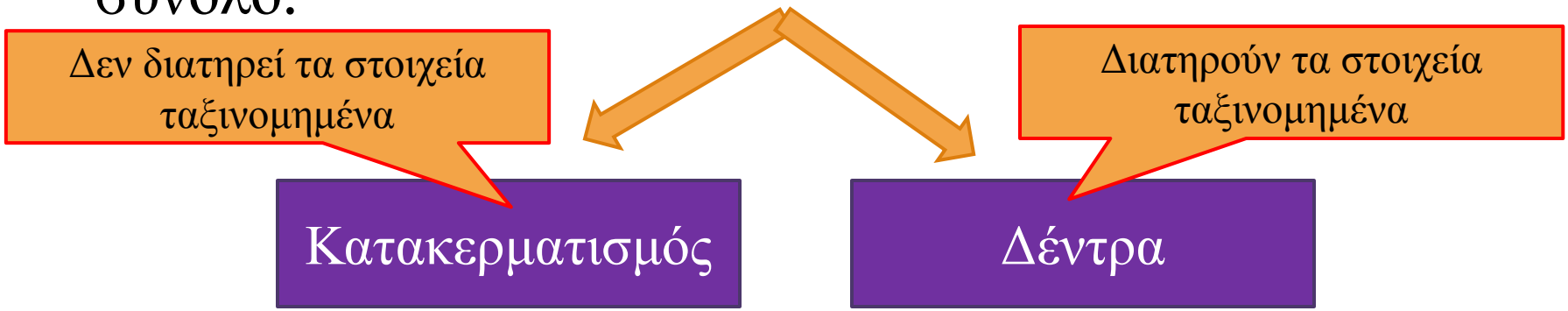

Advanced Data Indexing
(Προηγμένη ευρετηρίαση δεδομένων)
Κατακερματισμός – Hashing (1ο Μέρος)

Η Αναζήτηση απαιτεί Δομές – Λύσεις

- Για να υποστηριχθεί η αποδοτική αναζήτηση απαιτείται η χρήση κατάλληλων δομών-ευρετηρίων (**indexing**).
- Οι δομές αυτές πρέπει να είναι κατάλληλες ώστε οι εξής 3 πράξεις να υποστηρίζονται αποδοτικά:

1. **Εύρεση (search)** ενός στοιχείου στο σύνολο.
2. **Ένθεση (insert)** ενός νέου στοιχείου στο σύνολο.
3. **Διαγραφή (delete)** ενός υπάρχοντος στοιχείου από το σύνολο.



Κατακερματισμός - Γιατί;

- Εφόσον έχουμε με τα δέντρα όλα τα πλεονεκτήματα (βέλτιστα κόστη I/O, διατήρηση ταξινομημένων στοιχείων => άμεση απάντηση σε ερωτήματα περιοχής), γιατί και πότε χρειάζεται ο κατακερματισμός;
- Όταν δεν μας ενδιαφέρει να κρατάμε τα στοιχεία ταξινομημένα, αλλά να τα «**σκορπίσουμε**» με τέτοιο τρόπο **ώστε να πετύχουμε απόδοση** για τις βασικές πράξεις (search, insert, delete) **καλύτερη από αυτή των δέντρων**.
- Πιο συγκεκριμένα:

Κατακερματισμός - Στόχοι

- Να διατηρήσουμε τον **χώρο γραμμικό**: ***O(N)*** στην κεντρική μνήμη ή ***O(N/B)*** στην δευτερεύουσα.
- Να πετύχουμε I/O **κόστος ενημερώσεων** (insert, delete) και **αναζητήσεων** (search) μικρότερο από O(log$N$) ή O($\log_B N$) αντίστοιχα. Ακόμα και ***O(1)***.
- Το I/O **κόστος κατασκευής της δομής** (index) να είναι μικρότερο από το κόστος κατασκευής bulkload των δέντρων, δηλαδή καλύτερο από: $\Theta(\frac{N}{B}\log_{M/B}\frac{N}{B})$. Ακόμα και **γραμμικό**.
- Φυσικά θα χάσουμε τη δυνατότητα άμεσης απάντησης ερωτημάτων περιοχής, αφού τα στοιχεία δεν θα διατηρούνται ταξινομημένα, αλλά και πάλι μπορούν να απαντηθούν με αναζήτηση των ***Z*** στοιχείων τους ένα-ένα με I/O κόστος ***O(Z)*** και όχι O(Z/B).

Κατακερματισμός - Βασικά

- Υπόθεση Ακεραίου Σύμπαντος (Integer Universe):
  - **Όλα τα στοιχεία είναι ακέραιοι** που ανήκουν μέσα σε ένα σύνολο $U = \{0, 1, \ldots, u - 1\}$ που καλείται *σύμπαν*.
    - Δεν είναι περιοριστική η υπόθεση αυτή καθώς κάθε είδος δεδομένων (κείμενο, εικόνα, κλπ.) μετασχηματίζεται σε ένα πλήθος bit στην μνήμη του υπολογιστή, άρα μετατρέψιμο σε ακέραιο του δεκαδικού συστήματος (π.χ. 'Α' → 1000001 → 65).
    - Προσοχή όμως: σύνθετα δεδομένα μπορεί να δώσουν πολύ μεγάλους ακέραιους, οπότε το u (max) να είναι ακέραιος που δεν χωράει σε μία λέξη μνήμης (single word).
  - **Όλα τα στοιχεία είναι διαφορετικά** μεταξύ τους.
    - Δεν υποστηρίζονται στην δομή καταγραφές στοιχείων που επαναλαμβάνονται (ενώ υποστηρίζονται στα δέντρα).

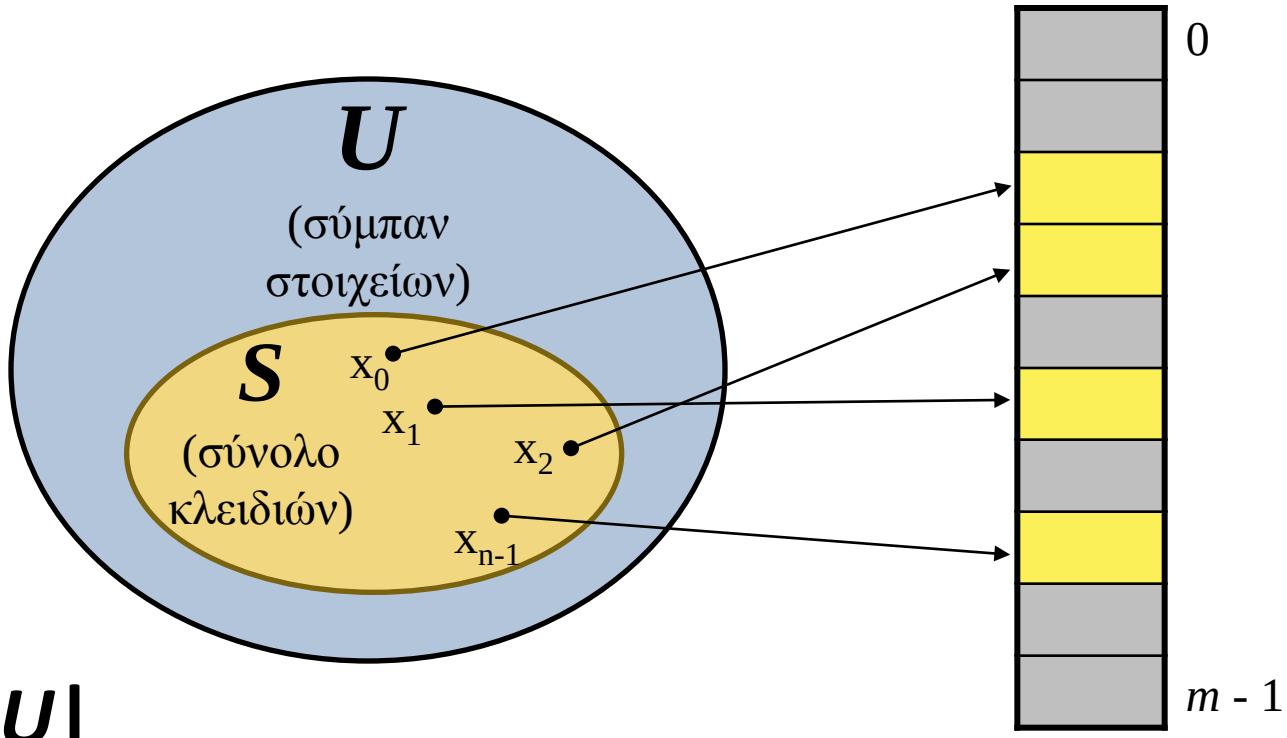
Κατακερματισμός - Βασικά

- Υπόθεση Τυχαίας Επιλογής (Random Probing):
  - Τα στοιχεία έρχονται με τη μορφή μίας ακολουθίας $x_1, x_2, \ldots, x_n$ και **είναι άγνωστο που ανήκουν** (black box).
  - Κάθε στοιχείο τοποθετείται στη δομή hash **ανεξάρτητα**.
  - Τα στοιχεία **κατανέμονται ομοιόμορφα** μέσα σε αυτήν.
  - Όλα τα στοιχεία είναι διαφορετικά μεταξύ τους.

Κατακερματισμός - Βασικά

- Τα στοιχεία είναι συνήθως τα κλειδιά των αντικειμένων.
- θεωρούμε ότι σχηματίζουν το σύνολο ***S*** και έχουν πλήθος ***n*** (***n*=|*S*|**).
- Η δομή του κατακερματισμού είναι μία απλή δομή πίνακα ***A*** (hash table) με μέγεθος ***m*=|*A*|**.
- Κάθε κλειδί του ***S*** αντιστοιχεί σε μία και μόνο μία θέση του πίνακα ***A*** (κάδος).
- Ανάλογα με τις τιμές των ***n, m*** και τον τρόπο αντιστοιχίας προκύπτει ένα συγκεκριμένο πλήθος κενών θέσεων στον πίνακα.

Κατακερματισμός (integer universe assumption)

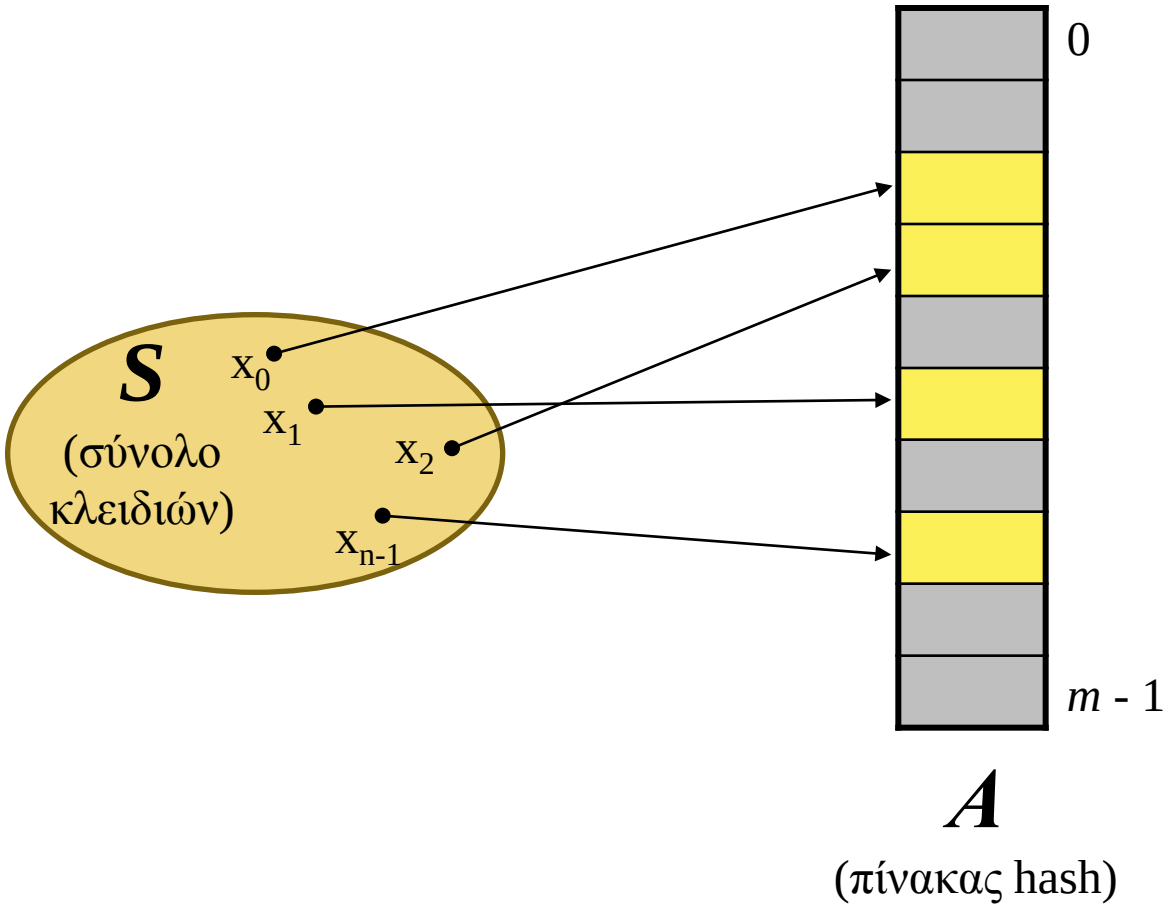


$u=|U|$

$n=|S|\leq u$

$m=|A|$

Κατακερματισμός (random probing assumption)



$n=|S|$

$m=|A|$

Μέθοδοι Κατακερματισμού

integer universe

- Direct Addressing
- *Hashing with chaining*
- Hashing by Division
- Hashing by Multiplication
- Universal Hashing
- Perfect Hashing (static, dynamic)

random probing

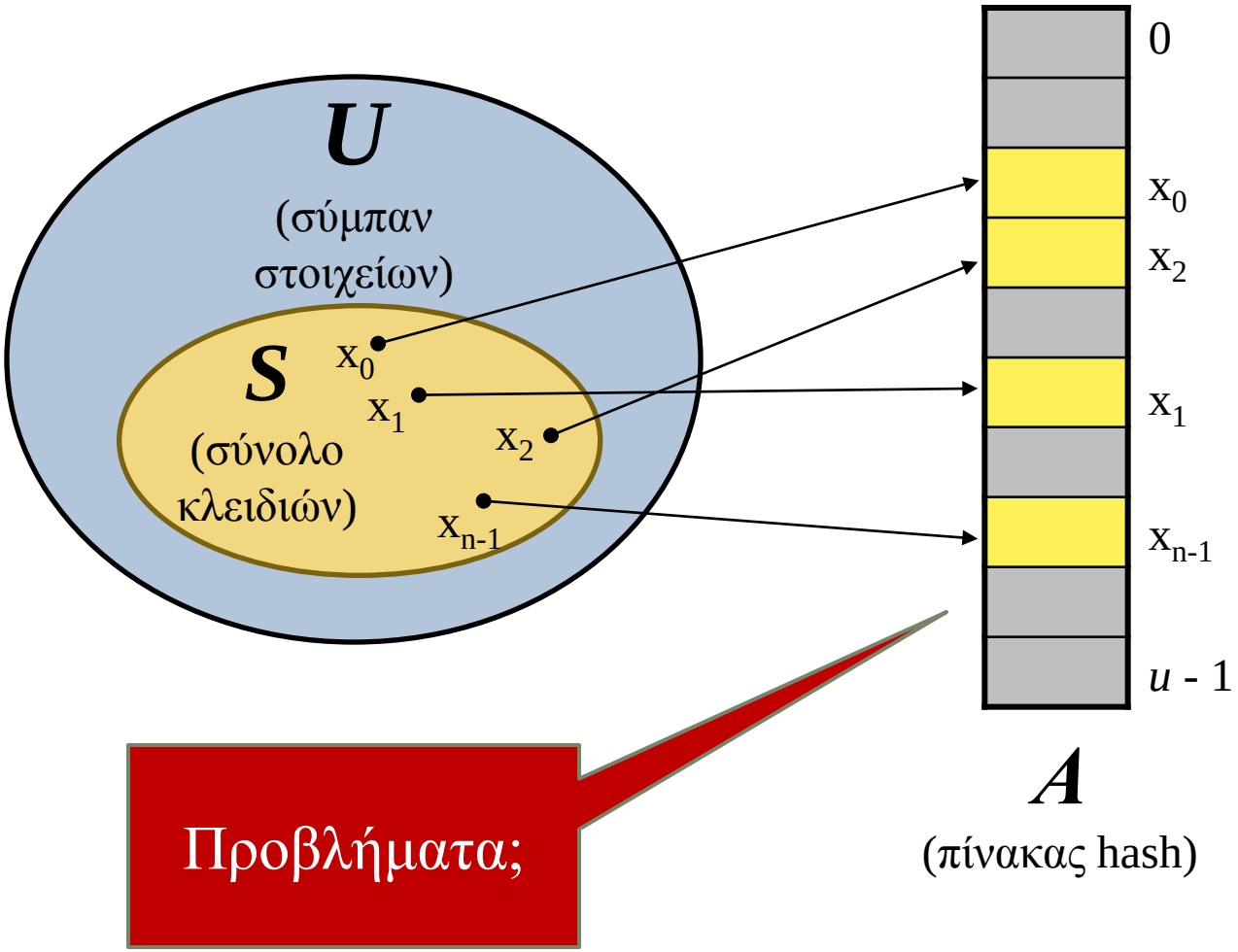
- *Hashing with chaining*
- Hashing with open addressing
- Linear probing
- Quadratic probing
- Double Hashing
- Brent's Method
- Multiple-Choice Hashing
- Asymmetric Hashing
- LCFS Hashing
- Robin-Hood Hashing
- Bloom Filters
- Cuckoo Hashing

Άμεση Διευθυνσιοδότηση
(Direct Addressing)

Άμεση Διευθυνσιοδότηση (Direct Addressing)

- Τα n στοιχεία (κλειδιά) του S είναι ακέραιοι διαφορετικοί μεταξύ τους που ανήκουν στο σύμπαν $U = \{0, 1, \ldots, u-1\}$, με $u=|U|$, $S \subseteq U$ $n=|S| \leq u$.
- Ο πίνακας hash A έχει το ίδιο πλήθος θέσεων (κάδων) με το σύμπαν, δηλαδή $|A|=|U|$ ή $m=u$.
- Κάθε στοιχείο x του S αποθηκεύεται απευθείας στην θέση $A[x]$, δηλαδή ο δείκτης τοποθέτησης στον πίνακα hash ταυτίζεται με το ίδιο το στοιχείο (κλειδί).

Άμεση Διευθυνσιοδότηση (Direct Addressing)



Προβλήματα - Αντιμετώπιση

- Το σύμπαν $\boldsymbol{U}$ μπορεί να είναι πολύ μεγαλύτερο του $\boldsymbol{S}$, οπότε υπάρχει μεγάλη σπατάλη χώρου $\mathbf{O}(\boldsymbol{U})$.
- Μπορεί να μην χωράει ο πίνακας hash στην μνήμη.
- Το μόνο πλεονέκτημα της άμεσης διευθυνσιοδότησης είναι ότι το κόστος για search, insert, delete είναι πάντοτε $\mathbf{O(1)}$.
- Όταν το $\boldsymbol{S}$ είναι μικρότερο του $\boldsymbol{U}$, ο πίνακας hash πρέπει να περιοριστεί σε μέγεθος $\boldsymbol{m}=\mathbf{O}(\boldsymbol{S})$ (ακόμα και $m=|S|$) και έτσι να απαιτεί λιγότερο χώρο από $\mathbf{O}(\boldsymbol{U})$.
- Θέλουμε όμως να κρατήσουμε το κόστος αναζήτησης σε $\mathbf{O(1)}$ τουλάχιστον για τη μέση περίπτωση (ή και για τη χειρότερη σε κάποιες περιπτώσεις).
- Πώς γίνεται αυτό;

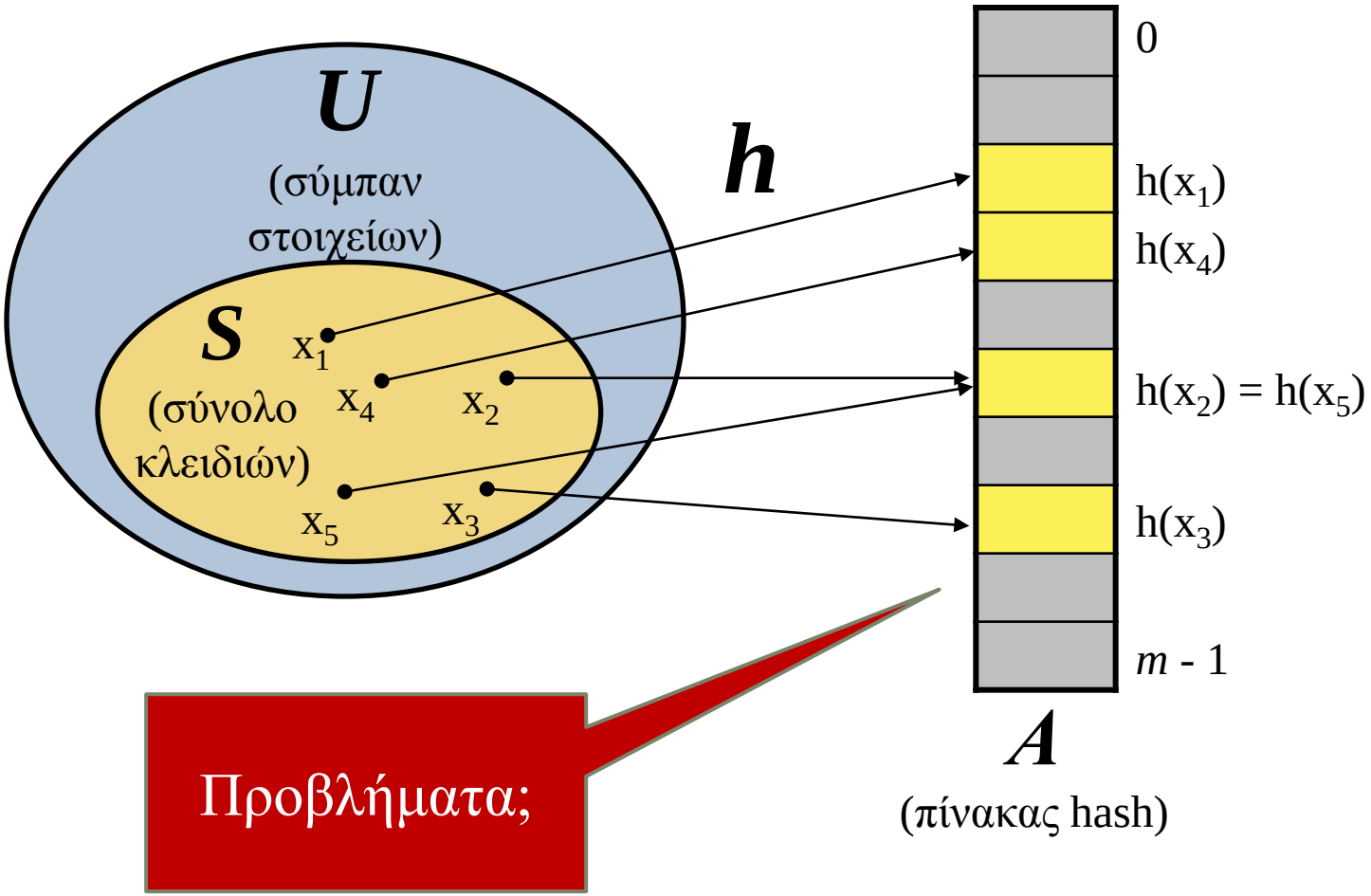
Συνάρτηση Κατακερματισμού

- Χρήση μίας κατάλληλης **συνάρτησης κατακερματισμού** h (hash function) που θα αντιστοιχίζει κάθε στοιχείο του U σε μία συγκεκριμένη θέση του πίνακα hash:

  $$h : U \rightarrow \{0, 1, \ldots, m-1\}$$

- Η συνάρτηση θα πρέπει να αντιστοιχίζει σε μία συγκεκριμένη θέση $i \in \{0, 1, \ldots, m-1\}$ όσο το δυνατόν λιγότερα x του U (δηλαδή έτσι ώστε $h(x)=i$). Ιδεατά θα θέλαμε να αντιστοιχίζει μόνο ένα (δηλαδή να είναι μία συνάρτηση 1-1).
- Τότε θα γίνεται:
  - Χρήση της h για υπολογισμό της θέσης από το στοιχείο x.
  - Αποθήκευση του στοιχείου x στη θέση $h(x)$.

Συνάρτηση Κατακερματισμού



Προβλήματα - Αντιμετώπιση

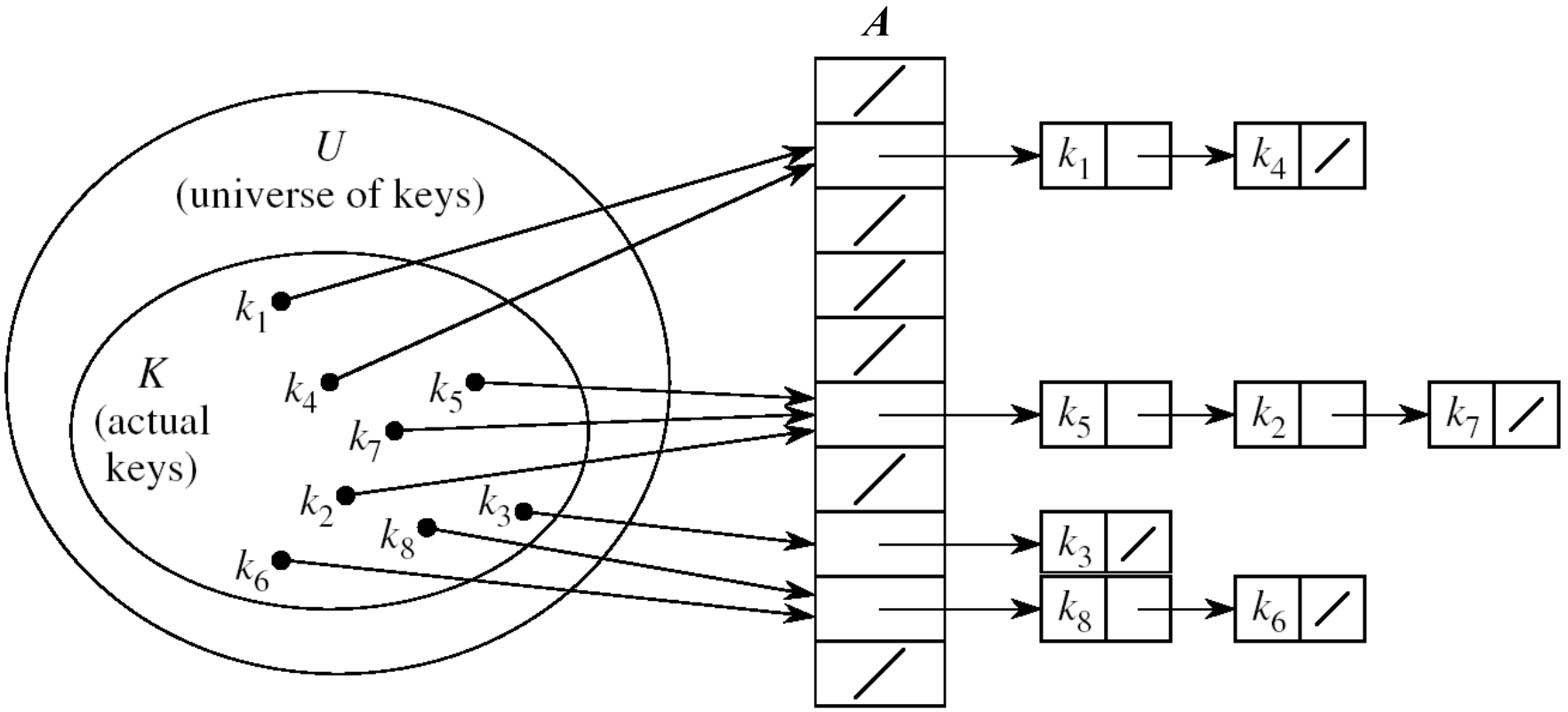
- Το μεγαλύτερο πρόβλημα είναι οι **συγκρούσεις** των στοιχείων σε ίδιες θέσεις.
- Η **αποφυγή των συγκρούσεων** γίνεται με κατάλληλες συναρτήσεις hash. Ωστόσο δεν μπορούμε να τις αποφύγουμε πάντα (π.χ. όταν $m<n$ είναι σίγουρο ότι θα συμβούν).
- Ανάλογα με το χειρισμό των συγκρούσεων έχουμε και διαφορετικές μεθόδους κατακερματισμού, π.χ. με αλυσίδες (chaining), με ανοικτή διευθυνσιοδότηση (open addressing), linear probing, cuckoo hashing, universal hashing, perfect hashing, κλπ.

Κατακερματισμός με Αλυσίδες
(Hashing with Chaining)

Κατακερματισμός με Αλυσίδες (Hashing with Chaining)

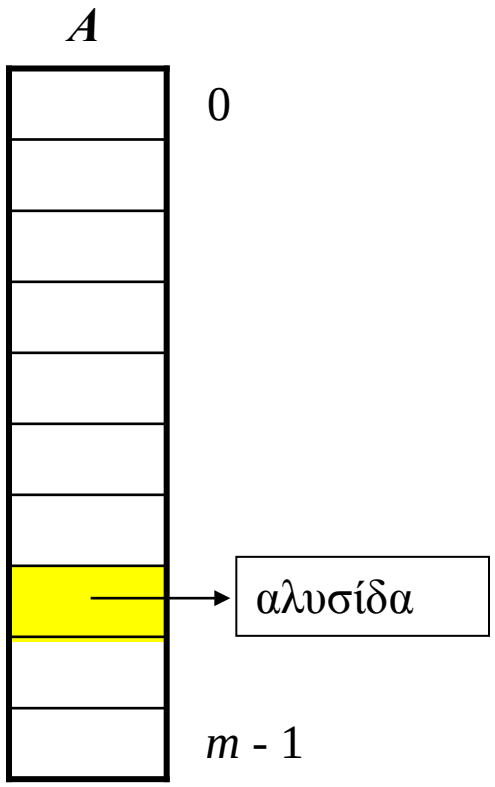
- Η μέθοδος κατακερματισμού με αλυσίδες (hashing with chaining) έχει την εξής απλή ιδέα:
  - **Σε κάθε θέση** (κάδο) του πίνακα hash ***A*** **δημιουργείται** και **μία γραμμική συνδεδεμένη λίστα** (linked list). Ο πίνακας ***A*** δηλαδή έχει ως στοιχεία τις κεφαλές (δείκτες heads) των λιστών.
  - Κάθε φορά που **ένα στοιχείο *x* εισάγεται** στην δομή, βρίσκεται η αντίστοιχη θέση του ***h*(*x*)** μέσω της συνάρτησης κατακερματισμού ***h***, και εισάγεται **στην αντίστοιχη λίστα *A*[*h*(*x*)]**.
  - Συνεπώς στοιχεία που συγκρούονται τοποθετούνται στην ίδια λίστα.
  - Η αναζήτηση ενός στοιχείου θα γίνεται βρίσκοντας την αντίστοιχη λίστα που ανήκει και **διασχίζοντάς την γραμμικά**.

Κατακερματισμός με Αλυσίδες (Hashing with Chaining)



Κατακερματισμός με Αλυσίδες: Απόδοση

- Η εισαγωγή μπορεί να γίνει πάντοτε σε O(1):
  - Εντοπίζεται η θέση του στοιχείου μέσω της συνάρτησης hash, και εισάγεται απευθείας στην αρχή της αντίστοιχης λίστας (γίνεται η νέα κεφαλή).
- Η διαγραφή προϋποθέτει την εύρεση οπότε έχει το ίδιο κόστος με αυτήν. Τι κόστος όμως απαιτείται για την εύρεση ενός στοιχείου;
- **<u>Χειρότερη περίπτωση</u>:**
  - Τα ***n*** στοιχεία πέφτουν όλα στον ίδιο κάδο. Τότε:
  - Κόστος χειρότερης περίπτωσης: **Θ(*n*)**, συν το κόστος υπολογισμών της συνάρτησης κατακερματισμού.



Κατακερματισμός με Αλυσίδες: Απόδοση

- **Μέση Περίπτωση:**
  - Εξαρτάται από πόσο καλά κατανείμει η συνάρτηση ***h*** τα ***n*** στοιχεία στους ***m*** κάδους.
- **Υπόθεση: έστω ότι η *h* είναι κατάλληλη και τα κατανείμει ομοιόμορφα (uniformly distributed).**
  - Τότε κάθε στοιχείο έχει την ίδια πιθανότητα να πέσει σε οποιονδήποτε από τους ***m*** κάδους **(η πιθανότητα σύγκρουσης $\Pr(h(x)=h(y))$ είναι $1/m$).**
- Αν το μήκος της κάθε λίστας (αλυσίδας) είναι:

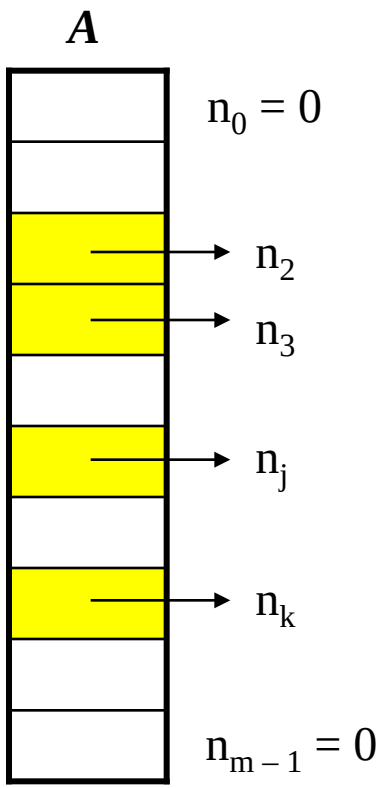
$$|A[j]| = n_j, \qquad j = 0, 1, \ldots, m-1$$

- Τότε το πλήθος των κλειδιών στον πίνακα είναι:

$$n = n_0 + n_1 + \cdots + n_{m-1}$$

- Και το μέσο μήκος n_j θα είναι:

$$E[n_j] = n/m = \alpha \text{ (occupancy)}$$



Κατακερματισμός με Αλυσίδες: Απόδοση

- Η αναζήτηση ενός στοιχείου x:
  - **όταν βρίσκεται στην δομή** απαιτεί κόστος **O(1)** για τον υπολογισμό της θέσης $h(x)$ από τη συνάρτηση hash, συν το μέσο κόστος σάρωσης της αλυσίδας που αντιστοιχεί, το οποίο είναι ανάλογο με το μέσο μήκος της α, δηλαδή συνολικά: $\Theta(1+(n-1)/m) = \Theta(1+\alpha)$.
  - **Όταν δεν βρίσκεται στην δομή** απαιτεί κόστος **O(1)** για τον υπολογισμό της θέσης $h(x)$ από τη συνάρτηση hash, συν το κόστος σάρωσης ολόκληρης της αλυσίδας που αντιστοιχεί (μέχρι το τέλος της) το οποίο είναι ανάλογο με το μέσο μήκος της α, δηλαδή συνολικά: $\Theta(1+n/m) = \Theta(1+\alpha)$.
- Κόστος αναζήτησης/διαγραφής μέσης περίπτωσης: $\Theta(1+\alpha)$
- Αν ο λόγος n/m, δηλαδή το α, είναι μία μικρή σταθερά τότε μπορούμε να πούμε με ασφάλεια ότι το κόστος και για τις πράξεις search, delete είναι **O(1)**.
- Αν όμως το πλήθος των στοιχείων n είναι πολύ μεγαλύτερο του διαθέσιμου χώρου των m αλυσίδων, τότε το κόστος είναι σημαντικό.

Συναρτήσεις Κατακερματισμού

- Είναι πολύ σημαντική για την απόδοση η επιλογή της κατάλληλης συνάρτησης ***h***.
- **Πότε μία συνάρτηση θεωρείται καλή;**

  (1) **Όταν είναι εύκολα υπολογίσιμη**.

  (2) **Όταν προσεγγίζει σε λειτουργία μία τυχαία συνάρτηση**: για κάθε στοιχείο εισόδου δίνει θέση εξόδου με την ίδια πιθανότητα (κάνει δηλαδή απλό ομοιόμορφο κατακερματισμό).
- Στην πράξη είναι δύσκολο να ικανοποιηθεί το δεύτερο κριτήριο καθώς δεν ξέρουμε συνήθως την **κατανομή** των στοιχείων εισόδου.

Χαρακτηριστικά Καλών Συναρτήσεων

- **Ελαχιστοποίηση πιθανότητας να πέσουν δύο κοντινά στοιχεία στον ίδιο κάδο.**
  - π.χ. οι λέξεις `rt` και `rts` θα πρέπει να απεικονίζονται σε διαφορετικούς κάδους.
- **Οι τιμές της συνάρτησης κατακερματισμού *h* πρέπει να είναι ανεξάρτητες των μοτίβων** που μπορεί να υπάρχουν στα στοιχεία του ***S***.

Κοινές Συναρτήσεις Κατακερματισμού

- Ορισμένες καλές συναρτήσεις στην πράξη όπως προτείνονται από τον **D.E. Knuth** (*The Art of Computer Programming*) είναι:
- **Μέθοδος διαίρεσης**: $h(x) = x \bmod m$
- Για **αλφαριθμητικά** της μορφής $x = s_0 s_1 \ldots s_{k-1}$:

$$h(x) = \left(\left(\sum_{i=0}^{k-1} B^i s_i \right) \bmod 2^w \right) \bmod m$$

π.χ. B=131 και w = μήκος λέξης (bits) (w = 32 ή w = 64).

Μέθοδος Διαίρεσης
(Hashing by Division)

Μέθοδος Διαίρεσης (Hashing by Division)

- Είναι **κατακερματισμός με αλυσίδες** όπου γίνεται χρήση της συνάρτησης:

$$h(x) = x \bmod m$$

- Το στοιχείο ***x*** τοποθετείται σε έναν από τους ***m*** κάδους, ανάλογα με το υπόλοιπο της διαίρεσής του με το ***m***.
- **Πλεονέκτημα**: Γρήγορη (μόνο μία πράξη).
- **Μειονέκτημα**: Μερικές τιμές του ***m*** προκαλούν πολλές συγκρούσεις:
  - Δυνάμεις του 2
  - Σύνθετοι αριθμοί

Παράδειγμα

- Αν $m = 2^p$, τότε το $h(x)$ είναι **τα p λιγότερο σημαντικά bits του x**:
  - $p = 1 \Rightarrow m = 2$

    $\Rightarrow h(x) = \{0,1\}$, 1ο LSB του x
  - $p = 2 \Rightarrow m = 4$

    $\Rightarrow h(x) = \{0,1,2,3\}$, 2 λιγότερα σημαντικά ψηφία του x
- **Συστήνεται η επιλογή του m ως πρώτου αριθμού, όχι κοντά σε δύναμη του 2.**
  - Στήλη 2: $m = 87$
  - Στήλη 3: $m = 100$

x	x mod 87	x mod 100
16838	47	38
5758	16	58
10113	21	13
17515	28	15
31051	79	51
5627	59	27
23011	43	11
7420	25	20
16212	30	12
4086	84	86
2749	52	49
12767	65	67
9084	36	84
12060	54	60
32225	35	25
17543	56	43
25089	33	89
21183	42	83
25137	81	37
25566	75	66
26966	83	66
4977	18	77
20495	50	95
10311	45	11
11367	57	67

Μέθοδος Πολλαπλασιασμού
(Hashing by Multiplication)

Η Μέθοδος Πολλαπλασιασμού

- Είναι όμοια με τη **μέθοδο της διαίρεσης** (κατακερματισμός με αλυσίδες) όπου όμως γίνεται χρήση της συνάρτησης:

$$h(x) = \lfloor m \cdot x \cdot b \rfloor \bmod m$$

- Χρησιμοποιούμε μία πραγματική σταθερά $b \neq 0$ η οποία πολλαπλασιάζεται με τα x και m.
- Κάποιες τιμές για την b είναι πιο ευνοϊκές. Ο Knuth προτείνει για b το χρυσό λόγο: $\phi = \frac{\sqrt{5}-1}{2} = 0.6180339887\ldots$
- **Πλεονέκτημα:** Η τιμή του m δεν είναι πια κρίσιμη, συνεπώς μπορούμε να επιλέγουμε και δυνάμεις του 2 ($m=2^p$).
- **Η απόδοση και των δύο μεθόδων (διαίρεσης-πολλαπλασιασμού) είναι ίδια με του κατακερματισμού με αλυσίδες.**

Παράδειγμα

- Εφόσον η τιμή του $\boldsymbol{m}$ μπορεί να είναι και δύναμη του 2, επιλέγουμε: $\boldsymbol{m = 128 = 2^7}$.
- Για σταθερά $\boldsymbol{b}$ επιλέγουμε τον χρυσό λόγο $\boldsymbol{\varphi}$.

$$h(x) = \lfloor 128 \cdot x \cdot \phi \rfloor \bmod 128$$

- Παρατηρούμε ότι και πάλι δεν έχουμε συγκρούσεις.

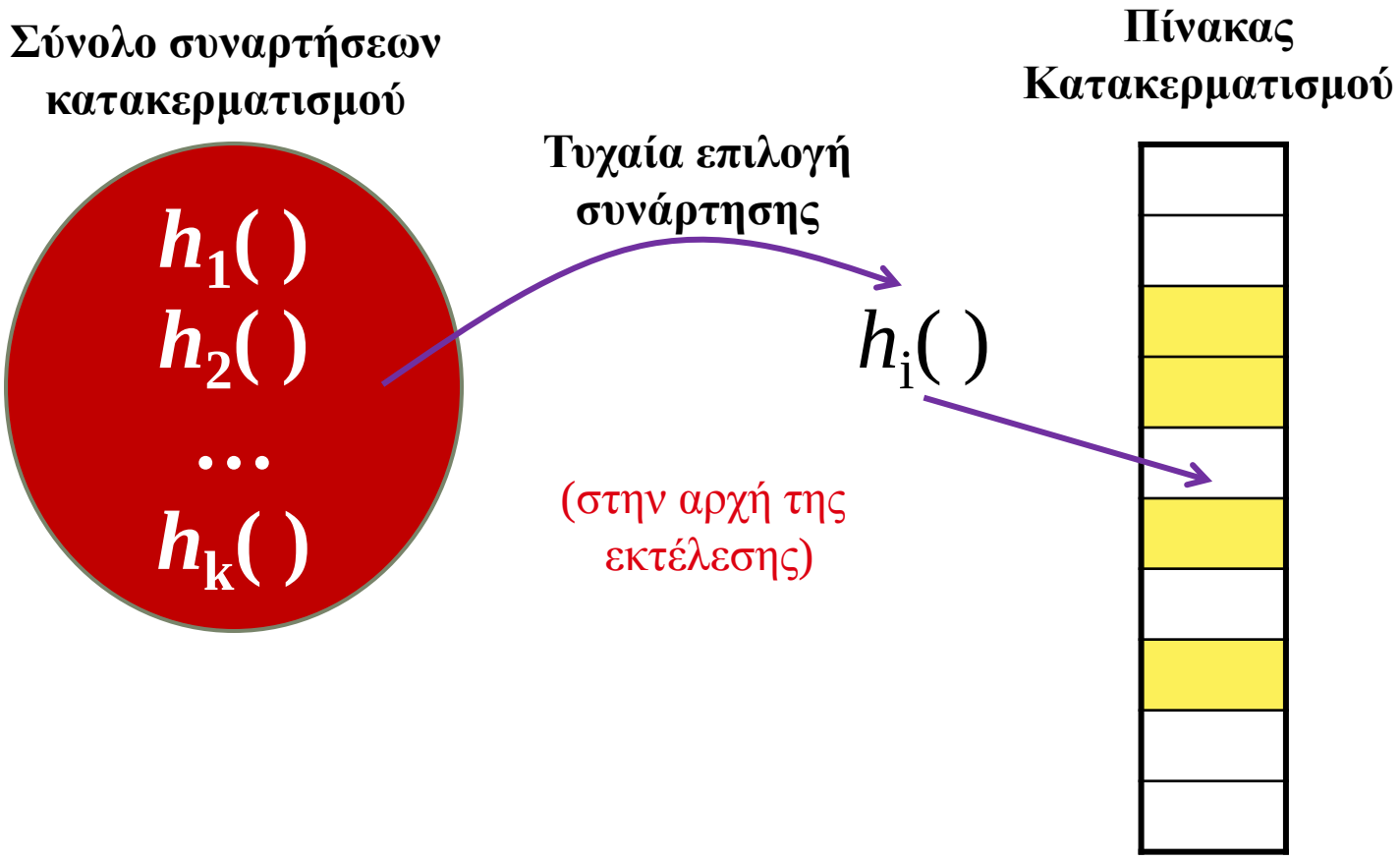
x	x mod 87	x mod 100	$h(x)$
16838	47	38	58
5758	16	58	81
10113	21	13	22
17515	28	15	110
31051	79	51	73
5627	59	27	86
23011	43	11	74
7420	25	20	103
16212	30	12	72
4086	84	86	36
2749	52	49	124
12767	65	67	56
9084	36	84	28
12060	54	60	62
32225	35	25	18
17543	56	43	21
25089	33	89	109
21183	42	83	104
25137	81	37	66
25566	75	66	84
26966	83	66	115
4977	18	77	122
20495	50	95	77
10311	45	11	70
11367	57	67	24

Καθολικός Κατακερματισμός
(Universal Hashing)

Καθολικός Κατακερματισμός (Universal Hashing)

- Το μεγαλύτερο πρόβλημα που έχουν οι προηγούμενες μέθοδοι είναι ότι όταν το μέγεθος του πίνακα hash είναι πολύ μικρότερο από το μέγεθος του σύμπαντος ($m << u$), τότε οποιαδήποτε συνάρτηση h και αν επιλεχθεί θα υπάρχουν ορισμένα μεγάλα υποσύνολα του U (μεγέθους τουλάχιστον $\lceil u/m \rceil$) με την ίδια τιμή hash (άρα αν είναι μέρη του S θα προκαλέσουν πολλές συγκρούσεις).
- Επίσης στην πράξη, τα στοιχεία του S δεν είναι ομοιόμορφα κατανεμημένα, οπότε **η κατανομή των τιμών τους επηρεάζει σημαντικά το πλήθος των συγκρούσεων**.
- **Στόχος:** Συναρτήσεις που παράγουν τυχαίες διευθύνσεις στον πίνακα hash ανεξάρτητα από τα στοιχεία/κλειδιά.
- **Ιδέα:**
  - Επέλεξε μία συνάρτηση τυχαία, από μία **προκαθορισμένη οικογένεια συναρτήσεων**.

Καθολικός Κατακερματισμός (Universal Hashing)



Καθολικός Κατακερματισμός (Universal Hashing)

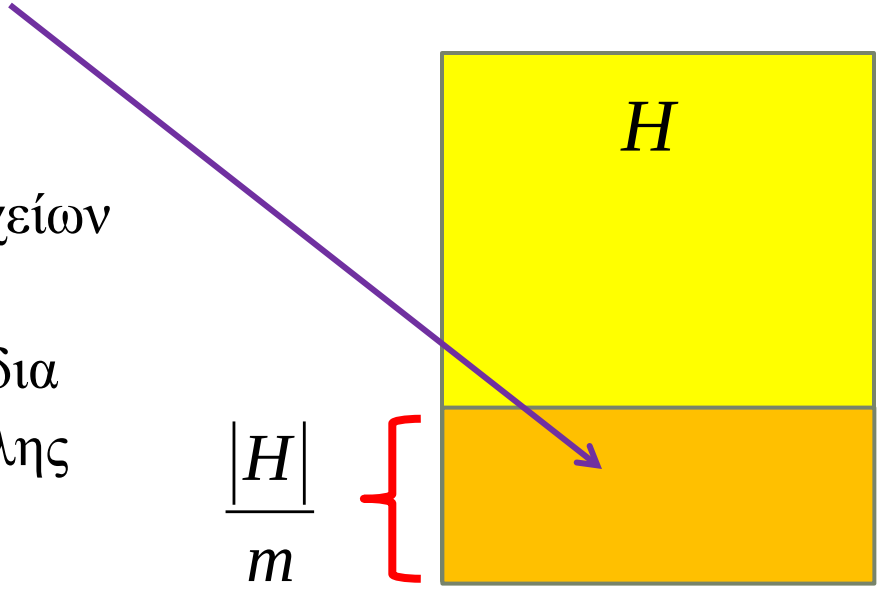
- Έστω $\boldsymbol{H}$ μία οικογένεια συναρτήσεων κατακερματισμού:

$$H=\{\ h(x): U \rightarrow \{0,1,..,m\text{-}1\}\ \}$$

- **Ορισμός:** Η $\boldsymbol{H}$ είναι καθολική αν για κάθε $\boldsymbol{x} \neq \boldsymbol{y}$:

$$\left|h \in H : h(x) = h(y)\right| \leq \frac{|H|}{m}$$

- Δηλαδή όταν για κάθε ζεύγος στοιχείων το πλήθος των συναρτήσεων της οικογένειας που επιστρέφουν την ίδια τιμή hash δεν ξεπερνά το πλήθος όλης της οικογένειας δια $\boldsymbol{m}$.



Καθολικός Κατακερματισμός (Universal Hashing)

- Η ιδιότητα αυτή είναι χρήσιμη διότι βάζει **φράγμα στην πιθανότητα των συγκρούσεων**.
- Πιο συγκεκριμένα: η πιθανότητα σύγκρουσης για δύο τυχαία στοιχεία x, y με $x \neq y$ είναι ίση με την πιθανότητα επιλογής μίας συνάρτησης $h \in H$ έτσι ώστε $x \neq y \Rightarrow h(x) = h(y)$ που είναι ίση με:

$$\Pr\left(h(x) = h(y)\right) \leq \frac{|H|/m}{|H|} = \frac{1}{m}$$

Καθολικός Κατακερματισμός (Universal Hashing)

- **Αποτέλεσμα:** Με τον καθολικό κατακερματισμό η πιθανότητα σύγκρουσης μεταξύ διαφορετικών στοιχείων x και y δεν είναι παραπάνω από $1/m$ εφόσον οι θέσεις $h(x)$ και $h(y)$ επιλέχθηκαν τυχαία και ανεξάρτητα από το σύνολο $\{0, 1, \ldots, m - 1\}$.
- **Στόχος:** Ο σχεδιασμός κατάλληλων οικογενειών συναρτήσεων κατακερματισμού ώστε να είναι καθολικές και να τηρούν την ιδιότητα αυτή.

Μία Οικογένεια Καθολικών Συναρτήσεων

- Επιλέγεται ένας πρώτος αριθμός p αρκετά μεγάλος ώστε κάθε στοιχείο x να ανήκει στο διάστημα $[0 \ldots p-1]$ (πλησιέστερος πρώτος μετά τον αριθμό u). Ορίζουμε: $Z_p = \{0, 1, \ldots, p-1\}$ και $Z_p^* = \{1, \ldots, p-1\}$.
- Ορίζουμε τις συναρτήσεις:

$$h_{a,b}(x) = ((ax + b) \bmod p) \bmod m,$$

$$\forall\ a \in Z_p^* \text{ και } b \in Z_p$$

- Η οικογένεια όλων αυτών των συναρτήσεων είναι η:

$$\mathcal{H}_{p,m} = \{h_{a,b} : a \in Z_p^* \text{ και } b \in Z_p\}$$

- Τα a , b επιλέγονται τυχαία στην αρχή της εκτέλεσης.

Παράδειγμα

- $h_{a,b}(x) = ((ax + b) \bmod p) \bmod m$
- Αν $p = 17$, $m = 6$ και γίνει τυχαία επιλογή $a=3$, $b=4$:

$$h_{3,4}(8) = ((3 \cdot 8 + 4) \bmod 17) \bmod 6$$

$$= (28 \bmod 17) \bmod 6$$

$$= 11 \bmod 6$$

$$= 5$$

- Άρα το στοιχείο $x=8$ θα τοποθετηθεί στην θέση **5**.

Παράδειγμα

α\b	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	2	3	4	5	0	1	2	3	4	0	1	2	3	4	5	0	1	2
2	4	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4
3	1	2	3	4	5	0	1	2	3	4	0	1	2	3	4	5	0	1
4	3	4	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3
5	0	1	2	3	4	5	0	1	2	3	4	0	1	2	3	4	5	0
6	2	3	4	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2
7	5	0	1	2	3	4	5	0	1	2	3	4	0	1	2	3	4	5
8	1	2	3	4	0	1	2	3	4	5	0	1	2	3	4	5	0	1
9	4	5	0	1	2	3	4	5	0	1	2	3	4	0	1	2	3	4
10	0	1	2	3	4	0	1	2	3	4	5	0	1	2	3	4	5	0
11	3	4	5	0	1	2	3	4	5	0	1	2	3	4	0	1	2	3
12	5	0	1	2	3	4	0	1	2	3	4	5	0	1	2	3	4	5
13	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	0	1	2
14	4	5	0	1	2	3	4	0	1	2	3	4	5	0	1	2	3	4
15	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	0	1
16	3	4	5	0	1	2	3	4	0	1	2	3	4	5	0	1	2	3
17	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	0

$\|H\|=$	306
$p=$	17
$m=$	6
$x=$	8

- Παρατηρήστε ότι δεν υπάρχουν εντελώς όμοιες γραμμές ή στήλες.
- Εξασφαλίζεται η τυχαιότητα στην εκλογή της θέσης με πιθανότητα **1/*m***.
- Διατηρούνται οι ιδιότητες αυτές για κάθε στοιχείο ***x***.
- Π.χ. ***x***=13 →

Παράδειγμα

$\alpha \backslash b$	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
1	1	2	3	4	0	1	2	3	4	5	0	1	2	3	4	5	0	1
2	3	4	5	0	1	2	3	4	0	1	2	3	4	5	0	1	2	3
3	5	0	1	2	3	4	5	0	1	2	3	4	0	1	2	3	4	5
4	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	0	1
5	2	3	4	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2
6	4	5	0	1	2	3	4	0	1	2	3	4	5	0	1	2	3	4
7	0	1	2	3	4	5	0	1	2	3	4	0	1	2	3	4	5	0
8	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	0	1	2
9	3	4	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3
10	5	0	1	2	3	4	0	1	2	3	4	5	0	1	2	3	4	5
11	1	2	3	4	5	0	1	2	3	4	0	1	2	3	4	5	0	1
12	3	4	5	0	1	2	3	4	5	0	1	2	3	4	0	1	2	3
13	4	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4
14	0	1	2	3	4	0	1	2	3	4	5	0	1	2	3	4	5	0
15	2	3	4	5	0	1	2	3	4	0	1	2	3	4	5	0	1	2
16	4	5	0	1	2	3	4	5	0	1	2	3	4	0	1	2	3	4
17	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	0

$\|H\|=$	306
$p=$	17
$m=$	6
$x=$	13

Μία Οικογένεια Καθολικών Συναρτήσεων

- Η οικογένεια των συναρτήσεων που ορίσαμε:

  $h_{a,b}(x) = ((ax + b) \bmod p) \bmod m, \qquad \forall\ a \in Z_p^* \text{ και } b \in Z_p$

  έχει μία επιπλέον ιδιότητα ανεξαρτησίας (***pairwise independent***):

- Γενικά μία οικογένεια $\boldsymbol{H}$ είναι $\boldsymbol{k}$-wise ανεξάρτητη αν για κάθε $\boldsymbol{h \in H}$ και για κάθε ομάδα $\boldsymbol{k}$ διακριτών στοιχείων $\boldsymbol{x_1, x_2, \ldots, x_k \in U}$ ισχύει:

$$\Pr\{h(x_1) = t_1 \wedge h(x_2) = t_2 \wedge \ldots h(x_k) = t_k\} = O\left(\frac{1}{m^k}\right)$$

- Η ιδιότητα της $\boldsymbol{k}$-wise ανεξαρτησίας είναι ακόμα πιο ισχυρή από την καθολική ιδιότητα.

Μία ***k***-wise ανεξάρτητη Οικογένεια

- Μία ***k***-wise ανεξάρτητη και καθολική οικογένεια συναρτήσεων είναι η:

$$h(x)=\left[\left(\sum_{i=0}^{k-1} a_i x^i\right) \bmod p\right] \bmod m \quad \text{με} \quad a_{k-1} \in Z_p^* \text{ και } a_i \in Z_p \ (i=0,1,\ldots,k-2)$$

- Και πάλι ο ***p*** είναι ένας πρώτος αριθμός μεγαλύτερος από τον ***u***.
- Η οικογένεια αυτή αποτελεί γενίκευση της προηγούμενης.
- Και στις δύο περιπτώσεις αν ***p<m*** οι μεγαλύτερες θέσεις του πίνακα μπορεί να μην χρησιμοποιούνται, συνεπώς πρέπει να αποφύγουμε μία τέτοια επιλογή.

Άλλη Καθολική Οικογένεια Συναρτήσεων

- Έστω m πρώτος. Αναπαριστούμε το στοιχείο x στο σύστημα αρίθμησης με βάση m. Έστω ότι έχει $r+1$ ψηφία: $x = \langle x_0, x_1, \ldots, x_r \rangle$, όπου $0 \le x_i < m$.
- Επιλέγουμε έναν τυχαίο αριθμό $a = \langle a_0, a_1, \ldots, a_r \rangle$, στο ίδιο σύστημα αρίθμησης όπου κάθε ψηφίο του a_i επιλέχθηκε τυχαία από το σύνολο $\{0, 1, \ldots, m-1\}$.

$$h_a(x) = \sum_{i=0}^{r} a_i x_i \bmod m$$

Εσωτερικό Γινόμενο

- Δεν απαιτείται σε αυτή την οικογένεια πρώτος μεγαλύτερος από τον u.

Απόδοση Καθολικού Κατακερματισμού

- Το κόστος για insert είναι πάντοτε **O(1)**.
- Το κόστος για search, delete στη **μέση περίπτωση** είναι **Θ(1) ανεξαρτήτως των στοιχείων που αποθηκεύονται**.
- Το κόστος για search, delete στη **χειρότερη περίπτωση** είναι: **O(1+*α*)**, όπου ***α=n/m***.
- Εγγυάται ότι **καμία συγκεκριμένη είσοδος στοιχείου δεν θα προκαλεί συμπεριφορά χειρότερης περίπτωσης**. Η συμπεριφορά είναι δηλαδή ανεξάρτητη από μοτίβα και από την κατανομή των στοιχείων του ***S***.
- Την χειρότερη απόδοση την έχουμε μόνο όταν η ίδια η τυχαία επιλογή επιστρέφει μία συνάρτηση κατακερματισμού που θα προκαλέσει σύγκρουση (μικρή πιθανότητα).

Τέλειος Κατακερματισμός
(Perfect Hashing)

Τέλειος Κατακερματισμός (Perfect Hashing)

- Οι καθολικές οικογένειες από πρακτικής πλευράς είναι ότι πιο απλό και αποδοτικό θα ήθελε κανείς.
- Μπορούμε και καλύτερα;
- Το καλύτερο από όλα είναι να έχουμε και για τις 3 πράξεις (insert, delete, search) κόστος χειρότερης περίπτωσης **O(1)**.
- Δυστυχώς αποδείχθηκε από τον Dietzfelbinger ότι είναι αδύνατον να γίνει αυτό.
- Εφόσον δεν υπάρχει τέτοια ελπίδα, το αμέσως επόμενο καλύτερο είναι να έχουμε **search σε O(1).**

Τέλειος Κατακερματισμός (Perfect Hashing)

- Για να συμβεί αυτό θα πρέπει να έχουμε τέλειες συναρτήσεις κατακερματισμού:

  Μία τέλεια συνάρτηση κατακερματισμού για ένα σύνολο S είναι μία συνάρτηση που δεν προκαλεί καμία σύγκρουση.

- Όμως για να μην έχουμε συγκρούσεις θα πρέπει να έχουμε **πολύ μεγάλο χώρο διαθέσιμο**.
- Ο λίγος χώρος πάντα θα προκαλεί συγκρούσεις ανεξάρτητα από την επιλογή των συναρτήσεων.
- Υπάρχει μέση λύση;

Τέλειος Κατακερματισμός (Perfect Hashing)

Μέθοδος **FKS** [Fredman, Komlós, Szemerédi]:

- Βασική ιδέα είναι η χρήση **δύο επιπέδων hash**.
- Χρήση ενός μικρού πίνακα hash ***A*** στο 1ο επίπεδο.
- Κάθε κελί A_i του πίνακα ***A*** είναι ένας νέος πίνακας hash χωρίς συγκρούσεις σε 2ο επίπεδο και επιλύει τις συγκρούσεις στο A_i του 1ου επιπέδου.
- lookup(***x***): αναζήτηση στο επίπεδο 1 για την εύρεση του σωστού κελιού για το επίπεδο 2. Αναζήτηση μετά στο επίπεδο 2.

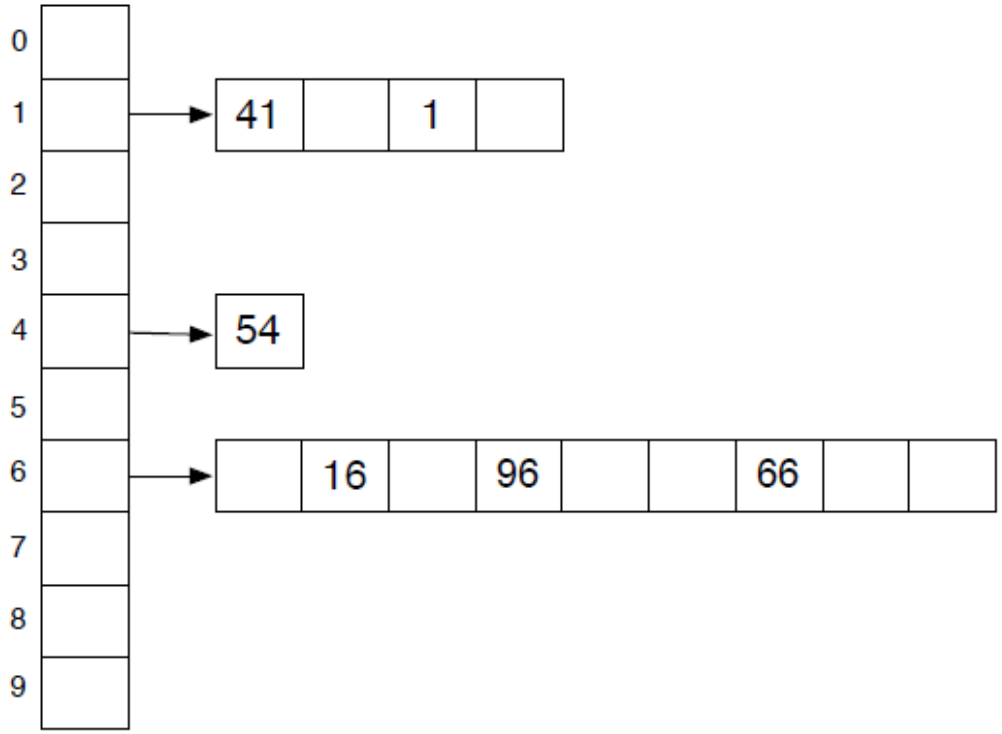
Τέλειος Κατακερματισμός (Perfect Hashing)

- Χρησιμοποιείται μία καθολική οικογένεια συναρτήσεων hash στο **1ο επίπεδο**.
- Όταν έχουμε σύγκρουση στο 1ο επίπεδο τότε επιλέγουμε μία **τέλεια συνάρτηση κατακερματισμού** για το 2ο επίπεδο (κατάλληλη καθολική που να μην προκαλεί καμία σύγκρουση).
- Αποδεικνύεται ότι **η αναμενόμενη πιθανότητα σύγκρουσης, όταν υπάρχει διαθέσιμος χώρος** ώστε να αναπτυχθούν ελεύθερα τα 2 επίπεδα, **είναι μόλις ½.**
- Αυτό σημαίνει ότι **έχουμε 50% πιθανότητα να επιλέξουμε** συνάρτηση (για το 2ο επίπεδο) που να μην προκαλεί σύγκρουση (τέλεια). Αν είμαστε «άτυχοι» και επιλέξουμε κάποια που προκαλεί, απλά επαναλαμβάνουμε με άλλη τυχαία επιλογή. *Δηλαδή σαν να ρίχνουμε ένα νόμισμα μέχρι να φέρει κορώνα.*

Τέλειος Κατακερματισμός - Απόδοση

- Η τελική επιλογή της τέλειας συνάρτησης hash για το 2^o επίπεδο αποθηκεύεται στο αντίστοιχο κελί.
- Με τον τρόπο αυτό μετατρέπουμε την τυχαία εκλογή σε συγκεκριμένη (ντετερμινιστικός αλγόριθμος).
- Αποδεικνύεται ότι ο αναμενόμενος χώρος που απαιτείται είναι $\mathbf{O(n^2/m)}$. Φυσικά αν $\boldsymbol{m=O(n)}$ τότε έχουμε γραμμικό χώρο $\mathbf{O(n)}$.
- Το κόστος αναζήτησης είναι πάντοτε **O(1)** (ακόμα και στη χειρότερη περίπτωση).
- Η δομή χτίζεται με κόστος $\mathbf{O(n)}$ και είναι **στατική** (δεν έχουμε περαιτέρω εισαγωγές-διαγραφές).
- Υπάρχει όμως και λύση για **δυναμική** δομή με **O(1)** κόστος αναζήτησης αλλά με **Θ(1)** για ενημερώσεις (*Dietzfelbinger, Karlin, Mehlhorn, Heide, Rohnert, Tarjan*).

Παράδειγμα



S={1,16,41,54,66,96}

Επίπεδο 1:

S_1 = {1, 41}

S_4 = {54}

S_6 = {16, 66, 96}

Επίπεδο 2: Η πληροφορία για την τέλεια συνάρτηση αποθηκεύεται μέσα στο κελί.


ΤΕΛΟΣ