Διάλεξη 28: Τεχνικές Κατακερματισμού

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

Διαχείριση Συγκρούσεων με Ανοικτή Διεύθυνση

a) Linear Probing, b) Quadratic Probing c) Double Hashing

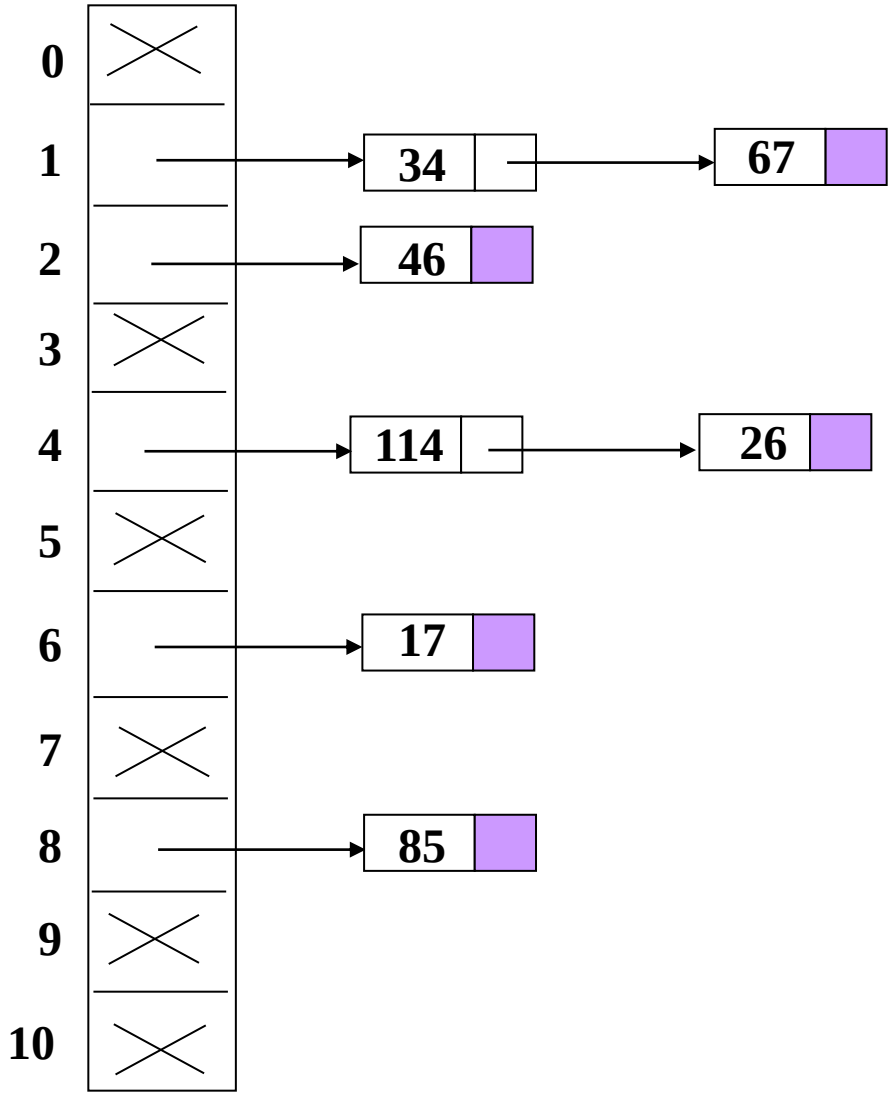
Διατεταγμένος Κατακερματισμός (Ordered Hashing)

Επανακατακερματισμός (Rehashing)

Εφαρμογές Κατακερματισμού

Διδάσκων: Δημήτρης Ζεϊναλιπούρ

Επανάληψη Διαχείρισης Συγκρούσεων με Αλυσίδωση



hsize = 11

Διαχείριση Συγκρούσεων με ανοικτή διεύθυνση

- Η αντιμετώπιση **συγκρούσεων με αλυσίδωση** περιλαμβάνει επεξεργασία δεικτών και δυναμική χορήγηση μνήμης. Επίσης δημιουργούνται overflow chains, τα οποία θα κάνουν τις αναζητήσεις ακριβότερες στην συνέχεια
- Η στρατηγική **ανοικτής διεύθυνσης** επιτυγχάνει την αντιμετώπιση συγκρούσεων χωρίς τη χρήση δεικτών. Τα στοιχεία αποθηκεύονται κατ' ευθείαν στον πίνακα κατακερματισμού ως εξής:
- Για να **εισαγάγουμε** το κλειδί k στον πίνακα:
  1. υπολογίζουμε την τιμή i=h(k), και
  2. αν η θέση H[i] είναι κενή τότε αποθηκεύουμε εκεί το k,
  3. διαφορετικά, δοκιμάζουμε τις θέσεις f(i), f(f(i)),…, για κάποια συνάρτηση f, μέχρις ότου βρεθεί κάποια κενή θέση όπου και τοποθετούμε το k.
- Για την **αναζήτηση** κάποιου κλειδιού k μέσα στον πίνακα:
  1. υπολογίζουμε την τιμή i=h(k), και
  2. κάνουμε διερεύνηση της ακολουθίας, i, f(i), f(f(i)),…, μέχρι, είτε να βρούμε το κλειδί, είτε να βρούμε κενή θέση, ή να περάσουμε από όλες τις θέσεις του πίνακα.

Γραμμική Αναζήτηση Ανοικτής Διεύθυνσης **(Linear Probing)**

- Η αρχική συνάρτηση κατακερματισμού είναι

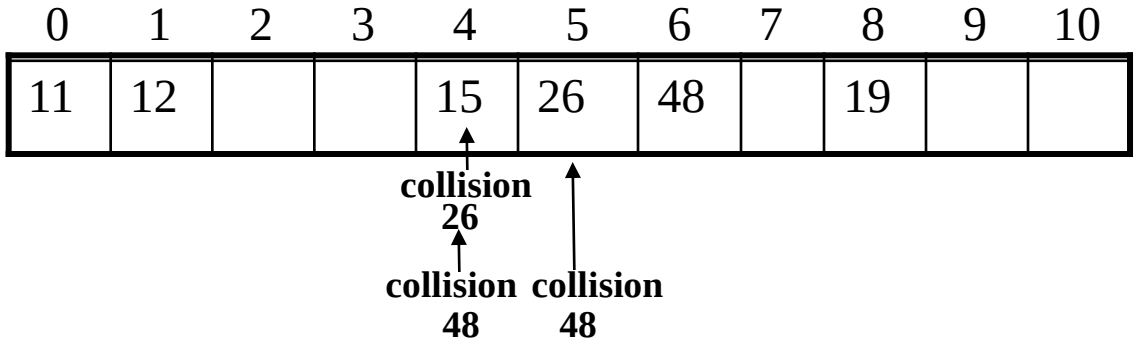
  f(x)'= x mod hsize

- Όταν υπάρξει σύγκρουση (collision) δοκιμάζουμε αναδρομικά την επόμενη συνάρτηση μέχρι να βρεθεί κενή θέση:

  f(x)= (f(x)'+i) mod hsize (i=1,2,3,…)

- Δηλαδή η αναζήτηση κενής θέσης γίνεται σειριακά, και η αναζήτηση ονομάζεται **γραμμική (linear probing).**

- Παράδειγμα: **hsize = 11**, εισαγωγή 11, 12, 15, 19, 26, 48.



Σχόλια για το Linear Probing

Εισαγωγή

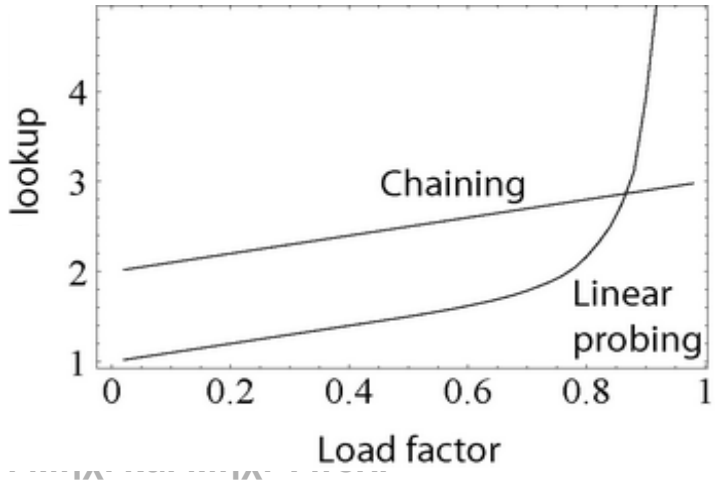
- Εφόσον ο πίνακας κατακερματισμού δεν είναι γεμάτος, είναι πάντα δυνατό να εισάγουμε κάποιο καινούριο κλειδί. Αν γεμίσει θα κάνουμε rehash τον πίνακα (θα το δούμε στην συνέχεια)
- Αν οι γεμάτες θέσεις του πίνακα είναι **μαζεμένες (clustered)** τότε ακόμα και αν ο πίνακας είναι σχετικά άδειος, πιθανόν να χρειαστούν πολλές δοκιμές για εύρεση κενής θέσης (κατά την εκτέλεση διαδικασίας insert), ή για εύρεση στοιχείου.

0	1	2	3	4	5	6	7	8	9	10
11	12			15	26	48		19		

cluster

Αναζήτηση

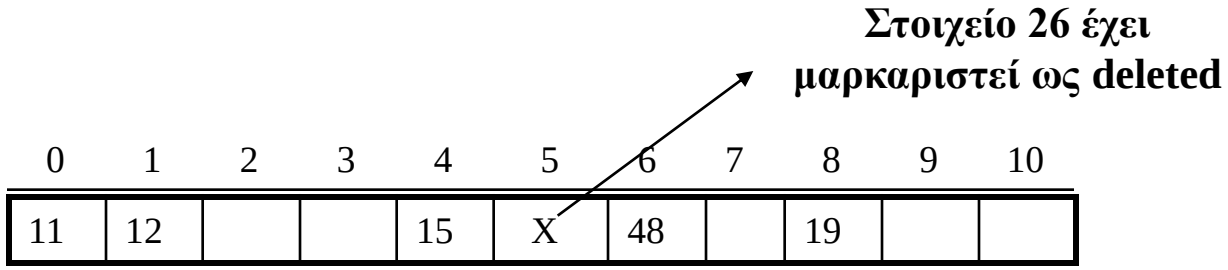
- Η αναζήτηση γίνεται όπως και την εισαγωγή (σταματάμε όταν βρούμε κενή θέση).
- Μπορεί να αποδειχθεί ότι για ένα πίνακα μισογεμάτο (δηλαδή $\lambda = 0.5$) και μια ομοιόμορφη κατανομή τότε:
  1. **Ανεπιτυχή Διερεύνηση:** Ο αριθμός βημάτων είναι ~2.5
  2. **Επιτυχή Διερεύνηση:** Ο αριθμός βημάτων είναι ~1.5.
- Αν το λ πλησιάζει το 1, τότε οι πιο πάνω αναμενόμενοι αριθμοί βημάτων αυξάνονται εκθετικά.



Σχόλια για το Linear Probing

Εξαγωγή

- Πρέπει να είμαστε προσεκτικοί με τις εξαγωγές στοιχείων
  1. μια θέση από την οποία έχει αφαιρεθεί στοιχείο δεν μπορεί να θεωρηθεί ως άδεια (γιατί;) *διότι στην find δεν θα ξέρουμε που να σταματήσουμε*
  2. έτσι μαρκάρουμε τη θέση ως deleted, και
  3. κατά τη διαδικασία find, αγνοούμε θέσεις deleted, και προχωρούμε μέχρις ότου είτε να βρούμε το κλειδί που ψάχνουμε, είτε να βρούμε (πραγματικά) μια άδεια θέση είτε να σαρώσουμε ολόκληρο τον πίνακα).



Δευτεροβάθμια αναζήτηση Ανοικτής Διεύθυνσης **(Quadratic Probing)**

- Η αρχική συνάρτηση κατακερματισμού είναι και πάλι:

  f(x)'= x mod hsize

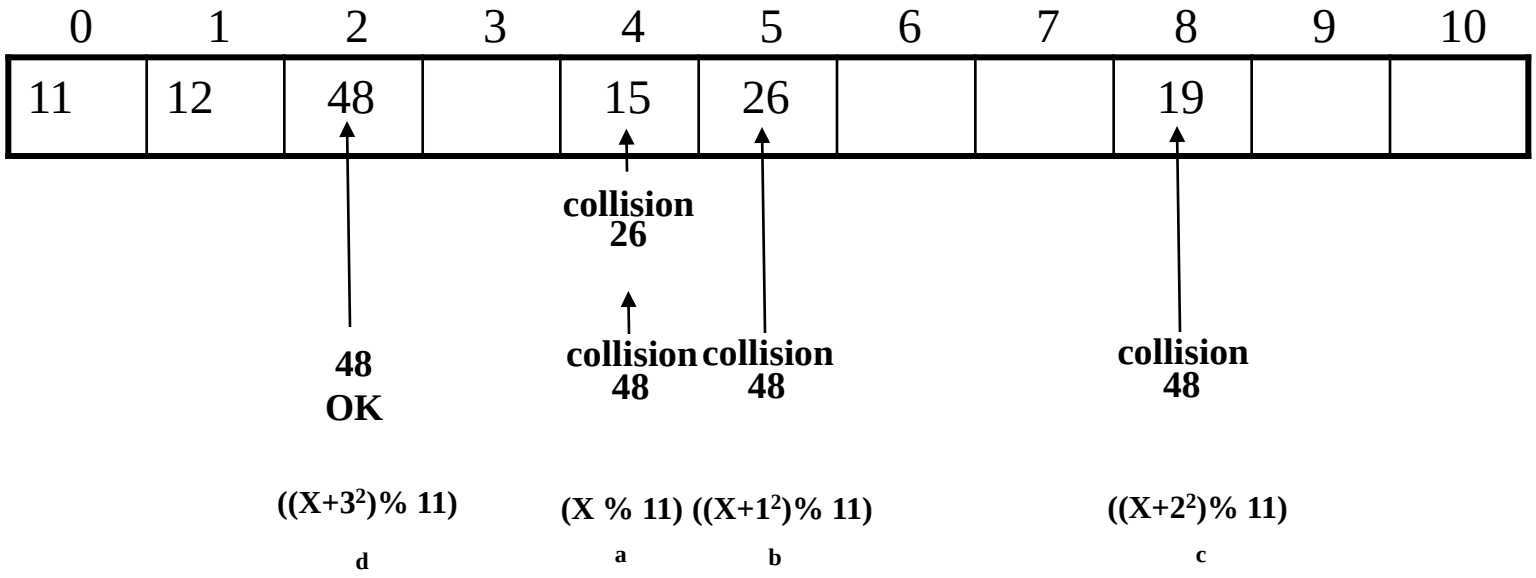
- Όταν υπάρξει σύγκρουση (**collision**) δοκιμάζουμε αναδρομικά την επόμενη συνάρτηση μέχρι να βρεθεί κενή θέση:

  f(x)= (f(x)'+ i^2) mod hsize (i=1,2,3,…)

- **Στόχος:** αποφυγή των μαζεμένων κλειδιών (clusters)

Παράδειγμα Quadratic Probing

Παράδειγμα: **hsize = 11**, εισαγωγή **11, 12, 15, 19, 26, 48**



• Στο linear probing ήταν εγγυημένη η εισαγωγή (εφόσον ο πίνακας δεν έχει γεμίσει)

• Και εδώ μπορεί να αποδειχθεί ότι :

Θεώρημα: Αν το μέγεθος hsize είναι **πρώτος (prime) αριθμός (>3)** τότε οποιοδήποτε καινούριο κλειδί μπορεί να εισαχθεί στον πίνακα εφόσον ο πίνακας έχει **λ≤ 0.5**.

Διπλός Κατακερματισμός Ανοικτής Διεύθυνσης (Double Hashing)

- Ο τελευταίος τρόπος αποφυγής συγκρούσεων χρησιμοποιεί δυο συναρτήσεις κατακερματισμού.
- Δηλαδή σε περίπτωση αρχικής αποτυχίας εισαγωγής / εύρεσης στοιχείου οι θέσεις που επιλέγουμε για να διερευνήσουμε στη συνέχεια (probe sequence) είναι **ανεξάρτητες** από την πρώτη.

  $\mathbf{f(x,0) = h_1(x)}$ ***// η αρχική συνάρτηση κατακερματισμού***

- Αυτό επιτυγχάνεται με τη χρήση μιας δεύτερης συνάρτησης κατακερματισμού, h_2, ως εξής:

  $\mathbf{f(x,n) = (\ h_1(x) \ + \ n \cdot h_2(x) \ ) \bmod hsize}$

- Στην πράξη δουλεύει αποδοτικά ωστόσο είναι πιο ακριβό να υπολογίζουμε δυο συναρτήσεις κάθε φορά

Επανακατακερματισμός (Rehashing)

- Αν ο hash πίνακας αρχίσει να γεμίζει, παρατηρείται μεγάλος **αριθμός συγκρούσεων (collisions)** με αποτέλεσμα τη μειωμένη επίδοση.
- Η μειωμένη επίδοση παρατηρείται και σε πράξεις εισαγωγής αλλά στις πράξεις αναζήτησης.
- Σε τέτοιες περιπτώσεις, όταν η τιμή λ υπερβεί κάποιο όριο, πολλές υλοποιήσεις hash-πινάκων, αυτόματα εφαρμόζουν **επανά-κατακερματισμό.**
- Αυτό το όριο σε τυπικές υλοποιήσεις είναι συνήθως λ=0.7 (π.χ. Java)
- Επανακατακερματισμός (rehashing)
  - Δημιούργησε ένα καινούριο πίνακα μεγαλύτερου (διπλάσιου) μεγέθους.
  - Εισήγαγε όλα τα στοιχεία του παλιού πίνακα στον καινούριο.
  - Επέστρεψε τη μνήμη του παλιού πίνακα.
- Ακριβή διαδικασία, αλλά καλείται σπάνια.
- Σε συστήματα **πραγματικού χρόνου (real-time systems)** το rehashing μπορεί να πάρει περισσότερο χρόνο από ότι υπάρχει!
- Εκεί το rehashing γίνεται σταδιακά (δηλαδή κρατούμε το παλιό και νέο HashTable), και σε κάθε εισαγωγή μετακινούμε K στοιχεία στο νέο table μέχρι να μετακινηθούν όλα τα στοιχεία (οπόταν διαγράφεται το παλιό table)

Επανακατακερματισμός (Rehashing)

- Σε βάσεις δεδομένων (databases), ο όγκος των δεδομένων είναι πολύ μεγάλος και τα δεδομένα είναι αποθηκευμένα στον δίσκο.
- Άρα το re-hashing, θα έπαιρνε παρά πολύ χρόνο μέχρι να ολοκληρωθεί.
- Για αυτό χρησιμοποιούνται dynamic hashing techniques (π.χ. Linear and extendible hashing)
- Σε αυτές τις τεχνικές, μόνο ένα πολύ μικρό ποσοστό δεδομένων χρειάζεται να γίνει rehashed.

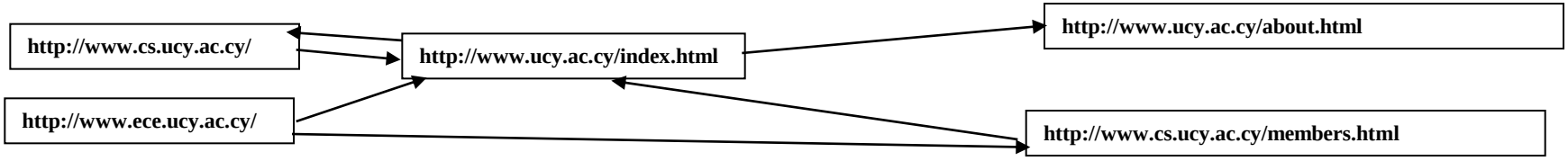
Μερικές Εφαρμογές του Κατακερματισμού

- Εφαρμογές Κατακερματισμού (Μνήμης & Μαγνητικού Δίσκου)
  - **Unique:** Έχετε ένα αρχείο από strings και θέλετε με ένα πέρασμα (χρόνος O(n)), να βρείτε όλες τις μοναδικές λέξεις σε αυτό το αρχείο.
  - **Ευρετήρια Λέξεων σε Μηχανές Αναζήτησης**: Ψάχνουμε σε μια μηχανή αναζήτησης την λέξη "car + rental". Η μηχανή μας επιστρέφει την τομή των αποτελεσμάτων (συνόλων) car και rental σε χρόνο O(1).
  - **Find Function:** Σε εργαλεία επεξεργασίας κειμένου (text editors, word, κτλ) το πρόγραμμα προσφέρει την δυνατότητα εύρεσης λέξεων. Πολλές φορές η πρώτη εκτέλεση του find είναι αργή (πχ. Microsoft Help) διότι χρειάζεται χρόνος για την δημιουργία του hash table).
  - **Σε μεταγλωττιστές,** πίνακες κατακερματισμού που ονομάζονται Symbol Tables αποθηκεύουν πληροφορίες για όλες τις μεταβλητές.
  - **Διερεύνηση γράφων** που δεν είναι εξ'αρχής γνωστοί.

Εφαρμογή Κατακερματισμού: Web Crawling

Περιγραφή

- Μπορούμε να θεωρήσουμε το WWW ως ένα γράφο του οποίου
  - κόμβοι είναι οι διάφορες σελίδες, και
  - ακμή μεταξύ δύο κόμβων υπάρχει αν η μια από τις δύο σελίδες περιέχει link στην άλλη.



- Ένας crawler είναι ένα πρόγραμμα το οποίο παίρνει σαν τιμή εισόδου ένα η περισσότερα URLs (π.χ. www.cs.ucy.ac.cy) και στην συνέχεια ανακτά (downloads) αναδρομικά αυτά τα URLs στον τοπικό του δίσκο.
- Η ανάκτηση τερματίζεται όταν ικανοποιηθεί κάποια συνθήκη (π.χ. Ν σελίδες έχουν ανακτηθεί, Φτάσαμε σε βάθος β, κτλ)
- Εφόσον το WWW είναι ένας γράφος μπορούμε να χρησιμοποιήσουμε αλγόριθμους διάσχισης γραφών
  - *Κατά-πλάτος διερεύνηση (BFS)*
  - *Κατά-βάθος διερεύνηση (DFS)*

Πρόβλημα:

- Ο Crawler, πρέπει με κάποιο τρόπο να θυμάται από ποιες ιστοσελίδες πέρασε.
- Έτσι δεν χρειάζεται να ξαναπεράσει από κάποια ιστοσελίδα που ήδη πέρασε

Λύση:

Χρήση HashTable που κρατά από ποιες ιστοσελίδες περάσαμε

Ψευδοκώδικας λύσης

```
HashTable H[hsize]; // Ο Πίνακας Κατακερματισμού
DFS(www.cs.ucy.ac.cy);

void DFS(URL url){

   // ανάκτηση της σελίδας στον τοπικό δίσκο
   download(url);

   // εισαγωγή του url στον πίνακα κατακερματισμού
   H = put(url);

   // εξαγωγή όλων των Links
   URL[] = extractURLs(page);

   for i=0 to |URL| {

         // επίσκεψη σε όλα τα ανεξερεύνητα Links
         if (!contains(H, hashcode(URL[i])))
               DFS(URL[i]);
   }
}
```