένη πολυπλοκότητα είναι συνήθως εξαιρετικά απλές και κατάλληλες για αποδοτικές υλοποιήσεις [119]. Όμως, υπάρχουν περιπτώσεις όπου η ανυπαρξία εγγύησης για το κόστος κάθε πράξης ξεχωριστά δεν είναι επιθυμητή - για παράδειγμα σε συστήματα πραγματικού χρόνου. Σε αυτές τις περιπτώσεις απαιτούμε κάθε πράξη να έχει εγγυημένα μικρό κόστος οπότε στην προκειμένη περίπτωση το καλύτερο μέτρο είναι η πολυπλοκότητα χειρότερης περίπτωσης για κάθε πράξη. Γεννάται το ερώτημα λοιπόν αν δοθείσης μία δομής δεδομένων με καλό επιμερισμένο κόστος για τις πράξεις που υποστηρίζει μπορούμε να σχεδιάσουμε μία άλλη δομή με αντίστοιχες πολυπλοκότητες χειρότερης περίπτωσης για τις ίδιες πράξεις. Αυτές συνήθως είναι πιο πολύπλοκες αφού απαιτούν μηχανισμούς αυξητικής δρομολόγησης των ακριβών πράξεων. Ένα κλασικό παράδειγμα είναι το δένδρο με δακτυλοδείκτες όπου οι υλοποιήσεις με επιμερισμένη [110] ή μέση πολυπλοκότητα [145] είναι εξαιρετικά πιο απλές σε σχέση με την αντίστοιχη για πολυπλοκότητα χειρότερης περίπτωσης [15] που είναι εξαιρετικά πολύπλοκη και δυσνόητη.

Μερικές φορές όμως είναι αδύνατο να εξισωθούν η επιμερισμένη πολυπλοκότητα με την πολυπλοκότητα χειρότερης περίπτωσης για συγκεκριμένο πρόβλημα. Κλασικό παράδειγμα αποτελεί το πρόβλημα της Ένωσης-Εύρεσης συνόλων (Union-Find). Η βέλτιστη επιμερισμένη πολυπλοκότητα για την πράξη Εύρεσης (Find) είναι $\Theta(\alpha(n,m))$ [156], όπου n είναι ο αριθμός των στοιχείων, m είναι το πλήθος των πράξεων Εύρεσης και $\alpha()$ είναι η αντίστροφη συνάρτηση Ackermann που αυξάνεται με πολύ αργό ρυθμό. Η βέλτιστη πολυπλοκότητα χειρότερης περίπτωσης για την πράξη της Εύρεσης είναι $\Theta(\frac{\log n}{\log \log n})$ [118] και ασυμπτωτικά είναι πολύ μεγαλύτερη από την βέλτιστη επιμερισμένη πολυπλοκότητα.

11.6 Ασκήσεις

1. Στην απόδειξη με την τεχνική του λογιστή έχουμε κάνει τη σιωπηρή υπόθεση ότι η πράξη της επέκτασης από έναν πίνακα T μεγέθους n σε έναν πίνακα T' μεγέθους $\frac{4n}{3}$ απαιτεί n βήματα. Αν απαιτείται η αρχικοποίηση

των άδειων κελιών του πίνακα πόσο πρέπει να γίνει το επιμερισμένο κόστος της εισαγωγής? Αντίστοιχα στη σύντμηση έχουμε κάνει τη σιωπηρή υπόθεση ότι όλα τα στοιχεία είναι συνεχόμενα στον πίνακα T και επομένως το κόστος είναι $\frac{n}{2}$ για την αντιγραφή στον T'. Πόσο πρέπει να είναι το επιμερισμένο κόστος της διαγραφής αν τα στοιχεία είναι διάσπαρτα στον T και απαιτούνται n βήματα για την αντιγραφή των στοιχείων? Αντίστοιχα να αλλάξετε και τη συνάρτηση δυναμικού στην απόδειξη με τη μέθοδο του φυσικού.

2. Να γενικεύσετε την απόδειξη με την τεχνική του φυσικού για δυναμικούς πίνακες ώστε να εφαρμόζεται σε τυχαία α και β. Πόσο είναι το επιμερισμένο κόστος σαν συνάρτηση των α και β? Επιπλέον, αν θεωρήσετε ότι με γ αναπαριστούμε την πληρότητα ενός πίνακα μετά από σύντμηση ή επέκταση, να υπολογίσετε το επιμερισμένο κόστος σαν συνάρτηση των α, β και γ. Προσέξτε ότι $\alpha < \gamma < \beta$ ενώ στην ανάλυση που κάναμε στο 11.3, $\gamma = \frac{3}{4}$.

3. Η γνωστή βιομηχανία συσκευών ΣΥΣΚΟ κατασκευάζει μαραφέτια. Στα κεντρικά γραφεία της η εταιρεία έχει μία οθόνη που απεικονίζει των αριθμό συσκευών που έχουν κατασκευασθεί μέχρι τώρα. Κάθε φορά που ένα μαραφέτι κατασκευάζεται, ένας υπάλληλος ενημερώνει την οθόνη. Το κόστος της ενημέρωσης είναι $c + dm$, όπου c είναι το σταθερό κόστος για την μετακίνηση του υπαλλήλου προς την οθόνη, d είναι το κόστος ανά ψηφίο που αλλάζει στην οθόνη ενώ m είναι ο αριθμός των ψηφίων που αλλάζουν. Για παράδειγμα, όταν η οθόνη μεταβάλλεται από 12999 σε 13000, το κόστος για την εταιρεία είναι $c + 4d$ μιας και αλλάζουν τέσσερα ψηφία συνολικά. Η εταιρεία θα ήθελε να επιμερίσει το κόστος της ενημέρωσης της οθόνης σε κάθε μαραφέτι που κατασκευάζεται, με σταθερό κόστος ανά μαραφέτι. Πιο συγκεκριμένα, θέλουμε να υπολογίσουμε μία ποσότητα e που προσάπτεται σε κάθε μαραφέτι έτσι ώστε το σύνολο των ποσοτήτων αυτών να είναι ίσο ή να ξεπερνά το συνολικό πραγματικό κόστος ενημέρωσης της οθόνης. Η ΣΥΣΚΟ για να κρατήσει χαμηλά την τιμή πώλησης θέλει να μειώσει το e όσο το δυνατόν περισσότερο χωρίς βέβαια να ζημιώνεται. Προφανώς, $e > c + d$ μιας και κάθε φορά που κατασκευάζεται ένα μαραφέτι τουλάχιστον ένα ψηφίο (το μικρότερης σημαντικότητας) πρέπει να αλλάξει.

4. Ένας δυαδικός μετρητής είναι ένας δυαδικός αριθμός που έχει k δυαδικά ψηφία και υποστηρίζει μόνο μία πράξη: την αύξησή του κατά ένα. Αν μετράμε τον αριθμό των δυαδικών ψηφίων που μεταβάλλονται σε κάθε

αύξηση τότε είναι προφανές ότι το κόστος χειρότερης περίπτωσης θα είναι $O(k)$ (στην περίπτωση αύξησης από το $2^k - 1$ στο 2^k).

5. Να αποδείξετε ότι το επιμερισμένο κόστος κάθε αύξησης είναι μόλις $O(1)$.

6. Μπορείτε να σκεφτείτε έναν τρόπο ώστε το κόστος χειρότερης περίπτωσης για τη μοναδιαία αύξηση να είναι $O(1)$?

   Υπόδειξη: Φανταστείτε τον δυαδικό μετρητή υλοποιημένο σε μία λίστα, όπου το πρώτο στοιχείο είναι το ψηφίο μικρότερης σημαντικότητας (LSB) ενώ το τελευταίο στοιχείο είναι το ψηφίο μεγαλύτερης σημαντικότητας (MSB) και προσπαθήστε να προσθέσετε και άλλους δείκτες.

7. Ποιο είναι το επιμερισμένο κόστος στην περίπτωση όπου επιτρέπουμε εκτός από τη μοναδιαία αύξηση και τη μοναδιαία μείωση?

8. Μπορείτε να προτείνετε μία λύση ώστε οι πράξεις της μοναδιαίας αύξησης και μοναδιαίας μείωσης να υποστηρίζονται σε $O(1)$ χρόνο χειρότερης περίπτωσης?

9. Να δείξετε πως μπορούμε να υλοποιήσουμε μία ουρά (υποστηρίζονται οι πράξεις εισαγωγής στο αριστερότερό της σημείο και διαγραφής από το δεξιότερό της σημείο) χρησιμοποιώντας δύο σωρούς έτσι ώστε το επιμερισμένο κόστος των δύο πράξεων να είναι $O(1)$.

10. Θεωρείστε μία απλή ουρά προτεραιότητας με τη μορφή πλήρους δυαδικού δένδρου όπου υποστηρίζονται οι πράξεις Διαγραφή_Ελαχίστου (DeleteMin) και Εισαγωγή (Insert) – στη ρίζα αποθηκεύεται το ελάχιστο στοιχείο ενώ κάθε κόμβος περιέχει στοιχείο που είναι ελάχιστο ανάμεσα σε όλα τα στοιχεία του υποδένδρου του. Γνωρίζουμε ότι το κόστος χειρότερης περίπτωσης και για τις δύο πράξεις είναι $O(\log n)$, όπου n είναι ο αριθμός των στοιχείων στην ουρά προτεραιότητας. Προτείνετε μία κατάλληλη συνάρτηση δυναμικού έτσι ώστε το επιμερισμένο κόστος της πράξης Διαγραφή_Ελαχίστου να είναι $O(1)$ ενώ το επιμερισμένο κόστος της Εισαγωγής να είναι $O(\log n)$.

Βιβλιογραφία

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison Wesley, 1974.

[2] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *Data Structures and Algorithms.* Addison Wesley, 1983.

[3] O. Amble and D.E. Knuth. Ordered hash tables. *The Computer Journal*, 17(2):135–142, 1974.

[4] P. Bachmann. *Analytische Zahlentheorie.* Teubner, 1894.

[5] J.R. Bell. The quadratic quotient method - a hash code eliminating secondary clustering. *Communications of the ACM*, 13(2):108–109, 1970. Also: 26(1):62–63, 1983.

[6] J. Bentley. *Writing Efficient Programs.* Prentice Hall, 1982.

[7] J. Bentley. *Programming Pearls.* Addison-Wesley, 1986.

[8] J. Bentley. *More Programming Pearls - Confessions of a Coder.* Addison-Wesley, 1988.

[9] J. Bentley, S. Haken, and J. Saxe. A general method for solving divide-and-conquer recurrences. *ACM SIGACT News*, 12(3):36–44, 1980.

[10] M. Blum, R.W. Floyd, V. Pratt, R.L. Lewis, and R.E. Tarjan. Time bounds for selection. *journal of Computer and System Sciences*, 7:448–461, 1973.

[11] J. Boothroyd. Algorithm 201 - Shellsort. *Communications of the ACM*, 6(8):445, 1963.

[12] M. Bradley. Analyzing multi-phase searching algorithms. *ACM SIGCSE Bulletin*, 28(3):5–8, 1996.

[13] G. Brassard and P. Bratley. *Fundamentals of Algorithmics.* Prentice Hall, 1996.

[14] B. Brejova. Analyzing variants of Shellsort. *Information Processing Letters*, 79(5):223–227, 2001.

[15] G.S. Brodal, G. Lagogiannis, C. Makris, A.K. Tsakalidis, and K. Tsichlas. Optimal finger search trees in the pointer machine. *Journal of Computer and System Sciences*, 67(2):381–418, 2003.

[16] C. Bron. Algorithm 426 - Merge sort algorithm. *Communications of the ACM*, 15(5):357–358, 1972.

[17] D. Brunskill and J. Turner. *Understanding Algorithms and Data Structures.* McGraw Hill, 1996.

[18] V. Bryant. *Aspects of Combinatorics.* Cambridge University Press, 1993.

[19] W.F. Burton and G.N. Lewis. A robust variation of interpolation search. *Information Processing Letters*, 10(4):198–201, 1980.

[20] R. Chaudhuri. Do arithmetic operations really execute in constant time? *ACM SIGCSE Bulletin*, 35(2):43–44, 2003.

[21] R. Chaudhuri. Teaching bit-level algorithm analysis to the undergraduates in computer science. *ACM SIGCSE Bulletin*, 36(2):62–63, 2004.

[22] R. Chaudhuri and A.C. Dempster. A note on slowing quicksort. *ACM SIGCSE Bulletin*, 25(2):57–58, 1993.

[23] I.P. Chu and R. Johnsonbaugh. The four-peg tower of Hanoi puzzle. *ACM SIGCSE Bulletin*, 23(3):2–4, 1991.

[24] E. Cohen. *Programming in the 1990s - an Introduction to the Calculation of Programs.* Springer-Verlag, 1990.

[25] J. Cohen and M. Ruth. On the implementation of Strassen's fast multiplication method. *Acta Informatica*, 6(4):341–355, 1976.

[26] C.R. Cook and D.J. Kim. Best sorting algorithm for nearly sorted lists. *Communications of the ACM*, 23(11):620–624, 1980.

[27] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, McGraw-Hill, 2nd edition, 2001.

[28] S. Dasgupta, C. Papadimitriou, and U. Vazirani. *Algorithms.* McGraw-Hill, 2006.

[29] F. Delahan, W. Klostermeyer, and G. Tharp. Another way to solve nine-trails. *ACM SIGCSE Bulletin*, 27(4):27–28, 1995.

[30] R. Devillers and G. Louchard. Hashing techniques - a global approach. *BIT*, 19(3):302–311, 1979.

[31] E.E. Doberkat. Asymptotic estimates for the higher moments of the expected behavior of straight insertion sort. *Information Processing Letters*, 14(4):179–182, 1982.

[32] W. Dobosiewicz. An efficient variation of bubble sort. *Information Processing Letters*, 11(1):5–6, 1980.

[33] J.R. Driscoll, N. Sarnak, D. Sleator, and R.E. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38(1):86–124, 1989.

[34] R.J. Embody and H.C. Du. Dynamic hashing schemes. *ACM Computer Surveys*, 20(2):85–113, 1988.

[35] S. Epp. *Discrete Mathematics and Applications*. Wadsworth Publishing, 2nd edition, 1995.

[36] H. Erkio. A heuristic approximation of the worst case of Shellsort. *BIT*, 20(2):130–136, 1980.

[37] T.O. Espelid. Analysis of Shellsort algorithm. *BIT*, 13(4):394–400, 1973.

[38] V. Esteville-Castro and D. Woods. A survey of adaptive sorting algorithms. *ACM Computing Surveys*, 24(4):441–476, 1992.

[39] J. Fenwick, C. Norris, and J. Wilkes. Scientific experimentation via the matching game. *ACM SIGCSE Bulletin Inroads*, 34(1):326–330, 2002.

[40] D.E. Ferguson. Fibonaccian searching. *Communications of the ACM*, 3(12):648, 1960.

[41] F. Ferri and J. Albert. Average-case analysis in an elementary course on algorithms. *ACM SIGCSE Bulletin Inroads*, 30(1):202–206, 1998.

[42] I. Flores and G. Madpis. Average binary search length for dense ordered lists. *Communications of the ACM*, 14(9):602–603, 1971.

[43] Jr. Ford, R. Lestor, and S. Johnson. A tournament problem. *The American Mathematical Monthly*, 66:387–389, 1959.

[44] W.R. Franklin. Padded lists - set operations in expected $O(\log \log n)$ time. *Information Processing Letters*, 9(4):161–166, 1979.

[45] M.R. Garey and D.S. Johnson. *Computers and Intractability - a Guide to the Theory of NP-Completeness*. Freeman, 1978.

[46] T. Gegg-Harrison. Ancient egyptian numbers - a cs-complete example. *ACM SIGCSE Bulletin Inroads*, 31(1):268–272, 2001.

[47] D. Ghoshdastidar and M.K. Roy. A study on the evaluation of Shell's sorting technique. *The Computer Journal*, 18(3):234–235, 1975.

[48] A. Gill. Hierarchical binary search. *Communications of the ACM*, 23(5):294–300, 1980.

[49] D. Ginat. Efficiency of algorithms for programming beginners. *ACM SIGCSE Bulletin Inroads*, 28(1):256–260, 1996.

[50] D. Ginat. Starting top-down, refining bottom-up, sharpening by zoom-in. *ACM SIGCSE Bulletin Inroads*, 33(4):28–31, 2001.

[51] D. Ginat. Algorithmic patterns and the case of the sliding data. *ACM SIGCSE Bulletin Inroads*, 36(2):29–33, 2004.

[52] D. Ginat. Domino arrangments. *ACM SIGCSE Bulletin Inroads*, 39(2):24–25, 2007. Also, 39(4):28-29.

[53] G. Gonnet and R. Baeza-Yates. *Handbook of Algorithms and Data Structures*. Addison-Wesley, 2nd edition, 1991.

[54] G.H. Gonnet and J.I. Munro. A linear probing sorting and its analysis. In *Proceedings 13th ACM Symposium on Theory of Computing (STOC)*, pages 90–95, 1981.

[55] G.H. Gonnet and L.D. Rogers. The interpolation sequential search algorithm. *Information Processing Letters*, 6(4):136–139, 1977.

[56] G.H. Gonnet, L.D. Rogers, and J.A. George. An algorithmic and complexity analysis of interpolation search. *Acta Informatica*, 13(1):39–52, 1980.

[57] S.E. Goodman and S.T. Hedetniemi. *Introduction to the Design and Analysis of Algorithms*. McGraw-Hill, 2nd edition, 1984.

[58] H.S. Govinda Rao. *Graph Theory and Combinatorics*. Galgotia Publications, 2006.

[59] R. Graham, D.E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison-Wesley, 1989.

[60] J.S. Gray. Is eight enough? the eight queens problem re-examined. *ACM SIGCSE Bulletin*, 25(3):39–44, 1993.

[61] D. Gries. *Science of Programming*. Springer-Verlag, 1981.

[62] D. Gries and G. Levin. Computing fibonacci numbers (and similarly defined functions) in log time. *Information Processing Letters*, 11(2):68–69, 1980.

[63] L.J. Guibas and E. Szeremedi. The analysis of double hashing. *journal on Computer Systems and Sciences*, 16(2):226–274, 1978.

[64] B.K. Haddon. Cycle sort - a linear sorting method. *The Computer Journal*, 33(4):365–367, 1990.

[65] P. Heck. Dynamic programming for pennies a day. *ACM SIGCSE Bulletin*, 26(1):213–217, 1994.

[66] T.H. Hibbard. An empirical study of minimal storage sorting. *Communications of the ACM*, 6(5):206–213, 1963.

[67] C.A.R. Hoare. Algorithm 63 - Partition and algorithm 65 - Find. *Communications of the ACM*, 4:321–322, 1961.

[68] C.A.R. Hoare. Quicksort. *The Computer Journal*, 5(1):10–15, 1962.

[69] M. Hofri. *Analysis of Algorithms*. Oxford University Press, 1995.

[70] E. Horowitz, S. Sahni, and S. Rajasekaran. *Computer Algorithms*. Computer Science Press, 1998.

[71] J. Incerpi and R. Sedgewick. Practical variations of Shellsort. *Information Processing Letters*, 26(1):37–43, 1987.

[72] J. Jaja. A perspective on quicksort. *IEEE Computing in Science and Engineering*, 2(1):43–49, 2000.

[73] W. Janko. Variable jump search - the algorithm and its efficiency. *Angewandte Informatik*, 23(1):6–11, 1981.

[74] P.R. Jones. Comment on average binary search length. *Communications of the ACM*, 15(8):774, 1972.

[75] G.D. Knott. Hashing functions. *The Computer Journal*, 18(2):265–278, 1975.

[76] D.E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison-Wesley, 1973.

[77] D.E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, 1975.

[78] D.E. Knuth. Big Omicron and big Omega and big Theta. *ACM SIGACT News*, 8(2):18–24, 1976.

[79] D.E. Knuth. Algorithms. *Scientific American*, 236(4):63–80, April 1977.

[80] D.E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, 1981.

[81] J. Lagarias. The $x+1$ problem and its generalizations. *American Mathematical Monthly*, 92:3–23, 1985.

[82] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys (eds.). *The Traveling Salesman Problem: a Guided Tour*. John Wiley, 1985.

[83] T. Leipala. On a generalization of binary search. *Information Processing Letters*, 8(5):230–233, 1979.

[84] R. Lesuisse. Some lessons drawn from the history of the binary search algorithm. *The Computer Journal*, 26(2):154–163, 1983.

[85] A. Levitin. *Introduction to the Design and Analysis of Algorithms*. Addison-Wesley, 2nd edition, 2006.

[86] G.N. Lewis, N.J. Boynton, and F.W. Burton. Expected complexity of fast search with uniformly distributed data. *Information Processing Letters*, 13(1):4–7, 1981.

[87] H.R. Lewis and C.H. Papadimitriou. The efficiency of algorithms. *Scientific American*, 238(1), 1974.

[88] H.R. Lewis and C.H. Papadimitriou. *Elements of Theory of Computation*. Prentice Hall, 1981.

[89] T.G. Lewis and C.R. Cook. Hashing for static and dynamic internal tables. *IEEE Computer*, 21(10):45–56, 1988.

[90] C.L. Liu. *Introduction to Combinatorial Mathematics*. McGraw-Hill, 1968.

[91] E. Lodi and F. Luccio. Split sequence hash search. *Information Processing Letters*, 20(3):131–136, 1985.

[92] R. Loeser. Some performance tests of quicksort and descendants. *Communications of the ACM*, 17(3):143–152, 1974.

[93] H. Lorin. A guided bibliography to sorting. *IBM Systems Journal*, 10(3):244–254, 1971.

[94] G. Luecker and M. Molodowitch. More analysis on double hashing. In *Proceedings 20th ACM Symposium on Theory of Computing (STOC)*, pages 354–359, 1988.

[95] Y. Manolopoulos. Variations of quicksort combined with insertion sort. *Wirtshafts Informatik*, 34(3):327–333, 1992.

[96] Y. Manolopoulos. Binomial coefficient computation - Recursion or iteration? *ACM SIGCSE Bulletin Inroads*, 34(4):65–67, 2002.

[97] Y. Manolopoulos. On the number of recursive calls of recursive functions. *ACM SIGCSE Bulletin Inroads*, 37(2):61–64, 2005.

[98] Y. Manolopoulos, J.G. Kollias, and F.W. Burton. Batched interpolation search. *The Computer Journal*, 30(6):565–568, 1987.

[99] Y. Manolopoulos, J.G. Kollias, and M. Hatzopoulos. Binary vs. sequential batched search. *The Computer Journal*, 29(4):368–372, 1986.

[100] R.J. Maresh. Sorting out basic sorting algorithms. *ACM SIGCSE Bulletin*, 17(4):54–59, 1985.

[101] W.A. Martin. Sorting. *ACM Computing Surveys*, 3(4):148–174, 1971.

[102] O. Martin-Sanchez and C. Pareja-Flores. A gentle introduction to algorithm complexity for CS1 with nine variations on a theme by Fibonacci. *ACM SIGCSE Bulletin*, 27(2):49–56, 1995.

[103] W.D. Maurer. An improved hash code for scatter storage. *Communications of the ACM*, 11(1):35–38, 1968. Also: 26(1):36–38, 1983.

[104] W.D. Maurer and T.G. Lewis. Hash table methods. *ACM Computing Surveys*, 7(1):5–19, 1975.

[105] R. McCloskey and J. Beidler. An analysis of algorithms laboratory utilizing the maximum segment sum problem. *ACM SIGCSE Bulletin*, 31(4):21–26, 1995.

[106] E. McCreight. Priority search trees. *SIAM Journal on Computing*, 14(2):257–276, 1985.

[107] B.J. McKenzie, R. Harries, and T. Bell. Selecting a hashing algorithm. *Software - Practice and Experience*, 20(2):209–224, 1990.

[108] K. Mehlhorn. *Graph Algorithms and NP-completeness*, volume 2 of *Data Structures and Algorithms*. Springer Verlag, 1984.

[109] K. Mehlhorn. *Multidimensional Searching and Computational Geometry*, volume 3 of *Data Structures and Algorithms*. Springer Verlag, 1984.

[110] K. Mehlhorn. *Sorting and Searching*, volume 1 of *Data Structures and Algorithms*. Springer Verlag, 1984.

[111] K. Mehlhorn and A.K. Tsakalidis. An amortized analysis of insertions into AVL-trees. *SIAM journal of Computing*, 15(1):22–33, 1986.

[112] S.M. Merritt. An inverted taxonomy of sorting algorithms. *Communications of the ACM*, 28(1):96–99, 1985.

[113] S. Minsker. The linear twin towers of Hanoi problem. *ACM SIGCSE Bulletin Inroads*, 39(4):37–40, 2007.

[114] B.M.E. Moret and H.D. Shapiro. *Design and Efficiency*, volume 1 of *Algorithms from P to NP*. Benjamin Cummings, 1991.

[115] D. Motzkin. A stable quicksort. *Software – Practice and Experience*, 11(6):607–611, 1981.

[116] D. Motzkin. Meansort. *Communications of the ACM*, 26(4):250–251, 1983.

[117] D. Motzkin and J. Kapenga. More about meansort. *Communications of the ACM*, 27(7):719–722, 1984.

[118] Blum N. On the single operation worst-case time complexity of the disjoint set-union problem. *SIAM Journal on Computing*, 15:1021–1024, 1986.

[119] C. Okasaki. *Purely Functional Data Structures*. Cambridge University Press, 1998.

[120] K.J. Overholt. Efficiency of the fibonacci search method. *BIT*, 13(1):92–96, 1973.

[121] V. Pan. Strassen's algorithm is not optimal. In *Proceedings 19th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 188–176, 1978.

[122] C.N. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1995.

[123] C.N. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, 1982.

[124] E. Peltola and H. Erkio. Insertion merge sorting. *Information Processing Letters*, 7(2):92–99, 1978.

[125] Y. Perl, A. Itai, and H. Avni. Interpolation search - a $\log\log n$ search. *Communications of the ACM*, 21(7):550–553, 1978.

[126] Y. Perl and E.M. Reingold. Understanding the complexity of interpolation search. *Information Processing Letters*, 6(6):219–221, 1977.

[127] W.W. Peterson. Addressing for random access storage. *IBM journal of Research and Development*, 1:130–146, 1957.

[128] C.G. Plaxton and T. Suel. Improved lower bounds for Shellsort. In *Proceedings 33rd Annual IEEE Symposium on the Foundations of Computer Science (FOCS)*, pages 226–235, 1992.

[129] C.E. Radke. The use of quadratic residue research. *Communications of the ACM*, 13(2):103–105, 1970.

[130] E.M. Reingold, J. Nievergelt, and N. Deo. *Combinatorial Algorithms*. Prentice Hall, 1977.

[131] J. Robertson. How many recursive calls does a recursive function make. *ACM SIGCSE Bulletin Inroads*, 31(2):60–61, 1999.

[132] J. Rohrich. A hybrid of quicksort with $O(n \log n)$ worst case complexity. *Information Processing Letters*, 14(3):119–123, 1982.

[133] T. Rolf. Binomial coefficient recursion - the good, the bad and the ugly. *ACM SIGCSE Bulletin Inroads*, 33(2):35–36, 2001.

[134] C. Roussos. Teaching growth of functions using equivalence classes - an alternative to big O notation. *ACM SIGCSE Bulletin Inroads*, 36(1):170–174, 2004.

[135] B. Salzberg. *File Structures: an Analytic Approach*. Prentice Hall, 1988.

[136] N. Santoro and J.B. Sidney. Interpolation binary search. *Information Processing Letters*, 20(4):179–181, 1985.

[137] R.E. Schönhage. Storage modification machines. *SIAM journal on Computing*, 9(3):490–508, 1980.

[138] R.S. Scowen. Algorithm 271 - Quicksort. *Communications of the ACM*, 8(11):669–670, 1965. Also: 9(5):354, 1966.

[139] R. Sedgewick. The analysis of quicksort programs. *Acta Informatica*, 7:327–355, 1977.

[140] R. Sedgewick. Quicksort with equal keys. *SIAM journal on Computing*, 6(2):240–267, 1977.

[141] R. Sedgewick. Implementing quicksort programs. *Communications of the ACM*, 21(10):847–857, 1978.

[142] R. Sedgewick. A new upper bound for Shellsort. *journal of Algorithms*, 7(2):159–173, 1986.

[143] R. Sedgewick. *Algorithms*. Addison-Wesley, 2nd edition, 1988.

[144] R. Sedgewick and P. Flajolet. *An Introduction to the Analysis of Algorithms*. Addison-Wesley, 1996.

[145] C.R. Seidel, R.and Aragon. Randomized search trees. *Algorithmica*, 16(4-5):464–497, 1996.

[146] D. Severance and R. Duhne. A practitioner's guide to addressing algorithms. *Communications of the ACM*, 19(6):314–326, 1976.

[147] C. Shaffer. *Data Structures and Algorithm Analysis*. Prentice Hall, 1987.

[148] D.L. Shell. A highspeed sorting procedure. *Communications of the ACM*, 2(7):30–32, 1959.

[149] B. Shneiderman. Polynomial search. *Software - Practice and Experience*, 3(1):5–8, 1973.

[150] B. Shneiderman. Jump searching - a fast sequential search technique. *Communications of the ACM*, 21(10):831–834, 1978.

[151] D. Sleator and R. Tarjan. Self adjusting binary search trees. *journal of the ACM*, 32(3):652–686, 1985.

[152] R. Sosic and J. Gu. A polynomial time algorithm for the n-queens problem. *ACM SIGART Bulletin*, 1(3):7–11, 1990.

[153] D.R. Stinson. *Cryptography: Theory and Practice*. Cambridge University Press, 2005.

[154] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.

[155] R. Tarjan. Algorithm design. *Communications of the ACM*, 30(3):205–212, 1987.

[156] R.E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, 1975.

[157] R.E. Tarjan. A class of algorithms which require non-linear time to maintain disjoint sets. *Journal of Computer and System Sciences*, 18(2):110–127, 1979.

[158] R.E. Tarjan. Amortized computational complexity. *SIAM journal on Algebraic and Discrete Methods*, 6(2):306–318, 1985.

[159] J. Tillison and C.K. Shene. On generating worst-cases for the insertion sort. *ACM SIGCSE Bulletin*, 27(2):57–58, 1995.

[160] M. Van Der Nat. A fast sorting algorithm - a hybrid of distributive and merge sorting. *Information Processing Letters*, 10(3):163–167, 1980.

[161] M.H. Van Emden. Algorithm 402 - Increasing the efficiency of quicksort. *Communications of the ACM*, 13(9):563–566, 1970.

[162] M.H. Van Emden. Algorithm 402 - Qsort. *Communications of the ACM*, 13(11):693–694, 1970.

[163] J. Vuillemin. A unifying look at data structures. *Communications of the ACM*, 23(4):229–239, 1980.

[164] R.L. Wainwright. A class of sorting algorithms based on quicksort. *Communications of the ACM*, 28(4):396–402, 1985. Also: 29(4):331–335, 1986.

[165] A. Weiss. Empirical results on the running time of Shellsort. *The Computer Journal*, 34(1):88–91, 1991.

[166] A. Weiss and R. Sedgewick. Bad cases for shaker-sort. *Information Processing Letters*, 28(3):13–16, 1988.

[167] A. Weiss and R. Sedgewick. More on Shellsort increment sequences. *Information Processing Letters*, 34(5):267–270, 1990.

[168] A. Weiss and R. Sedgewick. Tight lower bounds for Shellsort. *journal of Algorithms*, 11(2):242–251, 1990.

[169] M.A. Weiss. *Data Structures and Algorithms Analysis.* Benjamin Cummings, 1995.

[170] H. Wilf. *Algorithms and Complexity.* Prentice Hall, 1986.

[171] H. Wilf. *Generatingfunctionology.* Academic Press, 1994.

[172] D.E. Willard. Searching unindexed and nonuniformly generated files in $\log \log n$ time. *SIAM journal of Computing*, 14(4):1014–1029, 1985.

[173] N. Wirth. *Algorithms + Data Structures = Programs.* Prentice Hall, 1976.

[174] N. Wirth. *Algorithms and Data Structures.* Prentice Hall, 1986.

[175] A.C. Yao. An analysis of (h,k,1)-Shellsort. *journal of Algorithms*, 1(1):14–50, 1980.