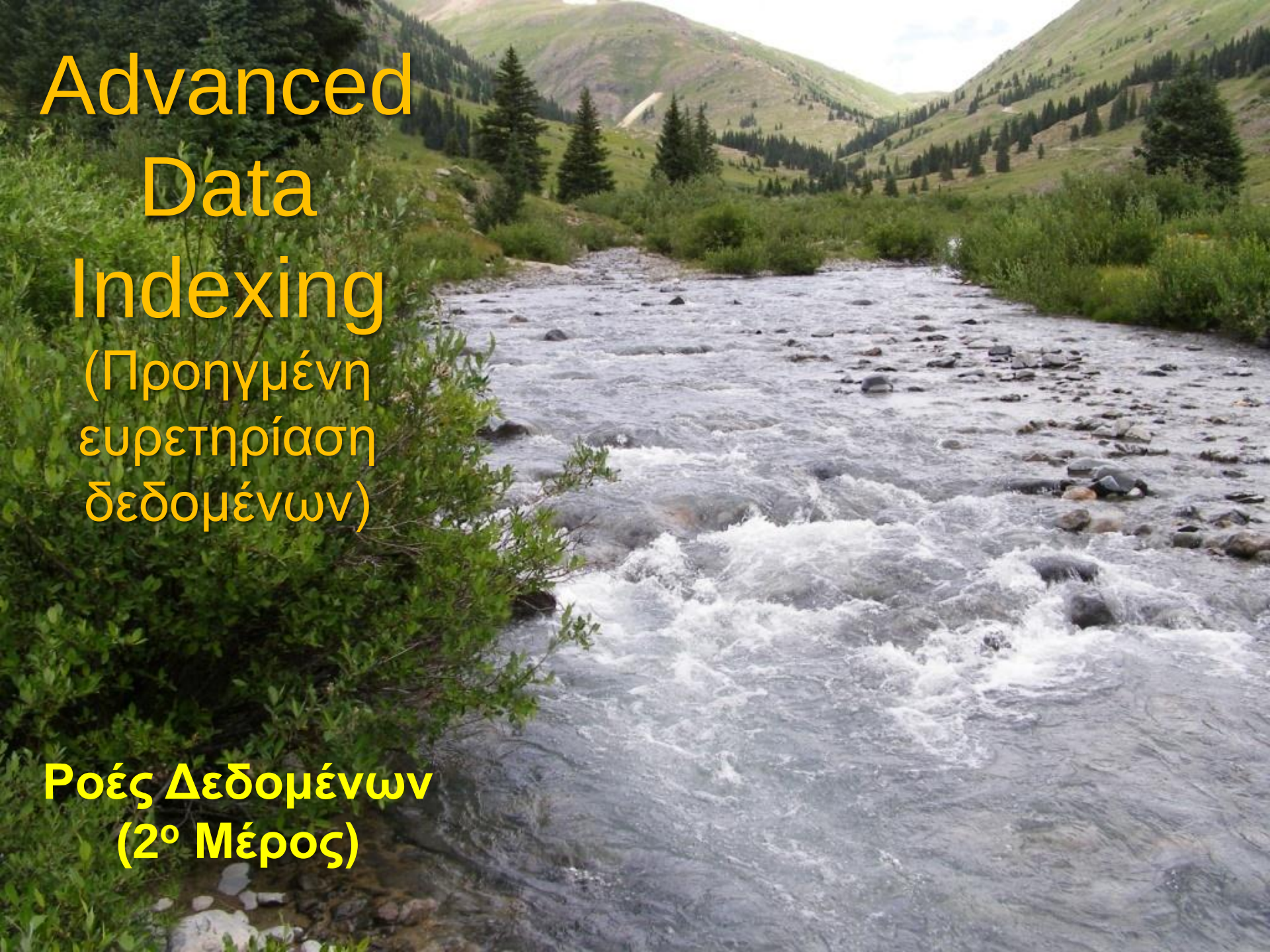


Advanced Data Indexing

(Προηγμένη ευρετηρίαση δεδομένων)

Ροές Δεδομένων (2ο Μέρος)

Το Πρόβλημα των Συχνών Στοιχείων

(Frequent Items – Heavy Hitters)

Παραδείγματα εφαρμογής

- Έχουμε μπροστά μας μία τεράστια ακολουθία από δοσοληψίες (για παράδειγμα από το Amazon)
  - Ποιοι είναι οι κορυφαίοι πωλητές (που έκαναν τις περισσότερες πωλήσεις);
- Ελέγχουμε την κίνηση στο δίκτυο.
  - Ποιες IP πηγές είναι υπεύθυνες για το μεγαλύτερο μέρος της κίνησης στο δίκτυο;
- Έχουμε ένα δίκτυο από δορυφόρους που παρατηρούν πολλές περιοχές.
  - Ποιες είναι οι περιοχές με την μεγαλύτερη κινητικότητα στη διάρκεια μίας εβδομάδας;

Σημαντικότητα

- Εμφανίζεται σε πολλά άλλα προβλήματα και εφαρμογές των ροών.
  - (itemset mining, entropy estimation, compressed sensing, …)
- Είναι από τα προβλήματα που έχουν μελετηθεί περισσότερο και έχουν προταθεί αρκετοί αλγόριθμοι.
- Οι πιο δημοφιλείς αλγόριθμοι χωρίζονται σε δύο κατηγορίες:
  - **Αλγόριθμοι που βασίζονται σε μετρητές (counter-based algorithms): Frequent, Lossy-Counting, Space-Saving.**
  - **Αλγόριθμοι που χρησιμοποιούν sketches: Hierarchical Search + Group Testing, Count-Min Sketch, Count Sketch.**

Αλγόριθμοι που βασίζονται σε μετρητές (counter-based algorithms)

- **Frequent Algorithm**
- **Lossy-Counting Algorithm**
- **Space-Saving Algorithm**

Το Γενικό Πρόβλημα

Φανταστείτε ότι παρατηρούμε μία ροή δεδομένων.

Αυτή η ροή αποτελείται από αριθμούς στο διάστημα [**1**,…,***U***], όπου ***U*** είναι ένα πάνω φράγμα στα στοιχεία.

Μας ενδιαφέρει να βρούμε τους αριθμούς εκείνους που εμφανίζονται πιο συχνά στη ροή.

Παράδειγμα:

1, 5, 8, 9, 10, 3, 23542341234, 15, 289, 31516, 23, 8, 3571, 1251, 8, 5, 124, 289, 17, 15, 23542341234, 126, 1251, 5,…

Αν τα στοιχεία δεν είναι ακέραιοι;

Συνήθως μπορούμε να αντιστοιχήσουμε τα στοιχεία σε ακέραιους:

- (2310) 998-765 → 2310998765
- 128.6.75.111 → 111+256(75+256(6+256*128))) = 2147896175 (32 bits)
- «δειγματοληψία» → 13 bytes

Χρησιμοποιούμε συναρτήσεις κατακερματισμού για να τα αντιστοιχήσουμε σε ακεραίους.

Τι γίνεται με εικόνες, ήχους, κλπ.;

Μία απλή λύση;

- Φτιάχνουμε έναν πίνακα μεγέθους ***U***. Προσθέτουμε 1 στην ***i***-οστή θέση αν εμφανιστεί στη ροή το στοιχείο ***i***. Έπειτα βρίσκουμε τις μεγαλύτερες τιμές διασχίζοντας τον πίνακα.

Αυτή η λύση όμως δεν δουλεύει…

- Για IP διευθύνσεις ο πίνακας θα έχει μέγεθος 4GB.
- Για ζευγάρια IP διευθύνσεων: 16384PB!
- Ομοίως και για τις δοσοληψίες στο Amazon.

Ακόμα και αν είχαμε διαθέσιμο χώρο, ο χρόνος για την διαπέραση του πίνακα θα είναι τεράστιος.

Απλοποίηση του προβλήματος

- Ας απλοποιήσουμε λίγο το πρόβλημα:
- Μπορούμε να βρούμε αν υπάρχει μέσα σε μία ροή μεγέθους ***n*** ένα στοιχείο που εμφανίζεται παραπάνω από ***n/2*** φορές (**στοιχείο πλειοψηφίας**); Π.χ.
  - **Ροή: 1 1 2 2 2 2 3 1 1 3 1 1 3 1 1 3 1 1 1 1 3 1**
  - **Στοιχείο Πλειοψηφίας: 1**
- Το πρόβλημα αυτό είναι ειδική περίπτωση του γενικότερου προβλήματος των συχνών στοιχείων που θέλουμε να λύσουμε.
- Προτάθηκε το 1981 από τον J. S. Moore στο Journal of Algorithms. Ο αλγόριθμος για τη λύση του ονομάστηκε Majority Algorithm.

Αλγόριθμος Στοιχείου Πλειοψηφίας
(majority algorithm)

- Αν είχαμε **O(*n*)** διαθέσιμη μνήμη θα ήταν άμεση η επίλυση του προβλήματος. Όμως δεν έχουμε.
- Μπορούμε να το λύσουμε με **O(1)** μνήμη;
- Αλγόριθμος δύο περασμάτων:
  - **1ο πέρασμα:** Εύρεση του πιθανού υποψήφιου στοιχείου.
  - **2ο πέρασμα:** Υπολογισμός της συχνότητάς του και επαλήθευση του αν είναι μεγαλύτερη από ***n*/2**.

Αλγόριθμος Στοιχείου Πλειοψηφίας
(majority algorithm)

- (1ο πέρασμα) Εύρεση του πιθανού υποψήφιου στοιχείου:
  - Αρχικοποίηση ενός μετρητή στο 0 και ενός κάδου 1 θέσης.
  - Για κάθε στοιχείο της ροής που διαβάζεται:
    - Αν ο μετρητής είναι 0, βάζουμε το στοιχείο στον κάδο (αντικαθιστώντας τυχόν άλλο που υπάρχει) και θέτουμε τον μετρητή στο 1.
    - Αλλιώς, αν το στοιχείο είναι ίδιο με αυτό του κάδου, αυξάνουμε τον μετρητή κατά 1.
    - Αλλιώς μειώνουμε το μετρητή κατά 1.
- (2ο πέρασμα) Υπολογισμός συχνότητας:
  - Το στοιχείο που έμεινε στον κάδο στο τέλος της ροής είναι το υποψήφιο στοιχείο πλειοψηφίας.
  - Με το 2ο πέρασμα υπολογίσουμε τη συχνότητά του με έναν μετρητή και επαληθεύουμε αν είναι μεγαλύτερη από $\boldsymbol{n/2}$.

Παράδειγμα 1

(1ο πέρασμα):

Ροή : 1 1 2 2 2 2 3 1 1 3 1 1 3 1 1 3 1 1 1 1 3 1 (***n***=22)

Κάδος : 1 1 1 1 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1

Μετρητής: 1 2 1 0 1 2 1 0 1 0 1 2 1 2 3 2 3 4 5 6 5 6

Υποψήφιο στοιχείο: **1**

(2ο πέρασμα):

Ροή : 1 1 2 2 2 2 3 1 1 3 1 1 3 1 1 3 1 1 1 1 3 1

Μετρητής: 1 2 2 2 2 2 2 3 4 4 5 6 6 7 8 8 9 10 11 12 12 13

Ισχύει 13>22/2 άρα το 1 είναι το στοιχείο πλειοψηφίας.

Παράδειγμα 2

(1ο πέρασμα):

Ροή : 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 1 2 3 (***n***=18)

Κάδος : 1 1 3 3 1 1 3 3 1 1 3 3 1 1 3 3 2 2

Μετρητής: 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0

Υποψήφιο στοιχείο: **2**

(2ο πέρασμα):

Ροή : 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 1 2 3

Μετρητής: 0 1 1 1 1 2 2 2 2 3 3 3 3 4 4 4 5 5

Ισχύει 5<18/2 άρα δεν υπάρχει στοιχείο πλειοψηφίας.

Ορθότητα Αλγόριθμου

Φανταστείτε ότι βάζουμε σε ζευγάρια όσες εμφανίσεις του στοιχείου πλειοψηφίας μπορούμε με άλλα στοιχεία της ροής:

Τότε θα υπάρχουν επιπλέον εμφανίσεις του στοιχείου πλειοψηφίας που δεν είναι σε ζευγάρια:

Κάθε ζευγάρι αλληλοαναιρείται (έχουμε μείωση και αύξηση κατά 1 στον μετρητή):

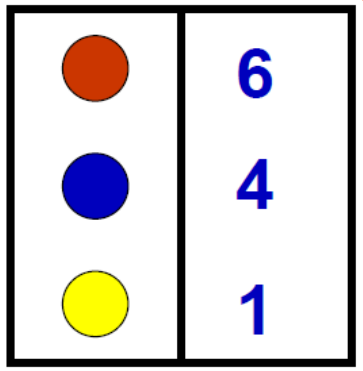
Επομένως, στο τέλος θα έχουμε ένα θετικό πλήθος εμφανίσεων για το στοιχείο πλειοψηφίας ανεξαρτήτως πως θα είναι τα ζευγάρια.

Αλγόριθμος Στοιχείου Πλειοψηφίας
(majority algorithm)

- Λύνεται με **ένα πέρασμα**;
  - Όχι πάντα.
  - **Αν υπάρχει στοιχείο πλειοψηφίας** τότε όπως αποδείξαμε ο αλγόριθμος ακόμα και με ένα πέρασμα θα το βρει σωστά (1ο παράδειγμα).
  - **Αν δεν υπάρχει** όμως **στοιχείο πλειοψηφίας** τότε το στοιχείο που επιστρέφεται με ένα πέρασμα δεν αποτελεί τη σωστή απάντηση (απαιτείται και το 2ο πέρασμα για τον έλεγχο).
- Ο αλγόριθμος αυτός γενικεύτηκε για ***k*** στοιχεία το 1982 από τους Misra and Gries με το όνομα Frequent Algorithm, ενώ ανακαλύφθηκε ξανά το 2002 με ακόμα πιο αποδοτικές υλοποιήσεις και όρια που διατυπώθηκαν το 2003 [Bose et al.].

Αλγόριθμος Συχνότητας
(Frequent Algorithm)

Ο αλγόριθμος συχνότητας βρίσκει κάθε στοιχείο με συχνότητα μεγαλύτερη από $n/(k+1)$ χρησιμοποιώντας k κάδους μίας θέσης και k μετρητές.



Αλγόριθμος Συχνότητας:

Για κάθε στοιχείο της ροής που διαβάζεται:

- Αν το στοιχείο είναι σε κάποιο κάδο (από τους k), αυξάνουμε κατά 1 τον αντίστοιχο μετρητή.
- Αλλιώς, αν ένας μετρητής είναι 0, τοποθετούμε το στοιχείο σε αυτόν τον κάδο (αντικαθιστώντας τυχόν άλλο στοιχείο) και κάνουμε τον αντίστοιχο μετρητή 1.
- Αλλιώς, μειώνουμε όλους τους μετρητές κατά 1.

Ορθότητα Αλγορίθμου

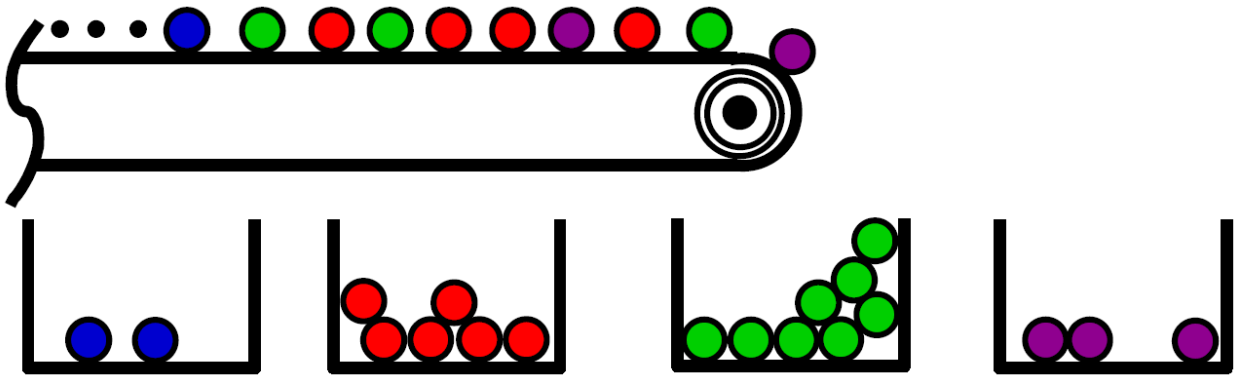
Ιδιότητα: Κάθε στοιχείο με συχνότητα μεγαλύτερη από $n/(k+1)$ θα υπάρχει σε έναν κάδο στο τέλος της ροής.

Απόδειξη: Αντίστοιχη με την προηγούμενη.

- Απαιτούμενος Χώρος: $O(k)$
- Με ένα μόνο πέρασμα μπορούμε να αναφέρουμε τα υποψήφια στοιχεία ως απάντηση αλλά **δεν έχουμε ακόμα φράγματα για το σφάλμα.**
- Απαιτείται **νέο μοντέλο** για το πρόβλημα ώστε να υποστηρίζει προσεγγίσεις με δεδομένα όρια σφάλματος.

Το Τελικό Πρόβλημα

- Το πρόβλημα των συχνών στοιχείων (Frequent Items – Heavy Hitters), διατυπώνεται τελικά ως εξής:
  - Αν έχουμε μία ροή που αποτελείται από n στοιχεία, μας ενδιαφέρει να βρούμε τα στοιχεία που εμφανίζονται με συχνότητα $\geq f \cdot n$, όπου $0<f<1$ ένα προκαθορισμένο όριο, και κανένα με συχνότητα $<(f-\varepsilon)\cdot n$, όπου $0<\varepsilon<1$ ένα προκαθορισμένο σφάλμα.
  - Η εκτίμηση κάθε συχνότητας πρέπει να γίνεται με σφάλμα $\pm\varepsilon \cdot n$.



Απόδοση Αλγορίθμου Συχνότητας

- Στο τελικό πρόβλημα ο αλγόριθμος εφαρμόζεται ακριβώς με τον ίδιο τρόπο, θέτοντας: $f=1/(k+1)$.
- Με ένα μόνο πέρασμα της ροής έχουμε την τελική εκτίμηση των k συχνών στοιχείων και μάλιστα **με όριο για το σφάλμα**:
- Η χρήση $k=1/\varepsilon$ μετρητών και κάδων εκτιμά τις συχνότητες με σφάλμα $\pm\varepsilon \cdot n$ [Bose et al., 2003].
- Απαιτούμενος **Χώρος**: $O(k)=O(1/\varepsilon)$.

Αλγόριθμος Lossy Counting

Ο αλγόριθμος αυτός προτάθηκε το 2002 από τους Manku, Motwani και η κεντρική του ιδέα είναι η **ομαδοποίηση στοιχείων** και η άθροιση των αντιστοίχων μετρητών τους.

Αλγόριθμος Lossy Counting (απλή έκδοση):

- Κάθε στοιχείο της ροής που διαβάζεται αποθηκεύεται σε μία ομάδα στην μνήμη (αν δεν υπάρχει ήδη). Αν το στοιχείο υπάρχει τότε αυξάνουμε κατά 1 τον αντίστοιχο μετρητή του (την συχνότητά του).
- Κάθε φορά που μία ομάδα φτάσει σε πλήθος τα **$1/\varepsilon$** στοιχεία, τα στοιχεία της συγχωνεύονται και οι αντίστοιχοι μετρητές τους αθροίζονται.
- Όλοι οι υπόλοιποι μετρητές μειώνονται κατά 1 ενώ σβήνονται όλα τα στοιχεία που έχουν μηδενικό μετρητή.

Απόδοση Αλγορίθμου Lossy Counting

- Από την ανάλυση της μεθόδου προκύπτει ότι ο απαιτούμενος χώρος αποθήκευσης για τα εισερχόμενα στοιχεία και μέχρι το τέλος της ροής είναι: $\mathbf{O(1/\varepsilon \cdot \log(\varepsilon \cdot n))}$, όπου ε το σφάλμα.
- Οι συχνότητες εκτιμώνται με σφάλμα $\pm\varepsilon \cdot n$.
- Στην πλήρη έκδοση του αλγορίθμου κρατούνται επιπλέον πληροφορίες για τις ομάδες επιτυγχάνοντας περαιτέρω μείωση του σφάλματος.

Αλγόριθμος Space Saving

Ο αλγόριθμος αυτός προτάθηκε το 2005 από τους Metwally, Agrawal, El Abaddi και η κεντρική του ιδέα είναι η **συγχώνευση** των δύο προηγούμενων αλγορίθμων (**Loosy Counting** και **Frequent**). Χρησιμοποιεί ***k*** κάδους μίας θέσης και ***k*** αντίστοιχους μετρητές, όπου $k=1/\varepsilon$.

Αλγόριθμος Space Saving:

- Αρχικοποιούνται οι μετρητές με **0**.
- Τα πρώτα ***k*** διαφορετικά στοιχεία τοποθετούνται στους κάδους ενώ αυξάνονται οι αντίστοιχοι μετρητές τους όσο διαβάζονται επαναλήψεις τους. Όσο δηλαδή διαβάζεται στοιχείο που υπάρχει, αυξάνουμε κατά **1** τον αντίστοιχο μετρητή του (ενημερώνοντας την συχνότητά του).
- Αν διαβαστεί στοιχείο από τη ροή που δεν υπάρχει σε κάποιον από τους ***k*** κάδους, τότε αυτό **αντικαθιστά το στοιχείο** των κάδων **που έχει την μικρότερη συχνότητα**, ενώ ο αντίστοιχος μετρητής γίνεται **1**.

Ανάλυση Αλγορίθμου Space Saving

Ιδιότητα: Βρίσκει κάθε στοιχείο με συχνότητα μεγαλύτερη από $n/(k+1)$, με σφάλμα στις συχνότητες όχι περισσότερο από $\pm\varepsilon \cdot n$.

Απόδειξη:

Αρχικά θα αποδείξουμε ότι η μικρότερη τιμή των μετρητών (έστω **min**) είναι το πολύ $\varepsilon \cdot n$:

- Πράγματι, η μέση τιμή όλων των μετρητών είναι $\varepsilon \cdot n$ καθώς το άθροισμα όλων των συχνοτήτων είναι n και διαιρώντας με το πλήθος τους k προκύπτει $n/(1/\varepsilon) = \varepsilon \cdot n$.
- Εφόσον η μέση τιμή τους είναι $\varepsilon \cdot n$ δεν μπορεί ο μικρότερος να είναι πάνω από την τιμή αυτή, άρα $\textbf{min} \leq \varepsilon \cdot n$.

Κατόπιν θα αποδείξουμε ότι η πραγματική συχνότητα κάθε αποθηκευμένου στοιχείου έχει σφάλμα το πολύ $\varepsilon \cdot n$:

- Πράγματι, με χρήση της μαθηματικής επαγωγής, καθώς διατρέχουμε την ροή, η τιμή του **min** αυξάνεται μονότονα (δεν μπορεί όμως να ξεπεράσει την τιμή $\varepsilon \cdot n$)

Ανάλυση Αλγορίθμου Space Saving

Τέλος θα αποδείξουμε ότι κάθε στοιχείο x που έχει πραγματική συχνότητα μεγαλύτερη από $\varepsilon \cdot n$ αποθηκεύεται σε κάποιον κάδο. Με απαγωγή σε άτοπο:

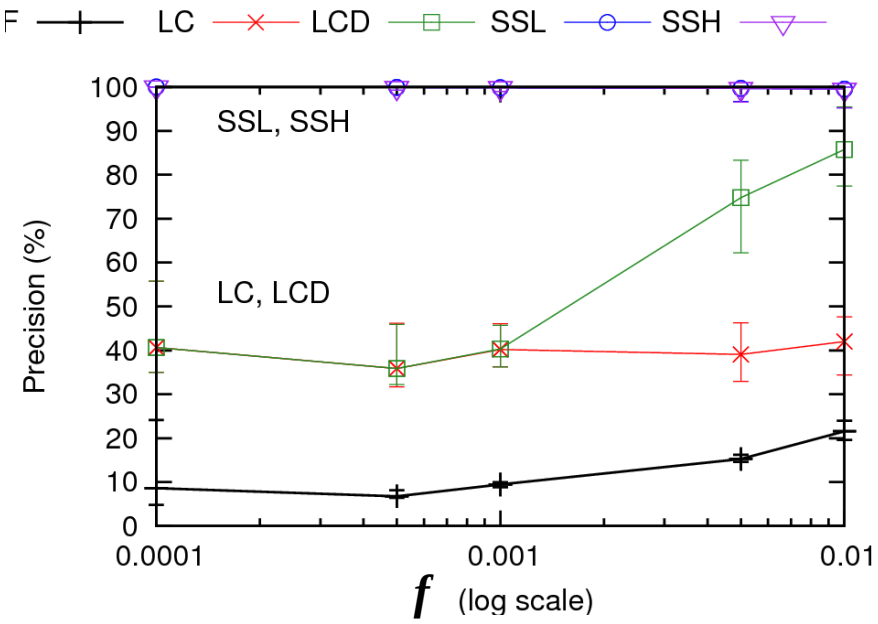
- Έστω αντίθετα ότι το στοιχείο x έχει φύγει από τους κάδους και αντικαταστάθηκε από άλλο στοιχείο.
- Αν $\mathbf{min}_t$ ήταν η τιμή του ελαχίστου τη στιγμή της αντικατάστασης του x, τότε η συχνότητα του x θα ήταν $\leq \mathbf{min}_t$.
- Όμως επειδή η **min** αυξάνεται μονότονα μέχρι το τέλος της ροής, θα ισχύει: συχνότητα(x) $\leq \mathbf{min}_t \leq$ **min** $\leq \varepsilon \cdot n$. Άτοπο.

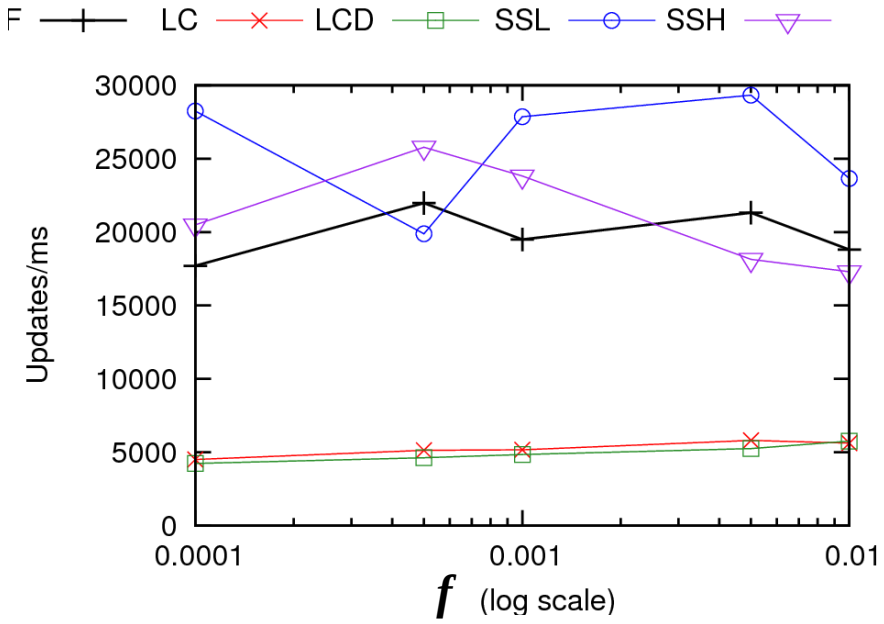
- Απαιτούμενος **Χώρος**: $O(k)=O(1/\varepsilon)$.

Υλοποιήσεις-Πειράματα

- Για τους αλγορίθμους που βασίζονται σε μετρητές (counter-based algorithms) υπάρχουν διάφορες υλοποιήσεις. Κάποιες υλοποιήσεις τους βρίσκονται στη διεύθυνση:
  - http://www.research.att.com/~marioh/frequent-items
- Πειράματα που έγιναν επιβεβαίωσαν τις θεωρητικές αναλύσεις.
- Όλοι οι αλγόριθμοι αυτοί φαίνεται να έχουν παρόμοια απόδοση χειρότερης περίπτωσης: **χρησιμοποιώντας χώρο $O(1/\varepsilon)$ δίνουν προσεγγιστική λύση με ακρίβεια $\pm\varepsilon \cdot n$.**
- Συχνά στην πράξη οι αλγόριθμοι αυτοί είναι πιο ακριβείς από όσο καθορίζουν τα θεωρητικά όρια.
- Συγκρίσεις έγιναν σε μία μεγάλη ποικιλία δεδομένων ροών: web, network, synthetic data, κλπ.

Υλοποιήσεις-Πειράματα





- **F** (Frequency), **LC-LCD** (Lossy Counting), **SSL-SSH** (Space Saving)
- Και οι δύο υλοποιήσεις του αλγορίθμου **Space Saving** επιτυγχάνουν τέλεια ακρίβεια χρησιμοποιώντας μικρό χώρο (10KB-1MB).
- Υποστηρίζουν γρήγορες ροές (μπορούν να επιτύχουν 20M-30M ενημερώσεις το δευτερόλεπτο).

Συμπεράσματα

- Οι αλγόριθμοι που βασίζονται σε μετρητές (**counter-based algorithms**) είναι πολύ αποδοτικοί εφόσον με χώρο **O(1/ε)** δίνουν προσεγγιστική λύση με ακρίβεια **±ε·n**.
- Είναι πολύ γρήγοροι στην πράξη υποστηρίζοντας αρκετά εκατομμύρια ενημερώσεις το δευτερόλεπτο.
- Είναι παρόμοιοι, αλλά υπάρχει σαφώς ένας «νικητής», ο **Space Saving Algorithm**.
- **Κυριότερο Μειονέκτημα:** Εφαρμόζονται μόνο στο Cash-Register Model (αφίξεων) και όχι στο γενικότερο μοντέλο Turnstile Model (αφίξεων και αναχωρήσεων).

Αλγόριθμοι που χρησιμοποιούν sketches

- Hierarchical Search Algorithm + Group Testing
- Count-Min Sketch Algorithm
- Count Sketch Algorithm

Τι γίνεται στο Μοντέλο του Μύλου;
(δυναμική πλειοψηφία)

- Οι προηγούμενες μέθοδοι δεν μπορούν να χειριστούν αφίξεις και αναχωρήσεις στοιχείων. Ας δούμε αρχικά ποια είναι η διαφορά για το πρόβλημα του στοιχείου πλειοψηφίας στο μοντέλο αυτό:
- Παράδειγμα: (+***i*** σημαίνει ότι το ***i*** φτάνει, -***i*** σημαίνει ότι το ***i*** αναχωρεί)

  +1 +1 +1 +2 +3 +4 +4 +4 +4 +4 +4 +4

  οπότε το στοιχείο πλειοψηφίας είναι το **4**. Όμως αν η ροή συνεχίζεται ως εξής:

  -4 -4 -4 -4 -4 -4 -4

  τότε το στοιχείο πλειοψηφίας είναι το **1**.
- Ντετερμινιστικές λύσεις τώρα δεν υπάρχουν. Έτσι καταφεύγουμε στους Sketch Algorithms.

Η ιδέα της Ιεραρχικής Προσέγγισης

- Έστω ότι **χωρίζουμε** τα στοιχεία που βρίσκονται στη ροή **σε δύο ομάδες** με οποιονδήποτε απλό τρόπο (π.χ. όσα είναι στο διάστημα **[1,*U*/2]** και όσα είναι στο **[*U*/2, *U*]**).
- Αν το πλήθος των στοιχείων μίας ομάδας είναι **>*n*/2** τότε αυτό σημαίνει ότι το πλειοψηφικό στοιχείο **πιθανόν** να είναι σε αυτή την ομάδα.
- Το σίγουρο όμως είναι ότι **κανένα από τα στοιχεία της άλλης ομάδας δεν μπορεί να είναι πλειοψηφικό στοιχείο**.
- Έτσι μπορούμε να αποκλείουμε ολόκληρες ομάδες στοιχείων από την απάντηση (pruning) και να συνεχίζουμε αναδρομικά στις ομάδες που έμειναν.
- Ο χώρος αναζήτησης περιορίζεται σημαντικά.

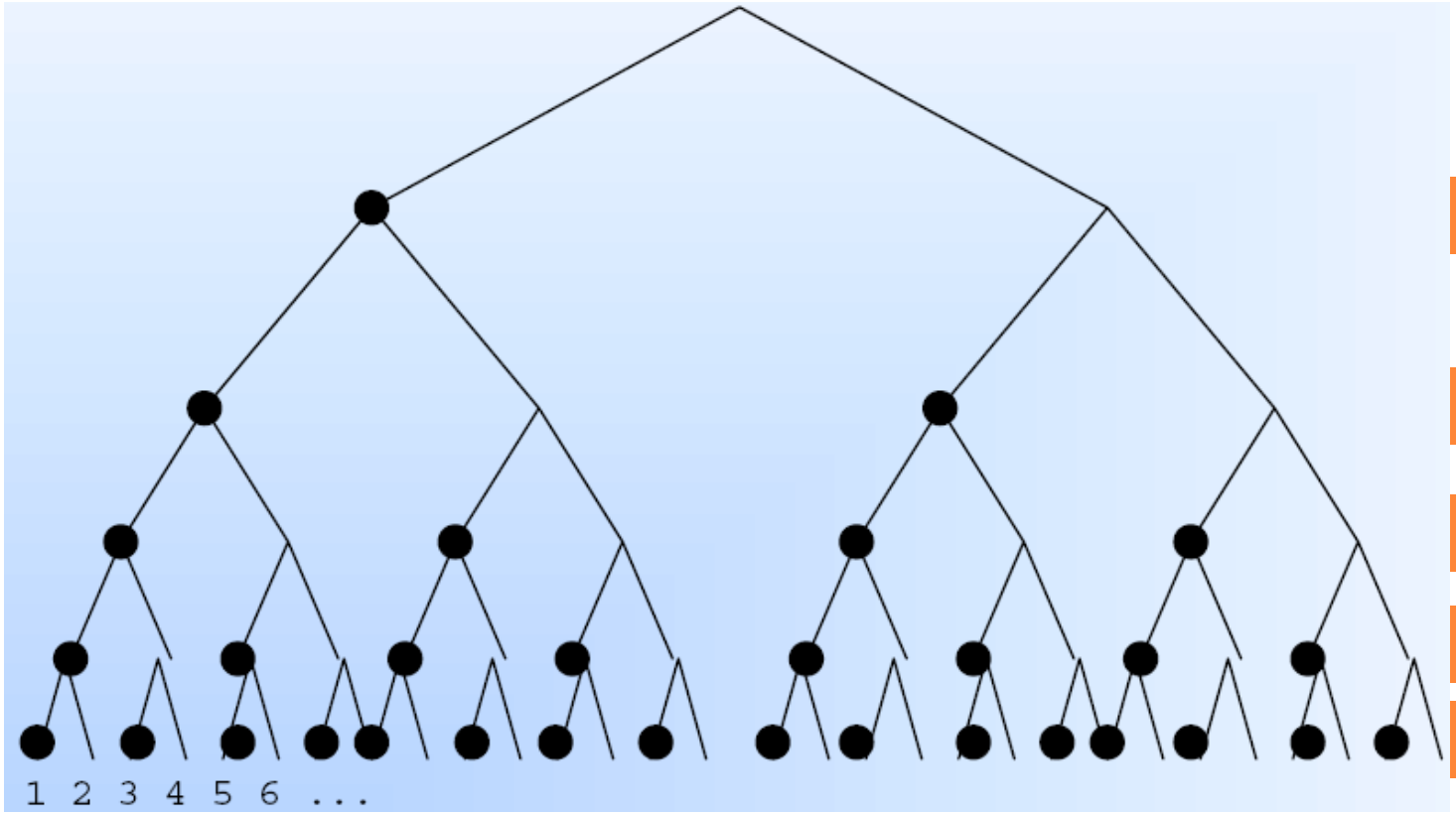
Ο αλγόριθμος Hierarchical Search

- Είναι ένας αναδρομικός αλγόριθμος divide-and-conquer ο οποίος ακολουθεί ένα δυαδικό δέντρο (binary tree) πάνω στο πεδίο ορισμού **[1,*U*]** των στοιχείων της ροής.
- Σε κάθε επίπεδο του δέντρου χρησιμοποιείται ένα κατάλληλο sketch που βοηθά στον εντοπισμό των συχνών στοιχείων (ή του στοιχείου πλειοψηφίας ανάλογα με το ποιο πρόβλημα λύνουμε) στους κόμβους.
- Δυαδική Αναζήτηση:
  - Βρίσκουμε αν το στοιχείο πλειοψηφίας ή κάποιο συχνό στοιχείο είναι στο διάστημα **[1,*U*/2]** ή στο διάστημα **[*U*/2, *U*]**.
  - Αναδρομικά συνεχίζουμε στο κατάλληλο υποδιάστημα.
- Ιδιότητα του δέντρου: όλοι οι πρόγονοι ενός συχνού στοιχείου είναι επίσης συχνά στοιχεία.

Παράλληλη Δυαδική Αναζήτηση

- Μπορούμε να κάνουμε πολλές αναζητήσεις ταυτόχρονα κατασκευάζοντας sketches στα διάφορα επίπεδα του δέντρου που αφορούν ομάδες διαστημάτων. Για παράδειγμα:
  - Η πρώτη ομάδα θα είναι το διάστημα **[1,*U*/2]**.
  - Τα διαστήματα που ορίζονται στο επόμενο επίπεδο είναι: **[1,*U*/4], (*U*/4,*U*/2], (*U*/2,3*U*/4], (3*U*/4,*U*]**.
  - Η δεύτερη ομάδα θα περιέχει τα διαστήματα: **[1,*U*/4], (*U*/2,3*U*/4]**
  - κλπ.
- Από το sketch της πρώτης ομάδας μπορούμε να βρούμε σε ποιο μισό ανήκει το στοιχείο πλειοψηφίας ενώ μαζί με το sketch της δεύτερης σε ποιο τέταρτο ανήκει, κλπ.

Δομή των Ομάδων



Ομάδα 1

Ομάδα 2

Ομάδα 3

Ομάδα 4

Ομάδα 5

- Θα έχουμε συνολικά $\log U$ ομάδες (sketches).
- Πώς όμως ορίζουμε σε ποια ομάδα θα ανήκει το κάθε στοιχείο;

Τοποθέτηση Στοιχείων στις Ομάδες

Ιδέα: Έστω το στοιχείο ***i***. Τότε, το ***i*** μπαίνει σε εκείνες τις ομάδες για τις οποίες τα αντίστοιχα bits της δυαδικής αναπαράστασής του είναι **1**.

Παράδειγμα: Το **13=00001101** μπαίνει στις ομάδες **1**, **3** και **4**. Δηλαδή προσθέτουμε **1** στους αντίστοιχους μετρητές τους.

Όταν στη ροή έχουμε άφιξη του ***i*** τότε προσθέτουμε **1** στις αντίστοιχες ομάδες, ενώ όταν έχουμε αναχώρηση αφαιρούμε **1**.

Απόδειξη Ορθότητας

- Έστω η δυαδική αναπαράσταση του πλειοψηφικού στοιχείου.
- Αν το bit ***j*** του στοιχείου είναι **1** τότε η ομάδα ***j*** θα έχει **μετρητή μεγαλύτερο από το μισό του μεγέθους της ροής**.
- Άρα, κοιτώντας τους μετρητές στα sketches μπορούμε να εντοπίσουμε το πλειοψηφικό στοιχείο οποιαδήποτε στιγμή.
- Επίσης μπορούμε να υποστηρίξουμε και αναχωρήσεις από τη στιγμή που οι μετρητές επιτρέπεται και να μειώνονται κατά **1**.

Παράδειγμα

+1 +1 +1 +2 +3 +4 +4 +4 +4 +4 +4 +4

Σύνολο Στοιχείων: **12**

Στοιχεία: **1→001, 2→010, 3→011, 4→100** (έχουμε 3 ομάδες)

1η Ομάδα: **{1[3], 3[1]}** 2η Ομάδα: **{2[1], 3[1]}** 3η Ομάδα: **{4[7]}**

Πλήθος-1ης=3+1=4 Πλήθος-2ης=1+1=2 Πλήθος-3ης=**7>12/2**

Άρα το στοιχείο πλειοψηφίας είναι το **4**.

Όμως αν η ροή συνεχιστεί ως εξής:

-4 -4 -4 -4 -4 -4 -4

Τελικό Σύνολο Στοιχείων: **5**

1η Ομάδα: **{1[3], 3[1]}** 2η Ομάδα: **{2[1], 3[1]}** 3η Ομάδα: **{4[0]}**

Πλήθος-1ης=3+1=**4>5/2** Πλήθος-2ης=1+1=2 Πλήθος-3ης=0

Άρα το στοιχείο πλειοψηφίας είναι το **1** καθώς ισχύει και **3>5/2**

Μία Χρήσιμη Ιδιότητα

- Έστω ότι σε διαφορετικούς τερματικούς σταθμούς επιβλέπουμε διαφορετικές ροές (ή τμήματα της ίδιας) για την εύρεση του πιο συχνού στοιχείου.
  - Μπορούν να **συνδυαστούν** οι παρατηρήσεις τους;
- Με την ιεραρχική μέθοδο των ομάδων: **ΝΑΙ**.
  - Ο μόνος περιορισμός είναι ότι πρέπει σε κάθε σταθμό να έχουν οριστεί οι ίδιες ομάδες (ίδιο δυαδικό δέντρο σε ίδιο πεδίο ***U***)
  - Προσθέτουμε απλά τους αντίστοιχους μετρητές των σταθμών για την κάθε μία ομάδα.
  - Το αποτέλεσμα που θα πάρουμε θα είναι ίδιο με το αν είχαμε όλες τις ροές σε μία ενιαία.
- Μπορούμε λοιπόν να κατανέμουμε την διαδικασία αυτή πολύ εύκολα.

Γενίκευση

- Η τεχνική των ομάδων είναι ευέλικτη και μπορεί να γενικευτεί και για το πρόβλημα των συχνών στοιχείων:
- Διαδικασία για τα ***k*** πιο συχνά στοιχεία:
  - Επιλέγουμε ***S*** υποσύνολα του σύμπαντος (υποσύνολα του **[1,*U*]**)
  - Για κάθε στοιχείο ***i*** που διαβάζεται από τη ροή:
    - Για κάθε υποσύνολο s_j που περιέχει το ***i***:
      - Αφαιρούμε ή προσθέτουμε 1 στους μετρητές των κατάλληλων ομάδων (ανάλογα με το αν έχουμε άφιξη ή αναχώρηση)
- Για να βρούμε τα συχνά στοιχεία:

Σε κάθε υποσύνολο, ελέγχουμε αν υπάρχει συχνό στοιχείο στις δυαδικές αναπαραστάσεις (sketches) που σχηματίζονται από τις ομάδες, και το αναφέρουμε.

Υποσύνολα

Λεπτομέρειες που πρέπει ακόμα να ξεκαθαριστούν:

- Πόσα υποσύνολα χρειάζονται, δηλαδή πόσο μεγάλο είναι το S; Πως τα επιλέγουμε;
- Πώς αποθηκεύουμε το κάθε υποσύνολο αποδοτικά;
- Τι εγγυήσεις έχουμε ότι αυτή η τεχνική δουλεύει;
- Τι γίνεται αν δεν υπάρχει μόνο ένα συχνό στοιχείο σε ένα υποσύνολο; Μπορούμε να ανιχνεύσουμε αυτή τη περίπτωση;

Επιλογή Υποσυνόλων

- Αν επιλέξουμε τα στοιχεία κάθε υποσυνόλου τυχαία τότε ο χώρος που θα χρειαστούμε είναι μεγάλος.
- Αντί αυτού χρησιμοποιούμε καθολικές συναρτήσεις κατακερματισμού (universal):
  - Οικογένεια συναρτήσεων: $F: \{0 \ldots U\text{-}1\} \rightarrow \{0 \ldots S\text{-}1\}$
  - Καθολική Ιδιότητα: $\forall\ x_1 \neq x_2$, και $\forall\ f$ επιλεγμένη ομοιόμορφα τυχαία από την οικογένεια F, ισχύει:

$$\mathbf{Pr}[f(x_1)=f(x_2)] = 1/S$$

- Η τοποθέτηση των στοιχείων στα υποσύνολα γίνεται ως εξής:
  - Αν $f(i)=j$ τότε συμπεριλαμβάνουμε το στοιχείο i στο υποσύνολο j (s_j).

Επιλογή Οικογένειας Συναρτήσεων

- Επιλέγουμε μία γνωστή καθολική οικογένεια η οποία επιπλέον έχει και την ιδιότητα ότι οι συναρτήσεις της είναι pairwise independent:

$$f_{a,b}(x) = ((ax + b) \bmod p) \bmod S$$

όπου p είναι ένας πρώτος αριθμός $>S$ αλλά με $p < 2S$ καθώς λόγω του ακόλουθου $\bmod S$ δεν χρειάζεται να είναι μεγαλύτερο.

- Τα a,b επιλέγονται από το διάστημα $[1,\ldots,p-1]$.
- Τι χρειάζεται να αποθηκεύσουμε τη συνάρτηση;
  - Αρκεί να αποθηκεύσουμε τις τιμές των παραμέτρων a,b που απαιτούν $\log p$ bits.

Ορισμός Συνόλων

- Χρησιμοποιούμε την προηγούμενη οικογένεια για να ορίσουμε τα τελικά S υποσύνολα του U.
- Έστω ότι ψάχνουμε για τα k πιο συχνά στοιχεία που εμφανίζονται με συχνότητα μεγαλύτερη από $n/(k+1)$.
- Θέτουμε: $S = 2k$ (θα δούμε σε λίγο γιατί). Δηλαδή **αρκούν $2k$ υποσύνολα.**
- Ορίζουμε το υποσύνολο s_j με κατάλληλα a_j και b_j.
- Το στοιχείο i τοποθετείται στο υποσύνολο s_j αν $f(i)=j$, δηλαδή αν:

$$((a_j i + b_j) \bmod p) \bmod S = j$$

Πιθανοτικά Αποτελέσματα

- Θέλουμε να επιλέξουμε κατάλληλα υποσύνολα ώστε να έχουμε **ακριβώς ένα συχνό στοιχείο** σε καθένα από αυτά.
- Μπορεί όμως να υπάρχουν περισσότερα από ένα συχνά στοιχεία σε ένα υποσύνολο που επιλέχθηκε.
- Ή μπορεί να μην υπάρχει κανένα συχνό στοιχείο σε ένα υποσύνολο που επιλέχθηκε.
- Αν μπορέσουμε να **φράξουμε** την πιθανότητα να συμβεί αυτό τότε ο αλγόριθμος θα τελειοποιηθεί.

Ανάλυση

- Έστω ***fr(i)*** η συχνότητα του στοιχείου ***i*** στην ροή, και ότι ένα συχνό στοιχείο ***x*** τοποθετείται σε ένα υποσύνολο (έχει συχνότητα μεγαλύτερη από ***n/(k+1)***).
- Η αναμενόμενη συχνότητα των υπολοίπων στοιχείων θα είναι:

Λόγω της καθολικής ιδιότητας

Αφαιρείται το όριο της συχνότητας του *x*.

$$E(fr) = \frac{1}{n}\sum_i fr(i)\frac{1}{S} = \sum_i \frac{fr(i)}{2kn} \le \frac{n - \frac{n}{k+1}}{2kn} = \frac{1}{2(k+1)}$$

Με $S=2k$ φράσσεται η πιθανότητα

- Έτσι, από την ανισότητα ***Markov*** η πιθανότητα ένα άλλο στοιχείο να έχει μεγαλύτερη συχνότητα από του ***x*** είναι:

$$\Pr\left(fr(other) > \frac{n}{k+1} \right) < \frac{\frac{n}{2(k+1)}}{\frac{n}{k+1}} = \frac{1}{2}$$

Επαναλήψεις

- Επειδή η πιθανότητα να έχουμε ένα ακόμα συχνό στοιχείο στο ίδιο υποσύνολο είναι μικρότερη του ½, αρκεί να επαναλάβουμε την διαδικασία (όπως με την ρίψη ενός νομίσματος) ώστε κάθε συχνό στοιχείο να μπει σε υποσύνολο που δεν υπάρχει άλλο συχνό στοιχείο.
- Μάλιστα αρκεί να επαναλαμβάνουμε την διαδικασία $\log(k/\delta)$ φορές (με διαφορετικές συναρτήσεις κατακερματισμού), ώστε να φράξουμε την πιθανότητα αποτυχίας στο δ, καθώς:
  - Η πιθανότητα αποτυχίας σε ένα υποσύνολο θα είναι:

$$(1/2)^{\log(k/\delta)}=\delta/k$$

  - Οπότε η συνολική πιθανότητα αποτυχίας για τα k συχνά στοιχεία θα είναι ίση με δ.

Ο Τελικός Αλγόριθμος

1. Επιλέγουμε $\log(k/\delta)$ ακέραιες τιμές a_j, b_j στο διάστημα $[0\ldots k-1]$ τυχαία με ίδια πιθανότητα.
2. Για κάθε στοιχείο i στη ροή:
   1. Για $j=1$ μέχρι $\log(k/\delta)$:
   - Αν $(((a_j \cdot i + b_j) \bmod p) \bmod 2k = x)$ τότε

     Πρόσθεσε (Αφαίρεσε) 1 από τις κατάλληλες ομάδες του x για μία άφιξη (αναχώρηση).
3. Για να βρούμε τα συχνά στοιχεία:
   - Για $j=1$ έως $\log(k/\delta)$, για $x=1$ έως $2k$, βρες αν υπάρχει συχνό στοιχείο και διάβασε την τιμή του από τις ομάδες του x.

Η Εύρεση των Συχνών Στοιχείων

Ψάχνουμε για στοιχεία με συχνότητα $> n/(k+1)$

- Άρα αν υπάρχει συχνό στοιχείο σε ένα υποσύνολο τότε η μέτρηση θα είναι $> n/(k+1)$
- Αν η μέτρηση είναι $< n/(k+1)$, τότε δεν υπάρχει συχνό στοιχείο στο αντίστοιχο υποσύνολο.
- Αν ψάχνουμε στις ομάδες ενός υποσυνόλου και βρούμε κάποια ομάδα M έτσι ώστε $\mu\acute{\epsilon}\tau\rho\eta\sigma\eta(M) > n/(k+1)$ και ταυτόχρονα $\mu\acute{\epsilon}\tau\rho\eta\sigma\eta(\upsilon\pi o\sigma\acute{\upsilon}\nu o\lambda o) - \mu\acute{\epsilon}\tau\rho\eta\sigma\eta(M) > n/(k+1)$ τότε υπάρχουν 2 ή περισσότερα συχνά στοιχεία στο υποσύνολο αυτό (έχουμε αποτυχία με πιθανότητα δ).

Απόδοση

Χρόνος Εκτέλεσης και Χώρος:

- Για την επεξεργασία ενός στοιχείου υπολογίζουμε $O(\log(k/\delta))$ συναρτήσεις κατακερματισμού.
- Απαιτείται $\log U$ χρόνος για επεξεργασία των ομάδων, άρα:

Συνολικός χρόνος: $O(\log k \cdot \log U)$

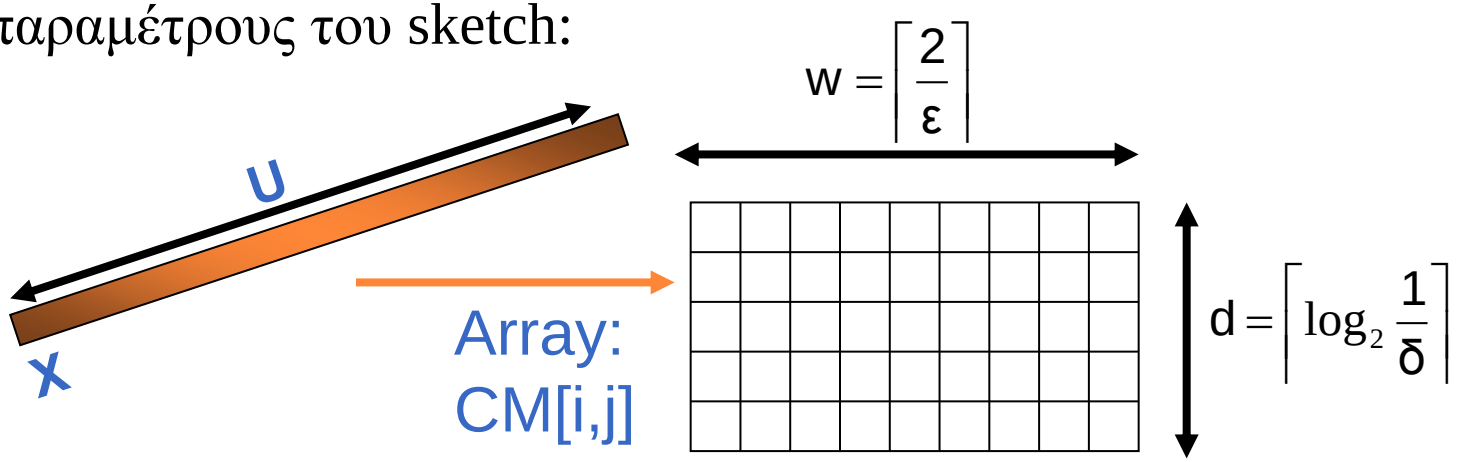
Χώρος: $O(\log k \cdot \log^2 U)$

Ένα μικρό θέμα: Ο αλγόριθμος μπορεί να μην δουλέψει όταν:

- Η συνολική συχνότητα των άλλων στοιχείων στο υποσύνολο είναι μεγαλύτερη από $n/(k+1)$
- Αυτό σπάνια συμβαίνει στην πράξη, και είδαμε ότι μπορεί να συμβεί με πιθανότητα δ (οπότε φράσσεται).

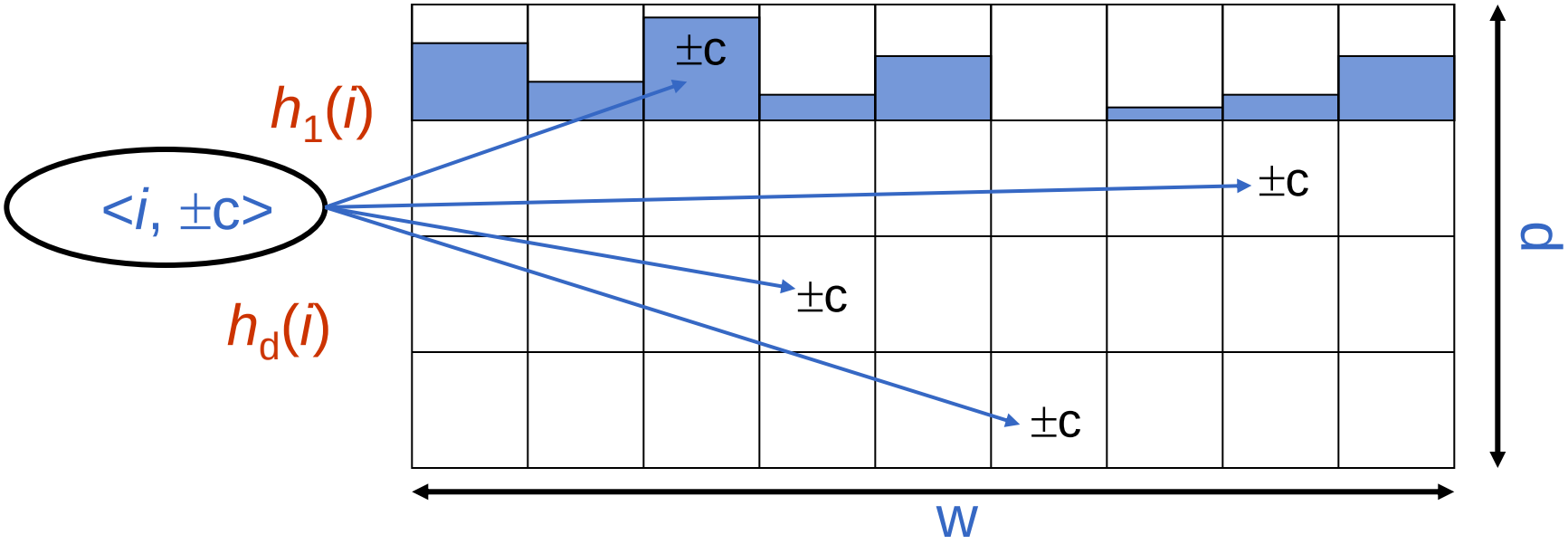
Count-Min Sketch [Cormode, Muthukrishnan'04]

- Η μέθοδος Count-Min Sketch που είδαμε στο 1ο μέρος μπορεί να εφαρμοστεί **απευθείας** στο πρόβλημα των k συχνών στοιχείων.
- Η ροή μοντελοποιείται ως ένα διάνυσμα X διάστασης U στο οποίο το στοιχείο $x[i]$ εκφράζει την συχνότητα του στοιχείου $i \in \{1 \ldots U\}$.
- Οι παράμετροι σφάλματος (ε) και αποτυχίας (δ) καθορίζουν τις παραμέτρους του sketch:



- Δημιουργείται μία συνοπτική εικόνα της ροής με τη μορφή ενός πίνακα **CM** διάστασης $w \times d$.
- Χρησιμοποιούνται d ανεξάρτητες συναρτήσεις hash $h()$ που απεικονίζουν ένα στοιχείο της ροής στο διάστημα $[1 \ldots w]$, άρα σε κάποιο κελί στην κάθε γραμμή (όχι απαραίτητα στο ίδιο).

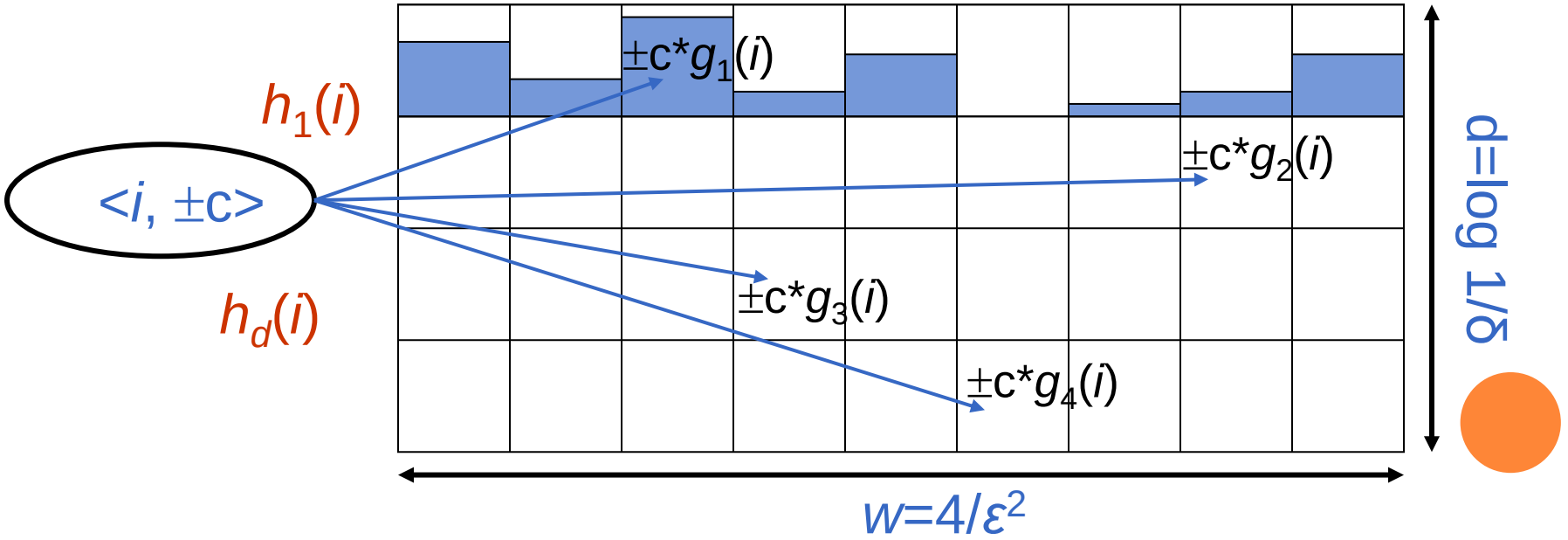
Απόδοση του Count-Min Sketch



- Η εκτίμηση κάθε συχνότητας ***x*[*i*]** (point query) γίνεται υπολογίζοντας την τιμή: **$\min_k$ { CM[k, $h_k(i)$] }.**
- Η εκτίμηση γίνεται με σφάλμα **$\pm\varepsilon \cdot n$**, με απαιτούμενο χώρο **$O(1/\varepsilon \log 1/\delta)$**, ενώ η πιθανότητα για μεγαλύτερο σφάλμα είναι μικρότερη από **$1-\delta$**.
- Ο χρόνος για μία ενημέρωση ή μία προσέγγιση είναι **$O(\log 1/\delta)$**.

Count Sketch [Charikar, Chen, Farach-Colton '02]

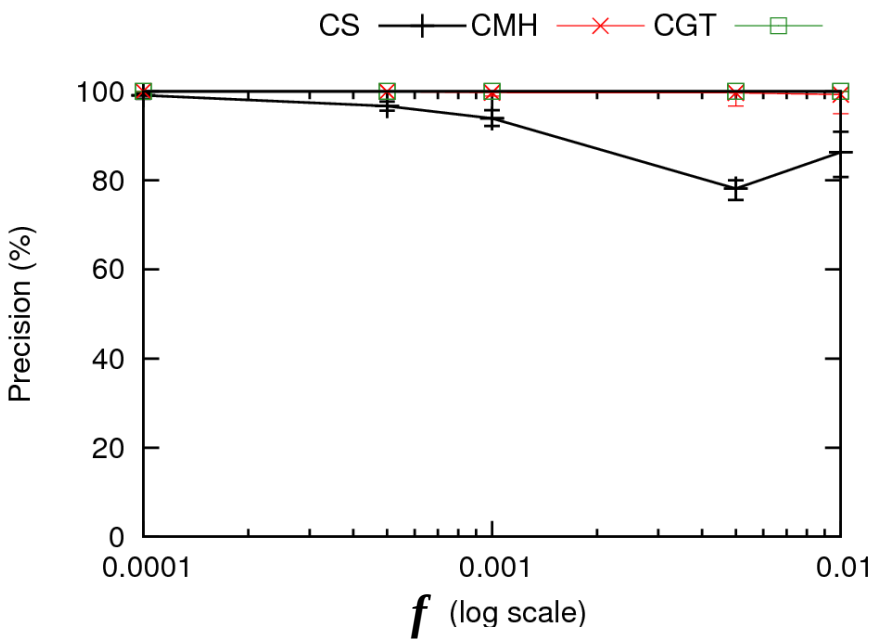
- Η βασική διαφορά της μεθόδου αυτής από την Count-Min Sketch είναι ότι δεν υπολογίζεται ελάχιστη τιμή στις εκτιμήσεις.
- Αυτό αποφεύγεται με τη χρήση $d=\log(1/\delta)$ έξτρα συναρτήσεων hash $g_1, g_2, \ldots, g_d: \{1 \ldots U\} \rightarrow \{+1,-1\}$.
- Το πλάτος w επιλέγεται να είναι $4/\varepsilon^2$.
- Σε μία ενημέρωση $\langle i, \pm c \rangle$ θέτουμε: $CM[k, h_k(i)] \mathrel{+}= c * g_k(i)$.

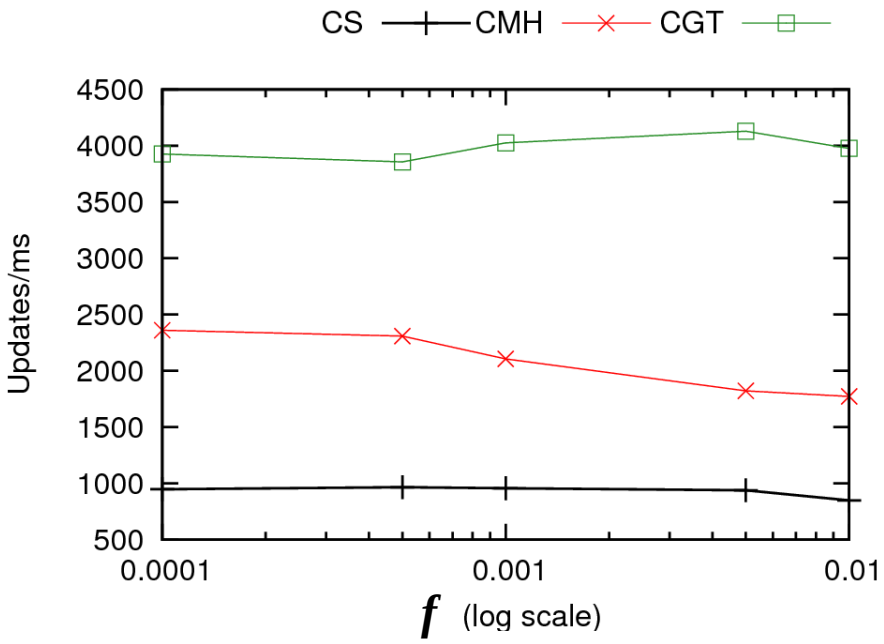


Ανάλυση του Count Sketch

- Η εκτίμηση κάθε συχνότητας $x[i]$ (point query) γίνεται υπολογίζοντας την **μέση τιμή** των: $CM[k, h_k(i)]*g_k(i)$.
- Τα όρια στο σφάλμα καθορίζονται αναλύοντας την διακύμανση των εκτιμήσεων και χρησιμοποιώντας την ανισότητα του Chebyshev. Αποδεικνύεται έτσι ότι και πάλι η εκτίμηση κάθε συχνότητας γίνεται με σφάλμα $\pm\varepsilon \cdot n$.
- Ο απαιτούμενος χώρος είναι: $O(1/\varepsilon \log 1/\delta)$.
- Η πιθανότητα για μεγαλύτερο σφάλμα είναι μικρότερη από $1-\delta$.
- Ο χρόνος για μία ενημέρωση ή μία προσέγγιση είναι και πάλι: $O(\log 1/\delta)$.

Υλοποιήσεις-Πειράματα





- **CGT** (Hierarchical Search with Group Testing), **CS** (Count Sketch), **CMH** (Count-Min Sketch).
- Παρατηρούμε ότι δεν είναι σαφές ποια μέθοδος είναι καλύτερη. Εξαρτάται από τις τιμές των **παραμέτρων** και των **δεδομένων**.
- Παρατηρούμε τώρα ότι η **ταχύτητα** ενημερώσεων είναι **10 φορές χειρότερη** από τους αλγορίθμους των μετρητών, ενώ **απαιτείται δεκαπλάσιος χώρος**.
- Αυτό μάλλον είναι και το κόστος που πληρώνουμε για να υποστηρίξουμε το μοντέλο του μύλου (αφίξεις και αναχωρήσεις).

Συμπεράσματα

- Το πρόβλημα των συχνών στοιχείων είναι ένα από τα σπουδαιότερα προβλήματα με τεράστιες εφαρμογές στις ροές.
- Μελετήθηκε πολύ και προτάθηκαν αρκετοί αλγόριθμοι για τη λύση του.
- **Οι αλγόριθμοι των μετρητών** έχουν μεγαλύτερη ταχύτητα ενημερώσεων απαιτώντας λιγότερο χώρο, αλλά δεν υποστηρίζουν το μοντέλο του μύλου (αφίξεις και αναχωρήσεις στοιχείων).
- **Οι αλγόριθμοι που χρησιμοποιούν sketches** έχουν μικρότερη ταχύτητα, απαιτούν περισσότερο χώρο αλλά υποστηρίζουν και το μοντέλο του μύλου.
- **Επεκτάσεις** του προβλήματος έχουν επίσης μελετηθεί όπως:
  - Η εύρεση συχνών υποσυνόλων (όχι μεμονομένων στοιχείων)
  - Η εύρεση των μεγαλύτερων αλλαγών μεταξύ διαδοχικών ροών της ίδιας πηγής
  - κλπ.


ΤΕΛΟΣ