Τεχνικές Κατακερματισμού

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

Τεχνικές Κατακερματισμού

Διαχείριση Συγκρούσεων με Αλυσίδωση

Διαχείριση Συγκρούσεων με Ανοικτή Διεύθυνση

Διπλός Κατακερματισμός, Διατεταγμένος Κατακερματισμός

Επανακατακερματισμός

Τεχνικές Κατακερματισμού – Hashing

- Έχουμε μελετήσει τη χρήση ισοζυγισμένων δυαδικών δένδρων για αποθήκευση δεδομένων. Οι διαδικασίες find και insert έχουν χρόνο εκτέλεσης της τάξης O( ).
- Έστω ότι θέλουμε να κατασκευάζουμε και να διατηρούμε σύνολα, στοιχεία των οποίων προέρχονται από ένα 'σύμπαν' (universe) U σταθερού μεγέθους n: $U = \{u_1, \ldots, u_n\}$
- Υπάρχει πιο αποδοτική μέθοδος από τη χρήση δυαδικών δένδρων;
- Ναι, δεδομένου ότι υπάρχει μια απλή διαδικασία η οποία για κάθε $u \in U$, υπολογίζει την τιμή i για την οποία $u = u_i$. π.χ. χρησιμοποιώντας την ιδέα του BucketSort.
- Μεταξύ των απλούστερων τρόπων να απεικονίζουμε ένα σύνολο $S \subseteq U$ είναι αυτός του διανύσματος δυφίων, ***bit-vector***.

Bit vectors

- Ένα bit-vector είναι μονοδιάστατος πίνακας με n bits, Bits[1..n], τέτοιος ώστε

  Bits[i] = 1, αν u $\in$ S

  Bits[i] = 0, αν u $\notin$ S

- Για παράδειγμα αν U = {1,…,9} τότε το σύνολο S = {1 , 3 , 7} αναπαρίσταται ως το bit-vector:

  Bits = [1,0,1,0,0,0,1,0,0]

- Ο χρόνος εύρεσης και εισαγωγής/διαγραφής στοιχείου με αυτή την απεικόνιση είναι O(1).

- Πρόβλημα της στρατηγικής αυτής είναι:

  σπατάλη μνήμης αν το N είναι κατά πολύ μεγαλύτερο από τον αριθμό των στοιχείων που πρόκειται να βρίσκονται ανά πάσα στιγμή στον πίνακα.

- Λύση: ο **κατακερματισμός** που είναι μια οικογένεια μεθόδων που αντιστοιχεί ένα κλειδί σε μία θέση ενός πίνακα (key-to-address transformation).

Κατακερματισμός – Κεντρική Ιδέα

- ***Πίνακας κατακερματισμού*** (hash table) είναι μια δομή δεδομένων που υποστηρίζει τις διαδικασίες insert και find σε (σχεδόν) σταθερό χρόνο.

- Ένας πίνακας κατακερματισμού χαρακτηρίζεται από
  1. το μέγεθος του, **hsize**, και
  2. κάποια ***συνάρτηση κατακερματισμού*** **h** η οποία αντιστοιχεί κλειδιά στο σύνολο των ακεραίων [0,…, hsize –1].

- Τα δεδομένα αποθηκεύονται στον πίνακα H[0,…, hsize –1]:

  το κλειδί k αποθηκεύεται στον H στη θέση H[h(k)].

- Αποδοτικές διαδικασίες insert και find.

- Δημιουργούνται δύο σημαντικά ερωτήματα:
  1. ποια είναι καλή επιλογή για τη συνάρτηση h;
  2. τι θα πρέπει να γίνεται αν $h(k_1) = h(k_2)$ για δύο κλειδιά k_1, k_2, με $k_1 \neq k_2$;

  (αυτό ονομάζεται ***σύγκρουση*** και είναι πολύ πιθανό να συμβεί.)

Παράδειγμα

- Έστω ότι τα δεδομένα μας είναι ζεύγη (όνομα, αρ. ταυτότητας) και hsize = 11. Χρησιμοποιούμε τον α.τ. ως κλειδί και θέτουμε $h((\alpha, \beta)) = \beta \bmod 11$.

H	data
0	
1	(Λουκία, 813352)
2	
3	(Ελένη, 782312)
4	
5	
6	
7	
8	
9	
10	(Χρίστος, 754621)

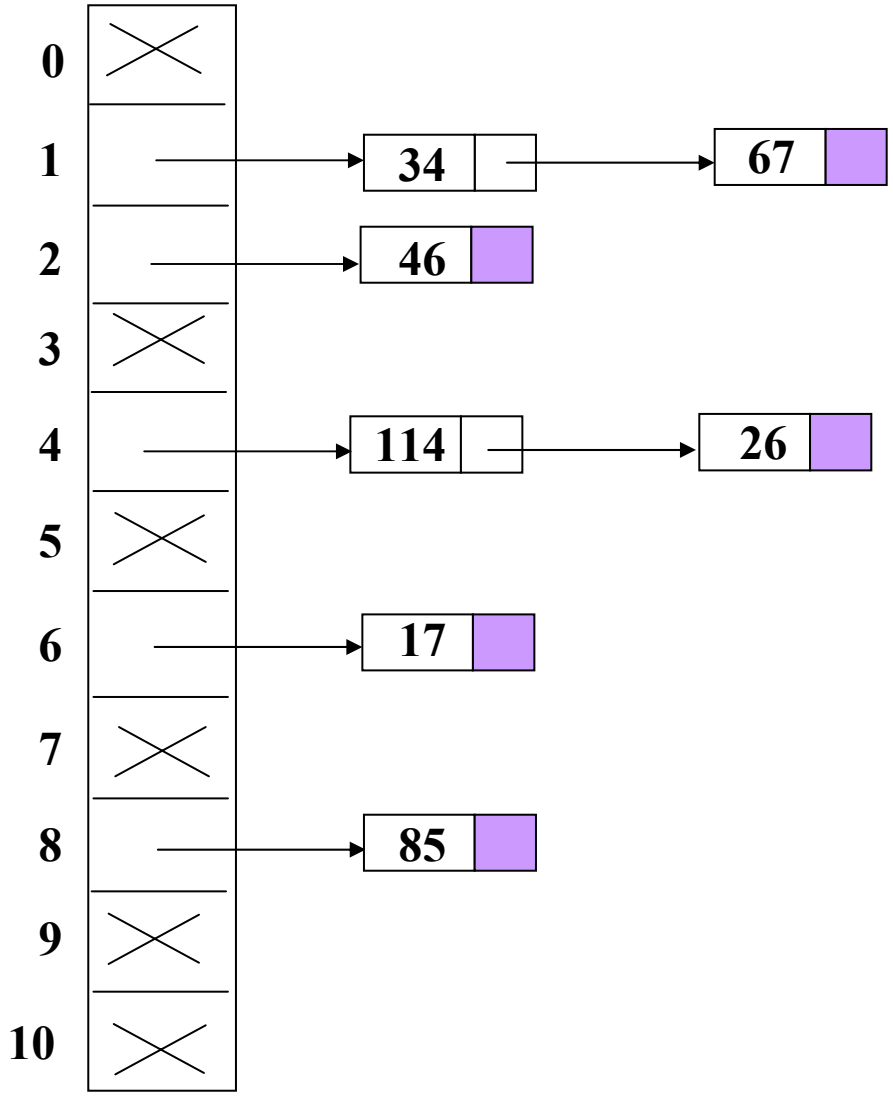
Επιλογή Συνάρτησης Κατακερματισμού

- Ιδιότητες μιας καλής συνάρτησης κατακερματισμού:
  1. Θα πρέπει να χρησιμοποιεί ολόκληρο τον πίνακα [0… hsize –1].
  2. Θα πρέπει να ΄σκορπίζει΄ ομοιόμορφα τα κλειδιά στον πίνακα H.
  3. Θα πρέπει να υπολογίζεται εύκολα.
- Συνήθως το μέγεθος του πίνακα hsize είναι *πρώτος* αριθμός. Η συνάρτηση h αρχικά αντιστοιχίζει το κλειδί σε κάποιο ακέραιο αριθμό a και στη συνέχεια παίρνει την τιμή a mod hsize.
- Έστω ότι το κλειδί είναι αλυσίδα από 8 χαρακτήρες s[0…7]. Δύο παραδείγματα συναρτήσεων κατακερματισμού είναι:
  1. το άθροισμα των κωδικών ASCII των χαρακτήρων.
  2. $s[0] + s[1]*128 + s[2]*128^2 + \ldots + s[7]*128^7$
- Ερώτημα: Πως γίνεται η διαχείριση συγκρουόμενων κλειδιών;
- Οι τεχνικές λύσεις διακρίνονται σε 2 κατηγορίες: μέθοδοι με ***αλυσίδωση*** και μέθοδοι ***ανοικτής διεύθυνσης.***

Διαχείριση συγκρούσεων με αλυσίδωση

- Αφού περισσότερα από ένα κλειδιά μπορούν να πάρουν την ίδια τιμή από τη συνάρτηση κατακερματισμού, μπορούμε να θεωρήσουμε ότι κάθε θέση του πίνακα 'δείχνει' σε μια ευθύγραμμη απλά συνδεδεμένη λίστα.
- Για κάθε i, στη θέση H[i] του πίνακα βρίσκουμε λίστα που περιέχει όλα τα κλειδιά που απεικονίζονται από τη συνάρτηση h στη θέση αυτή.
- Για να βρούμε κάποιο κλειδί k, πρέπει να ψάξουμε στη λίστα που δείχνεται στη θέση H[h(k)].
- Εισαγωγές και εξαγωγές στοιχείων μπορούν να γίνουν εύκολα με βάση τις διαδικασίες συνδεδεμένων λιστών.

Παράδειγμα



hsize = 11

Ανάλυση της Τεχνικής Αλυσίδωσης

- Εισαγωγή στοιχείου απαιτεί χρόνο O(1).
- Εύρεση/διαγραφή κλειδιού k συνεπάγεται τη διέλευση της λίστας H[h(k)].
- Για να αναλύσουμε τον χρόνο εκτέλεσης των πιο πάνω διαδικασιών ορίζουμε τον συντελεστή φορτίου (load factor) λ του πίνακα H ως τον λόγο

  $$\lambda = n/hsize$$

  όπου n είναι ο αριθμός των στοιχείων που αποθηκεύει ο πίνακας.
- Δηλαδή, κατά μέσο όρο, κάθε λίστα του πίνακα έχει μήκος λ. Επομένως ο αναμενόμενος χρόνος χειρίστης περίπτωσης εύρεσης στοιχείου είναι 1+λ.
- Ιδανικά θα θέλαμε ο λόγος λ να έχει σταθερή τιμή (συνήθως κοντά στο 1), ώστε οι διαδικασίες να εκτελούνται σε σταθερό χρόνο. Άρα πρέπει να διαλέγουμε το μέγεθος του πίνακα να είναι ανάλογο του αριθμού των στοιχείων, n.

Διαχείριση Συγκρούσεων με ανοικτή διεύθυνση

- Η αντιμετώπιση συγκρούσεων με αλυσίδωση περιλαμβάνει επεξεργασία δεικτών και δυναμική χορήγηση μνήμης.
- 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, είτε να περάσουμε από όλες τις θέσεις του πίνακα.

Γραμμική Αναζήτηση ανοικτής διεύθυνσης

- Αν η συνάρτηση f είναι η

  $$f(x)= (x+1) \bmod hsize$$

  δηλαδή η αναζήτηση κενής θέσης γίνεται σειριακά, τότε η αναζήτηση ονομάζεται γραμμική (linear probing).

- Παράδειγμα: hsize = 11, εισαγωγή 23, 15, 96, 114, 49.

0	1	2	3	4	5	6	7	8	9	10

Ανάλυση του Linear Probing

- Εφόσον ο πίνακας κατακερματισμού δεν είναι γεμάτος, είναι πάντα δυνατό να εισάγουμε κάποιο καινούριο κλειδί.
- 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.
- Αν $\lambda = 0.5$ (ο πίνακας είναι μισογεμάτος) και τα στοιχεία είναι σκορπισμένα ομοιόμορφα στον πίνακα, τότε
  1. ο αναμενόμενος αριθμός βημάτων για ανεπιτυχή διερεύνηση είναι 2.5, και
  2. ο αναμενόμενος αριθμός βημάτων για επιτυχή διερεύνηση είναι 1.5.
- Αν το λ πλησιάζει το 1, τότε οι πιο πάνω αναμενόμενοι αριθμοί βημάτων αυξάνονται δραστικά.

Δευτεροβάθμια αναζήτηση ανοικτής διεύθυνσης

- Quadratic Probing.
- Στόχος: αποφυγή των μαζεμένων κλειδιών (clusters).
- Χρησιμοποιούμε τη συνάρτηση

  $$\mathbf{f(k,i) = (h(k) + c_1 \cdot i + c_2 \cdot i^2) \bmod hsize}$$

- Για την εισαγωγή κλειδιού k, πρώτα δοκιμάζουμε τη θέση
  - $x = h(k) \bmod hsize$,

  και αν είναι γεμάτη προχωρούμε στις θέσεις
  - $(x + c_1 + c_2) \bmod hsize$, $(x + c_1 \cdot 2 + c_2 \cdot 4) \bmod hsize$, $(x + c_1 \cdot 3 + c_2 \cdot 9) \bmod hsize$, και ούτω καθεξής.
- Μπορούμε να είμαστε σίγουροι ότι εισαγωγή στοιχείου είναι δυνατή όταν ο πίνακας δεν είναι γεμάτος;
- Θεώρημα: Αν το μέγεθος hsize είναι πρώτος αριθμός (>3) τότε οποιοδήποτε καινούριο κλειδί μπορεί να εισαχθεί στον πίνακα εφόσον ο πίνακας έχει $\lambda \leq 0.5$.

Παράδειγμα

- Παράδειγμα: hsize = 11, εισαγωγή 23, 15, 96, 114, 49, με c_1=0, c_2=1.

0	1	2	3	4	5	6	7	8	9	10

Άλλες Τεχνικές Κατακερματισμού

1. **Αναζήτηση με διπλό κατακερματισμό ανοικτής διεύθυνσης (double hashing)**

- Σε περίπτωση αρχικής αποτυχίας εισαγωγής/εύρεσης στοιχείου οι θέσεις που επιλέγουμε για να διερευνήσουμε στη συνέχεια (probe sequence) είναι ανεξάρτητες από την πρώτη. Αυτό επιτυγχάνεται με τη χρήση μιας δεύτερης συνάρτησης κατακερματισμού, h_2, ως εξής:

$$f(x,0) = h_1(x)$$
$$f(x,n) = (\ h_1(x) \ + \ n \cdot h_2(x) \ ) \bmod hsize$$

Άλλες Τεχνικές Κατακερματισμού

2. Αναζήτηση με διατεταγμένο κατακερματισμό ανοικτής διεύθυνσης (ordered hashing)

- Η μέθοδος αυτή χρησιμοποιείται σε συνδυασμό με οποιαδήποτε από τις άλλες τεχνικές με στόχο την ελάττωση του χρόνου εκτέλεσης της διερεύνησης. Η βασική ιδέα είναι να εξασφαλίζεται ότι τα κλειδιά που συναντούμε κατά τη διερεύνηση μιας probing sequence είναι σε αύξουσα σειρά. Έτσι, αν συναντήσουμε κλειδί που είναι μεγαλύτερο από αυτό που ψάχνουμε, τότε συμπεραίνουμε πως δεν υπάρχει στον πίνακα.

Μέθοδος υλοποίησης

- Μέθοδος υλοποίησης: κατά την εισαγωγή κλειδιού k σε ένα πίνακα, αν βρούμε κλειδί k΄> k, τότε εισάγουμε το k στη θέση του k΄ και αναλαμβάνουμε να εισάγουμε το k΄ βάσει της ακολουθίας θέσεων του k΄.

- Σαν αποτέλεσμα έχουμε βελτιωμένη διαδικασία ανεπιτυχούς αναζήτησης.

Επανακατακερματισμός (rehashing)

- Αν ο hash πίνακας αρχίσει να γεμίζει, παρατηρείται μεγάλος αριθμός συγκρούσεων (thrashing) με αποτέλεσμα τη μειωμένη επίδοση.
- Σε τέτοιες περιπτώσεις, όταν η τιμή λ υπερβεί κάποιο όριο, πολλές υλοποιήσεις hash-πινάκων, αυτόματα εφαρμόζουν επανακατακερματισμό.
- Επανακατακερματισμός (rehashing)
  - Δημιούργησε ένα καινούριο πίνακα μεγαλύτερου μεγέθους.
  - Εισήγαγε όλα τα στοιχεία του παλιού πίνακα στον καινούριο.
  - Επέστρεψε τη μνήμη του παλιού πίνακα.
- Ακριβή διαδικασία, αλλά καλείται σπάνια.

Μερικά Σχόλια

- Στην περίπτωση του κατακερματισμού ανοικτής διεύθυνσης πρέπει να είμαστε προσεκτικοί με τις εξαγωγές στοιχείων
  1. μια θέση από την οποία έχει αφαιρεθεί στοιχείο δεν μπορεί να θεωρηθεί ως άδεια (γιατί;)
  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).

- Πολλές Εφαρμογές
  - Σε μεταγλωττιστές, πίνακες κατακερματισμού που ονομάζονται Symbol Tables αποθηκεύουν πληροφορίες για όλες τις μεταβλητές.
  - Διερεύνηση γράφων που δεν είναι εξ'αρχής γνωστοί.

Πρόβλημα: Web Search

- Έστω ότι θέλουμε να βρούμε όλες τις σελίδες (web-pages) που είναι
  - προσιτές από τη σελίδα www.cs.ucy.ac.cy μέσω links που βρίσκονται σ'αυτό το domain, και
  - περιέχουν τη λέξη-κλειδί αλγόριθμος.
- Μπορούμε να θεωρήσουμε το διαδίκτυο 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.
- Μπορούμε να χρησιμοποιήσουμε κατά-βάθος διερεύνηση (DFS) για να βρούμε όλες τις σελίδες οι οποίες είναι προσιτές από την www.cs.ucy.ac.cy
- Πως μπορούμε να διατηρήσουμε το σύνολο visited που περιέχει το σύνολο των σελίδων που έχουμε ήδη επισκεφθεί; Δεν γνωρίζουμε το γράφο (π.χ. το σύνολο των κόμβων) από την αρχή.
- Λύση: χρησιμοποιούμε πίνακα κατακερματισμού.

Ψευδοκώδικας λύσης

```
DFS(www.cs.ucy.ac.cy);


DFS(url page){

  hashtable H (hsize);

  if page contains('αλγόριθμος')
       output page;

  H.insert(page);

  for all links npage in page
       if H.find(npage) == false
            DFS(npage)
}
```