Διάλεξη 27: Τεχνικές Κατακερματισμού I

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

Ανασκόπηση Προβλήματος και Προκαταρκτικών Λύσεων

Bit-Διανύσματα

Τεχνικές Κατακερματισμού & Συναρτήσεις Κατακερματισμού

Διαχείριση Συγκρούσεων με Αλυσίδωση

Διδάσκων: Δημήτρης Ζεϊναλιπούρ

Εισαγωγή – Το πρόβλημα

Το Πρόβλημα

- Έχουμε ένα στοιχείο **u**. Θέλουμε κάποια αποδοτική δομή η οποία θα μας επιτρέψει να εκτελέσουμε τις ακόλουθες δυο πράξεις σε σταθερό χρόνο O(1).
- **Εύρεση** στοιχείου **u** σε μια συλλογή **S (**δηλ. **Find(u,S)).**
- **Εισαγωγή** του **u** στην συλλογή S **(**δηλ. **Insert(u, S))**

Παράδειγμα

- Η συλλογή S περιέχει μια μεγάλη λίστα φοιτητών. Θέλουμε να βρούμε αν ο u="Νεόφυτος Χαραλάμπους" είναι μέρος αυτής της λίστας.
- Αν δεν είναι στην λίστα, τότε θέλουμε να τον εισάγουμε.

Εισαγωγή – Ανασκόπηση Λύσεων

Ακατάλληλες Υλοποιήσεις

1. *Συνδεδεμένη λίστα*

Insert: O(1), Find: O(n)

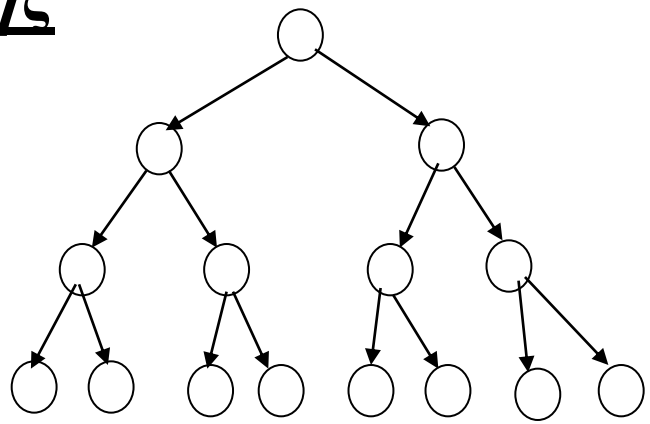
Προϋποθέτει την εκτέλεση της find, για να επιβεβαιώσουμε ότι δεν έγινε ήδη η εισαγωγή

2. *Ισοζυγισμένο δένδρο αναζήτησης*

Insert: $O(\log_m n)$, Find: $O(\log_m n)$

n: ο αριθμός των **κόμβων** και

m: ο **βαθμός** (branching factor) κάθε κόμβου.



Εισαγωγή – Ανασκόπηση Λύσεων ΙΙ

- **Υπάρχει πιο αποδοτική μέθοδος από τη χρήση ισοζυγισμένων δένδρων;**

- **Ναι,** δεδομένου ότι υπάρχει μια συνάρτηση η οποία μας επιτρέπει για κάθε στοιχείο **u**, να βρούμε την ακριβή θέση του **u** στον πίνακα.

- Έχουμε χρησιμοποιήσει αντίστοιχη ιδέα στον αλγόριθμο ταξινόμησης BucketSort.

- Μια απλή λύση είναι να απεικονίσουμε το σύνολο $S \subseteq U$ (όπου U είναι το πεδίο ορισμού της S) με ένα **διάνυσμα διφίων (διάνυσμα δυαδικών ψηφίων, *bit-vector*).**

- Παρόμοια δομή χρησιμοποιήσαμε και στον αλγόριθμο ταξινόμησης bucketsort.

Bit vectors (Διανύσματα Διφύων)

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

  Bits[u] = 1, αν u $\in$ S και Bits[u] = 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 / αναζήτησης κάποιου στοιχείου είναι σε χρόνο O(1)!

- **Πρόβλημα**: Αν το |U| είναι πολύ μεγαλύτερο από το |S| τότε σπαταλάμε πάρα πολύ χώρο!

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

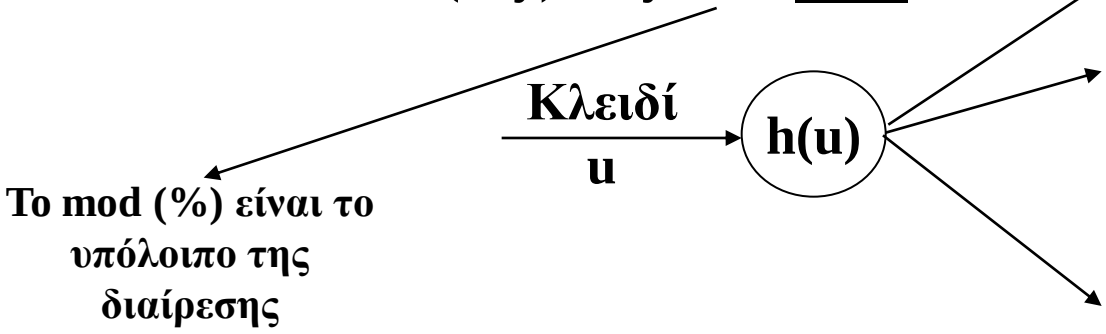
Βασική Ιδέα Κατακερματισμού

- Έστω το σύνολο ακεραίων S από το πεδίο ορισμού U

  S= {0, 1, 2, 3, 4, 5, 8, 30, 57} (U=[0..99])
- Θα φτιάξουμε ένα **Πίνακα Κατακερματισμού (Hash Table),** ο οποίος στο παράδειγμα μας έχει μέγεθος **hsize=5 (**το μέγεθος είναι συνήθως συναρτήσει του διαθέσιμου χώρου)
- Χρησιμοποιώντας κάποια **συνάρτηση κατακερματισμού (hashing function) h(key),** θα εισάγουμε τα στοιχεία του **S** στο hashtable.

Παράδειγμα (hsize: 5)

Hash Function h(key): key mod hsize



H	data
0	0, 5, 30
1	1,
2	2, 57
3	3, 8
4	4,

- Αν ψάχνουμε το key=21, τότε ξέρω ότι πρέπει να ψάξω στην θέση 1 (21 mod 5)
- Αν ψάχνω το key=4;

Κατακερματισμός – Ορισμοί & Ερωτήματα

- ***Πίνακας κατακερματισμού* (hash table)** είναι μια δομή δεδομένων που υποστηρίζει τις διαδικασίες **insert** και **find** σε (σχεδόν) σταθερό χρόνο O(1).

- 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] (εφόσον εφαρμοστεί το MOD)

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

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

- Ωστόσο τα hashtable δεν είναι ιδανικό για να ανακτούμε τα στοιχεία **ταξινομημένα,** ή γενικότερα, για **αναζητήσεις σε κάποιο εύρος (range queries).**
  - π.χ. Αν ψάχνω κλειδιά μεταξύ 2-10 (range query); – θα πρέπει δυστυχώς να κοιτάξω σε όλες τις θέσεις του πίνακα.

- **Επίσης, δημιουργούνται δύο νέα σημαντικά ερωτήματα:**
  1. ποια είναι καλή επιλογή για τη συνάρτηση κατακερματισμού h;
  2. τι θα πρέπει να γίνεται αν πολλά κλειδιά είναι στο ίδιο bucket (κάδο). Τέτοιου είδους **συγκρούσεις (collisions),** είναι πολύ πιθανό να συμβούν.

     Δηλαδή για δύο κλειδιά k_1, k_2, με $k_1 \neq k_2$, το bucket να είναι ο ίδιος $h(k_1) = h(k_2)$;

1) 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 Η.**
  3. **Θα πρέπει να υπολογίζεται εύκολα.**

- Η συνάρτηση **h** αρχικά αντιστοιχίζει το κλειδί σε κάποιο ακέραιο αριθμό a και στη συνέχεια παίρνει την τιμή «**a mod hsize**».

- Πρέπει επίσης να μπορούμε να υπολογίζουμε την συνάρτηση κατακερματισμού για **συμβολοσειρές**!

Παράδειγμα Συνάρτησης Κατακερματισμού

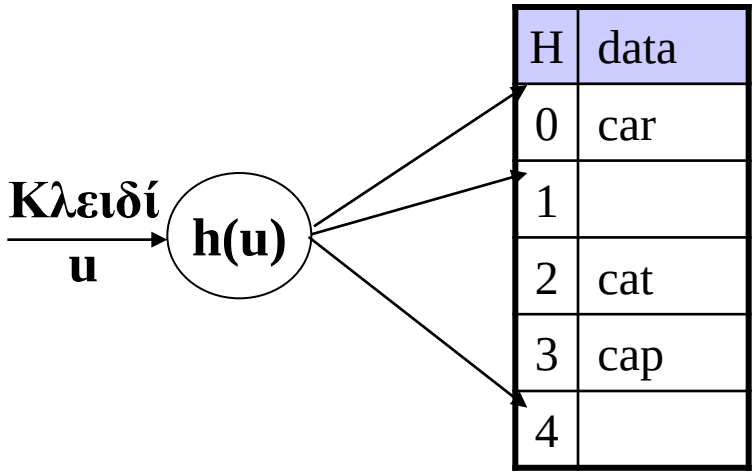
```
// Η συνάρτηση κατακερματισμού αθροίζει όλους τους
   χαρακτήρες του string s, και στην συνέχεια βρίσκει το
   υπόλοιπο από την διαίρεση με το hsize
int hash(char *s, int hsize) {
    int hash = 0;

    while ((*s) != '\0') {
          hash += (*s);
          s++;
    }
    return (hash % hsize);
}

int main() {
    char *name1 = "cat";
    char *name2 = "car";
    char *name3 = "cap";

    printf("%s hashes to bucket %d\n", name1, hash(name1, 5));
    printf("%s hashes to bucket %d\n", name2, hash(name2, 5));
    printf("%s hashes to bucket %d\n", name3, hash(name3, 5));

    return 0;
}
```



Output>

```
car hashes to bucket 2
cat hashes to bucket 0
cap hashes to bucket 3
```

1) Επιλογή Συνάρτησης Κατακερματισμού

- Έστω ότι το κλειδί είναι αλυσίδα από 3 χαρακτήρες s[0]..s[2].
- Παραδείγματα συνάρτησης κατακερματισμού είναι:

A) Function 1 (Απλή συνάρτηση):

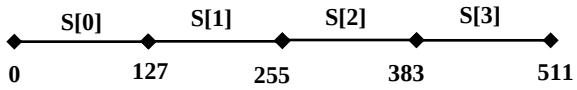
Το άθροισμα των κωδικών ASCII των χαρακτήρων s[0] + s[1] + s[2]

Πρόβλημα: Λέξεις μπορούν να έχουν το ίδιο άθροισμα

π.χ. Οι λέξεις "cat" και "tac" θα έχουν το άθροισμα 99+97+116 == 116+97+99

B) Function 2 (Βελτιωμένη Λύση): (Υποθέστε ότι οι χαρακ. είναι Ascii-7bit)

(s[0]+127*0) + (s[1]+127*1) + (s[1]+127*2)



Τώρα κάθε επί-μέρους άθροισμα απέχει το πιο λίγο κατά 128 από το επόμενο.

Ωστόσο εάν έχουμε χαρακτήρες σε κάποια άλλη κωδικοποίηση π.χ. UNICODE-16bit τότε αυτό μας δίδει πολύ μεγάλους αριθμούς.

C) Function 3 (Η συνάρτηση hashcode() στην γλώσσα JAVA)

$s[n-1] + s[n-2] * 31^{1} + \ldots + s[1]*31^{(n-2)} + s[0]*31^{(n-1)}$

Όπου **n είναι το μήκος του string**. Το hash κάποιου κενού string είναι 0.

π.χ. "cat" => n=3 => (t)**99 +** **(a)97*31^{1} +** **(c)116* 31^{2}** = 114,582

Η επιλογή του 31 δεν είναι τυχαία. Είναι Prime & επίσης υπολογίζεται αποδοτικά με ένα shift (a * 31 == (a << 5 - 1) (Άλλοι αριθμοί που έχουν τέτοιες ιδιότητες 17,31,127,…)

1. Συνάρτηση Κατακερματισμού (συνέχεια)

Συνάρτηση Κατακερματισμού

- Πολλές φορές οι συμβολοσειρές μπορεί να είναι πολύ μεγάλες «1304 Lincoln Ave, Alameda, 94501, CA, USA» , που θα κάνει ακριβό τον υπολογισμό της συνάρτησης κατακερματισμού.
- Για αυτό μπορεί να χρησιμοποιούνται επιλεκτικά κάποιοι χαρακτήρες (π.χ. κάθε 10ος)

Μέγεθος Πίνακα Κατακερματισμού

- Αν το μέγεθος του πίνακα κατακερματισμού είναι πολύ μεγάλο (π.χ. 10,000) και οι τιμές παράγονται όλες σε κάποιο μικρό-διάστημα τότε θα υπάρχουν πολλές συγκρούσεις.
- **Παράδειγμα:** Ένα σύνολο λέξεων όπου κάθε λέξη αποτελείται από 10 χαρακτήρες ASCII-7bit. Το συνολικό άθροισμα της κάθε λέξης είναι το πολύ 10x127=1,270.
- Άρα όλες οι πιθανές 127^10 γραμματοσειρές θα βρίσκονται στις πρώτες 1270%10000=1270 θέσεις του πίνακα και οι υπόλοιπες 10000-1270=8730 θα είναι άδειες παρόλο που χρησιμοποιήσαμε το MODulo.

Το θέμα της εύρεσης της πιο κατάλληλης συνάρτησης κατακερματισμού είναι δύσκολο.

Πάντοτε θα είναι πιθανή η ύπαρξη συγκρούσεων!

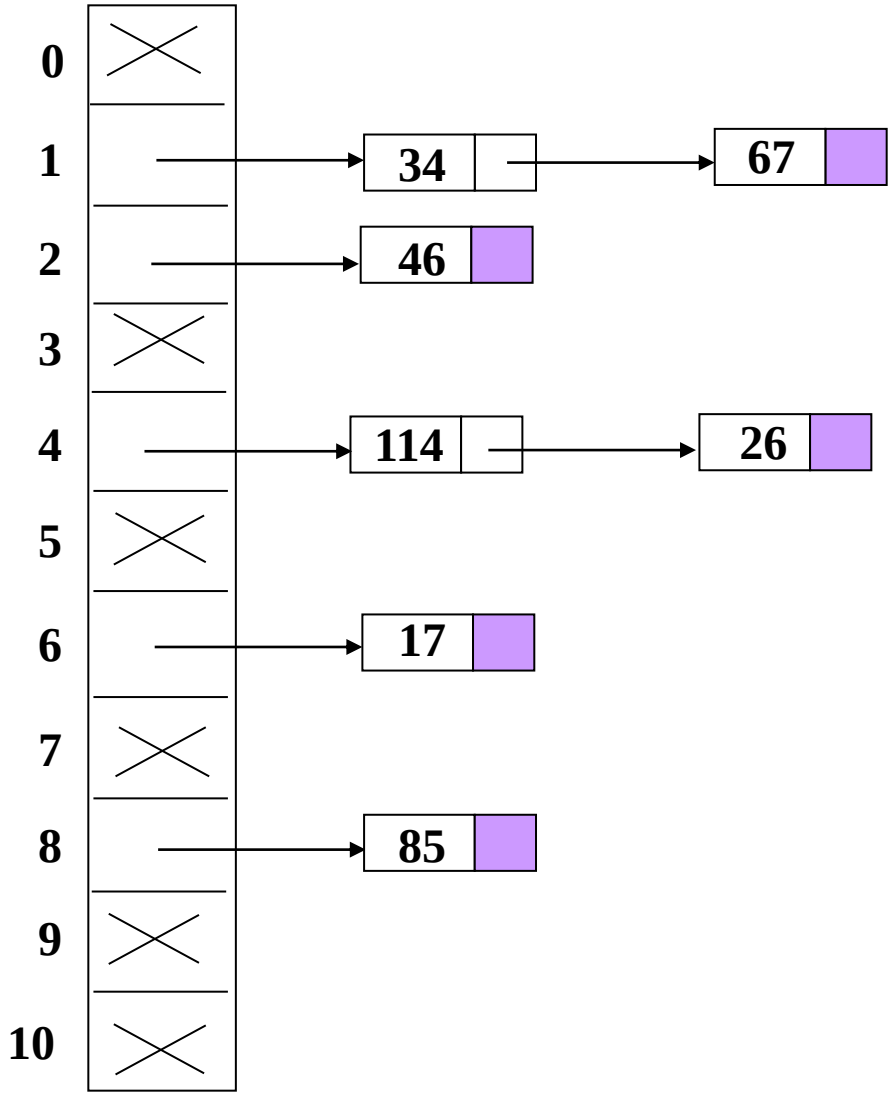
Πως γίνεται η διαχείριση συγκρουόμενων κλειδιών;

- Οι τεχνικές λύσεις διακρίνονται σε 2 κατηγορίες: μέθοδοι με **Αλυσίδωση (Chaining)** και μέθοδοι **Ανοικτής Διεύθυνσης (Open Addressing)** **[επόμενο μάθημα]**

Διαχείριση συγκρούσεων με Αλυσίδωση (Chaining)

- 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 Διαχείρισης με Αλυσίδωση



hsize = 11

Ανάλυση της Τεχνικής Αλυσίδωσης

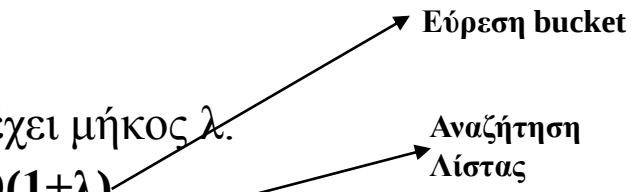
- **Εισαγωγή Στοιχείου:** απαιτεί χρόνο O(1). (προσθήκη στην αρχή)
- **Εύρεση/διαγραφή Κλειδιού k**
  - συνεπάγεται τη διέλευση της λίστας H[h(k)].
  - Για να αναλύσουμε τον χρόνο εκτέλεσης των πιο πάνω διαδικασιών ορίζουμε τον συντελεστή φορτίου **(load factor) λ** του πίνακα **H** ως τον λόγο

    λ = (αριθμός των στοιχείων που αποθηκεύει ο πίνακας) / hsize

    π.χ. 100 τιμές σε 5 buckets => λ=20.0

  - Δηλαδή, κατά μέσο όρο, κάθε λίστα του πίνακα έχει μήκος λ.
  - Το στοιχείο δεν υπάρχει (χειρίστη περίπτωση): **O(1+λ)**
  - Το στοιχείο υπάρχει (μέση περίπτωση): **O(1+λ/2)**

    Όπου O(1) κόστος για εύρεση του bucket και O(λ) & O(λ/2) αντίστοιχα για ανάλυση των στοιχείων της λίστας.



- **Ιδανικά** θα θέλαμε ο λόγος λ να έχει σταθερή τιμή (συνήθως κοντά στο 1), ώστε οι διαδικασίες να εκτελούνται σε σταθερό χρόνο.
- Στην συνέχεια θα δούμε τεχνικές **δυναμικής αύξησης / μείωσης** του πίνακα σε συνδυασμό με διαδικασίες **επανακερματισμού**, έτσι ώστε το μέγεθος του πίνακα να είναι πάντα ανάλογο του αριθμού των στοιχείων.