Δομές Δεδομένων

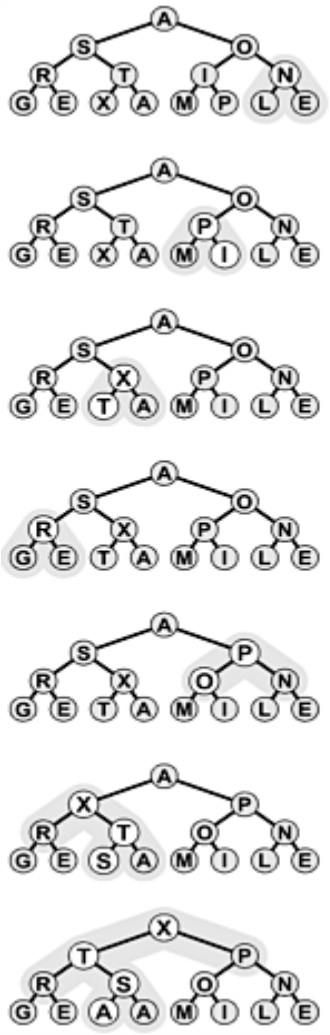
13η Διάλεξη

Πίνακες Συμβόλων

Ε. Μαρκάκης

Ο αλγόριθμος heapsort

- Παράδειγμα ταξινόμησης με σωρό
  - Δημιουργία σωρού (σε μορφή δέντρου)
    - Τα στοιχεία ταξινομούνται μερικώς
  - Ταξινόμηση με βάση το σωρό (σε μορφή πίνακα)
    - Ουσιαστικά ταξινόμηση με επιλογή
    - Επιταχύνεται όμως η επιλογή!



Ο αλγόριθμος heapsort

- Χρήση σωρού για επιλογή
  - Εντοπισμός των k μεγαλύτερων από τα N στοιχεία ή απλώς του k-οστού μεγαλύτερου
  - Η ταξινόμηση λύνει το πρόβλημα σε χρόνο NlogN
  - Ο σωρός λύνει το πρόβλημα σε γραμμικό χρόνο για μικρό k
- 1η μέθοδος (προφανής)
  - Κατασκευάζουμε το σωρό σε χρόνο O(N)
  - Αφαιρούμε k στοιχεία σε χρόνο 2klogN
- 2η μέθοδος (όχι τόσο προφανής)
  - Κατασκευάζουμε έναν ανάποδο σωρό με k στοιχεία
    - Η εξαγωγή αφαιρεί το μικρότερο στοιχείο
  - Κάνουμε N-k εισαγωγές ακολουθούμενες από εξαγωγές του ελάχιστου
    - Κάθε φορά βγάζουμε το μικρότερο από k+1 στοιχεία
  - Συνολικό κόστος 2k + 2(N-k)logk

Μέρος 4

Αναζήτηση

Περίληψη

- Πίνακες συμβόλων
- Διεπαφή πίνακα συμβόλων
- Αναζήτηση με αριθμοδείκτη
- Ακολουθιακή αναζήτηση
- Δυαδική αναζήτηση

Πίνακες συμβόλων

- Αναζήτηση και ανάκτηση πληροφοριών
  - Από τα πιο στοιχειώδη προβλήματα στην πληροφορική
- Απαιτήσεις
  - Δομές δεδομένων για την αποθήκευση των στοιχείων
  - Ταχύτητα ανάκτησης (Google, τράπεζες, δημόσιες υπηρεσίες, ...)
- Ορισμός πίνακα συμβόλων (symbol table)
  - Είναι οποιαδήποτε δομή δεδομένων που περιέχει στοιχεία με κλειδιά και υποστηρίζει τις εξής λειτουργίες:
  - Εισαγωγή νέου στοιχείου με κάποιο κλειδί
  - Επιστροφή στοιχείου με κάποιο δοσμένο κλειδί (αναζήτηση)
  - Ονομάζεται και λεξικό (dictionary)
    - Τα έντυπα λεξικά είναι πάντα ταξινομημένα
    - Τα ηλεκτρονικά λεξικά δεν είναι απαραίτητα ταξινομημένα

Διεπαφή πίνακα συμβόλων

- Πληθώρα εφαρμογών
  - Λεξικά, ευρετήρια, μεταγλωττιστές, DNS, δρομολόγηση
- Διάφορες υλοποιήσεις
  - Ταξινομημένοι ή αταξινόμητοι πίνακες ή λίστες
  - Δέντρα αναζήτησης, δυαδικά ή με μεγαλύτερο βαθμό
  - Πίνακες κατακερματισμού
- Άλλες υποστηριζόμενες λειτουργίες
  - Αφαίρεση, επιλογή k-οστού στοιχείου, ταξινόμηση, ένωση (προαιρετικά)

Διεπαφή πίνακα συμβόλων

- Γενική μορφή στοιχείων και κλειδιών
  - Κάνει τις υλοποιήσεις πιο γενικές
  - Ένας πίνακας συμβόλων περιέχει στοιχεία (π.χ. αντικείμενα με όλες τις πληροφορίες ενός πελάτη μιας τράπεζας)
  - Κάθε στοιχείο περιέχει ένα κλειδί (που είναι συγκρίσιμο) με πρόσβαση μεσω της μεθόδου key()

```
interface ITEM {
  KEY key(); /*μέθοδος πρόσβασης στο κλειδί */ }
interface KEY {
  boolean less(KEY v);
  boolean equals(KEY v); }
```

- Εναλλακτικά αντί για less και equals, μπορούμε να έχουμε τριμερή σύγκριση (μικρότερο, ίσο, μεγαλύτερο)

```
int compareTo(KEY v);
```

  - Επιστρέφει `1` αν το κλειδί `>` `v`, `0` αν είναι ίσα και `-1` διαφορετικά

Διεπαφή πίνακα συμβόλων

- Διεπαφή τάξης στοιχείων / κλειδιών
  - Οι πίνακες συμβόλων χρησιμοποιούν μόνο αυτή τη διεπαφή
  - Δημιουργία στοιχείων / κλειδιών με δύο τρόπους

```
class myItem implements ITEM {
  public KEY key()
  void read()
  void rand() //μπορούμε να προσθέσουμε και άλλες λειτουργίες εδώ
  public String toString() }
class myKey implements KEY {
  public boolean less(myKey)
  public boolean equals(myKey)
  void read()
  void rand()
  public String toString() }
```

Διεπαφή πίνακα συμβόλων

- Παράδειγμα υλοποίησης κλειδιών
  - Τα κλειδιά είναι ακέραιες τιμές

```
class myKey implements KEY {
  private int val; // η τιμή του κλειδιού
  public boolean less(KEY w) {
       return val < ((myKey) w).val; }
  public boolean equals(KEY w) {
       return val == ((myKey) w).val; }
  public void read() { val = In.getInt(); }
  public void rand() {
       val = (int) (M * Math.random()); }//M=μέγιστη
  τιμή
  public String toString() { return val + ""; } }
```

Διεπαφή πίνακα συμβόλων

- Παράδειγμα υλοποίησης στοιχείων
  - Στοιχεία με έναν πραγματικό αριθμό (π.χ. υπόλοιπο λογαριασμού)
  - Τα κλειδιά είναι ακέραιες τιμές (υλοποιούνται χωριστά) (π.χ. ΑΦΜ)

```
class myItem implements ITEM {
  private myKey bankkey;
  private float balance;
  myItem() { bankkey = new myKey(); }
  public KEY key() { return bankkey; }
  void read() { bankkey.read(); balance = In.getFloat();
  }
  void rand() {
       bankkey.rand(); balance = (float) Math.random();
  }
  public String toString() {
       return "(" + key() + " " + balance + ")"; } }
```

Διεπαφή πίνακα συμβόλων

- Παράδειγμα απλούστερης υλοποίησης στοιχείων
  - Όταν τα κλειδιά είναι στοιχειώδεις τύποι δεδομένων, μπορούμε να αποφύγουμε την υλοποίηση της KEY
  - Μικρότερο κόστος αλλά και μικρότερη ευελιξία

```
class intkeyItem {
  private int val; private float info;
  public int key() { return val; }
  void read() {
       val = In.getInt(); info = In.getFloat(); }
  void rand() {
       val = (int) (M * Math.random());
       info = (float) Math.random(); }
  public String toString() {
       return "(" + key() + " " + info + ")"; } }
```

Διεπαφή πίνακα συμβόλων

- Διεπαφή του ίδιου του πίνακα συμβόλων
  - Χρησιμοποιούμε ένα μέγιστο μέγεθος στον κατασκευαστή
    - Βολικό στις υλοποιήσεις με πίνακα

```
class ST {
  ST(int)
  int count() //μετρά το μέγεθος του πίνακα
  void insert(ITEM)
  ITEM search(KEY)
  void remove(KEY)
  ITEM select(int) //επιστροφή k-οστού μικρότερου
  public String toString() }
```

Διεπαφή πίνακα συμβόλων

- Χειρισμός διπλών κλειδιών
  - Η διεπαφή δεν διευκρινίζει τι γίνεται με τα διπλά κλειδιά
  - Σε ορισμένες περιπτώσεις δεν επιτρέπονται διπλά κλειδιά
  - Τι γίνεται όμως αν επιτρέπονται;
  - Πρώτη προσέγγιση: μία καταχώρηση ανά κλειδί
    - Χρήση λίστας για τα στοιχεία με το ίδιο κλειδί
    - Βολική όταν θέλουμε να βρίσκουμε τα στοιχεία με ίδιο κλειδί
  - Δεύτερη προσέγγιση: πολλές καταχωρήσεις ανά κλειδί
    - Επιστρέφουμε οποιαδήποτε από αυτές στην αναζήτηση
    - Μπορεί η δομή να χρειάζεται τροποποίηση
- Επιλογή υλοποίησης
  - Κάθε υλοποίηση βελτιστοποιεί συγκεκριμένες πράξεις

Διεπαφή πίνακα συμβόλων

- Παράδειγμα πελάτη πίνακα συμβόλων
  - Παραλείπει τα διπλά στοιχεία από την είσοδο

```
class DeDup {
  public static void main(String[] args) {
    int i, N = Integer.parseInt(args[0]),
        sw = Integer.parseInt(args[1]);
    ST st = new ST(N);
    for (i = 0; i < N; i++) {
      myItem v = new myItem();
      if (sw == 1) v.rand(); else v.read();
      if (st.search(v.key()) == null) {
        st.insert(v); Out.println(v + ""); } }
   Out.print(N + " keys, ");
   Out.println(N-st.count() + " dups"); } }
```

Αναζήτηση με αριθμοδείκτη

- Κατάλληλη όταν τα κλειδιά είναι μικροί ακέραιοι
  - Χρήση ενός πίνακα με αριθμοδείκτες τα κλειδιά
  - Οι θέσεις του πίνακα δείχνουν σε στοιχεία ή σε null
  - Εισαγωγή: αποθήκευση στοιχείου με κλειδί i στη θέση st[i]
  - Αναζήτηση κλειδιού i: επιστροφή θέσης i
  - Αφαίρεση κλειδιού i: τοποθέτηση null στη θέση st[i]
  - Καταμέτρηση και εκτύπωση με σάρωση του πίνακα
  - Σε κάποιες περιπτώσεις μπορούμε να έχουμε πίνακα από bits
    - bit k = 0: δεν έχει εισαχθεί το κλειδί k, διαφορετικά bit k = 1

Αναζήτηση με αριθμοδείκτη

- Παράδειγμα υλοποίησης (μέσω της intkeyItem)

```
class ST {
  private intkeyItem[] st;
  ST(int M) { st = new intkeyItem[M]; }
  int count() {
    int N = 0;
    for (int i = 0; i < st.length; i++)
      if (st[i] != null) N++; return N; }
  void insert(intkeyItem x) { st[x.key()] = x; }
  void remove(int key) { st[key] = null; }
  intkeyItem search(int key) { return st[key]; }
  intkeyItem select(int k) {
    for (int i = 0; i < st.length; i++)
      if (st[i] != null && k-- == 0) return st[i];
     return null; } }
```

Αναζήτηση με αριθμοδείκτη

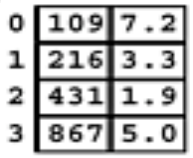
- Καταμέτρηση:
  - Η υλοποίηση εφαρμόζει ράθυμη (lazy) προσέγγιση
  - Θα μπορούσαμε να διατηρούμε έναν μετρητή στοιχείων
  - Κόστος σε κάθε λειτουργία αντί όποτε ζητηθεί καταμέτρηση
- Ανάλυση κόστους
  - Έστω ότι τα κλειδιά είναι N θετικοί ακέραιοι < M
  - Εισαγωγή, αναζήτηση και αφαίρεση: σταθερό κόστος
  - ταξινόμηση: M βήματα
  - Υπενθύμιση: ταξινόμηση με αριθμοδείκτη είναι O(M) όταν τα κλειδιά είναι όλα ακέραιοι ≤ M

Ακολουθιακή αναζήτηση

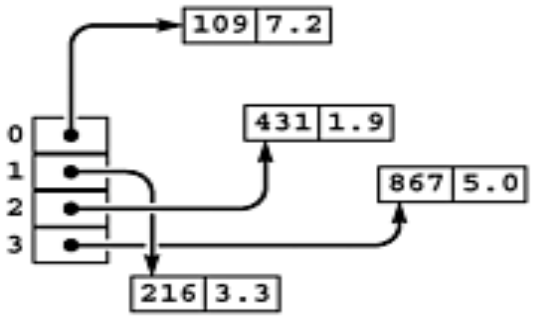
- Αν δεν μπορούμε να χρησιμοποιήσουμε τα κλειδια ως αριθμοδείκτες:

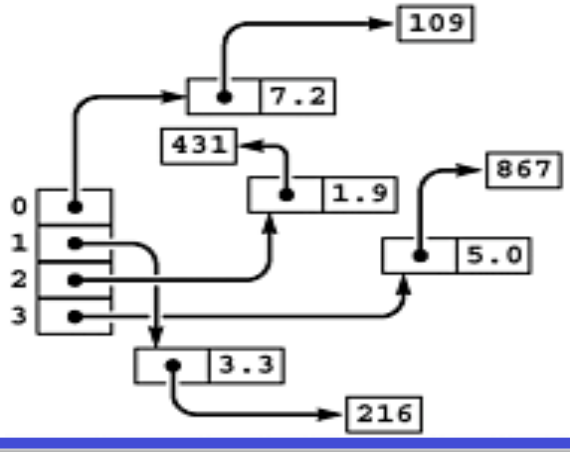
- Χρήση ταξινομημένου πίνακα
  - Η εισαγωγή απαιτεί μετακίνηση στοιχείων για να μπει στη σωστή θέση το στοιχείο
  - Η αναζήτηση απαιτεί σάρωση στοιχείων
    - Σταματάμε όταν βρούμε ίσο ή μεγαλύτερο κλειδί
  - Απλή υλοποίηση της επιλογής
  - Αφαίρεση ανάλογη με την εισαγωγή (πρέπει ο πίνακας να παραμείνει ταξινομημένος)

Ακολουθιακή αναζήτηση



- Αναπαράσταση στοιχείων / κλειδιών
  - Παράλληλοι πίνακες
  - Τάξη στοιχείων με δύο πεδία
  - Τάξη στοιχείων και τάξη κλειδιών (για στοιχειώδεις τύπους υλοποίηση ITEM και KEY κοστίζει πολύ)
    - 2 δείκτες για την προσπέλαση του κλειδιού
  - Αυξανόμενη γενικότητα
  - Αυξανόμενο κόστος





Ακολουθιακή αναζήτηση

```
class ST { private ITEM[] st; private int N = 0;
  ST(int maxN) { st = new ITEM[maxN]; }
  void insert(ITEM x) {
       int i = N++; KEY v = x.key();
       while (i > 0 && less(v, st[i-1].key())) {
             st[i] = st[i-1]; i--; }
       st[i] = x; }
  ITEM search(KEY key) {
       int i = 0; for ( ; i < N; i++)
             if (!less(st[i].key(), key)) break;
       if (i == N) return null;
       if (equals(key, st[i].key())) return st[i];
       return null; } }
   ITEM select(int k) {return st[k];}
   int count() {return N;}
```

Ακολουθιακή αναζήτηση

- Χρήση αταξινόμητου πίνακα
  - Πιο γρήγορη εισαγωγή (στο τέλος)
  - Πιο αργή αναζήτηση (πιθανή σάρωση όλου του πίνακα)
  - Πιο γρήγορη αφαίρεση (με ανταλλαγή με το τελευταίο στοιχείο), πιο αργή επιλογή
- Χρήση συνδεδεμένης λίστας
  - Ταξινομημένη ή αταξινόμητη
  - Ανάλογα πλεονεκτήματα και μειονεκτήματα με τους πίνακες
  - Η λίστα διευκολύνει την ταξινομημένη εισαγωγή
  - Κόστος (χρόνος και χώρος) διαχείρισης δεικτών
- Ταξινομημένη ή αταξινόμητη δομή;
  - Εξαρτάται από την εφαρμογή
  - Αν είναι πιο συχνή η εισαγωγή, τότε αταξινόμητη
  - Αν είναι πιο συχνή η αναζήτηση, τότε ταξινομημένη

Ακολουθιακή αναζήτηση

- Υλοποίηση με αταξινόμητη λίστα

```
class ST {
  private class Node { ITEM item; Node next;
       Node(ITEM x, Node t) { item = x; next = t; } }
  private Node head; private int N;
  ST(int maxN) { head = null; N = 0; }
  int count() { return N; }
  void insert(ITEM x) {
       head = new Node(x, head); N++; }
  private ITEM searchR(Node t, KEY key) {
       if (t == null) return null;
       if (equals(t.item.key(), key)) return t.item;
       return searchR(t.next, key); }
  ITEM search(KEY key) {return searchR(head, key); }}
```

Ακολουθιακή αναζήτηση

- Ανάλυση κόστους
  - Διακρίνουμε επιτυχημένες και αποτυχημένες αναζητήσεις
- Γενική ιδιότητα: N/2 συγκρίσεις για επιτυχή αναζήτηση κατά μέσο όρο
  - Λίστα ή πίνακας, ταξινομημένα ή αταξινόμητα
- Μη ταξινομημένα στοιχεία
  - Σταθερό κόστος εισαγωγής
  - N συγκρίσεις για αποτυχημένη αναζήτηση
  - Ισχύει για λίστα ή πίνακα
- Ταξινομημένα στοιχεία
  - Κατά μέσο όρο N/2 συγκρίσεις για εισαγωγή και επιτυχή ή ανεπιτυχή αναζήτηση
  - Ισχύει για λίστα ή πίνακα

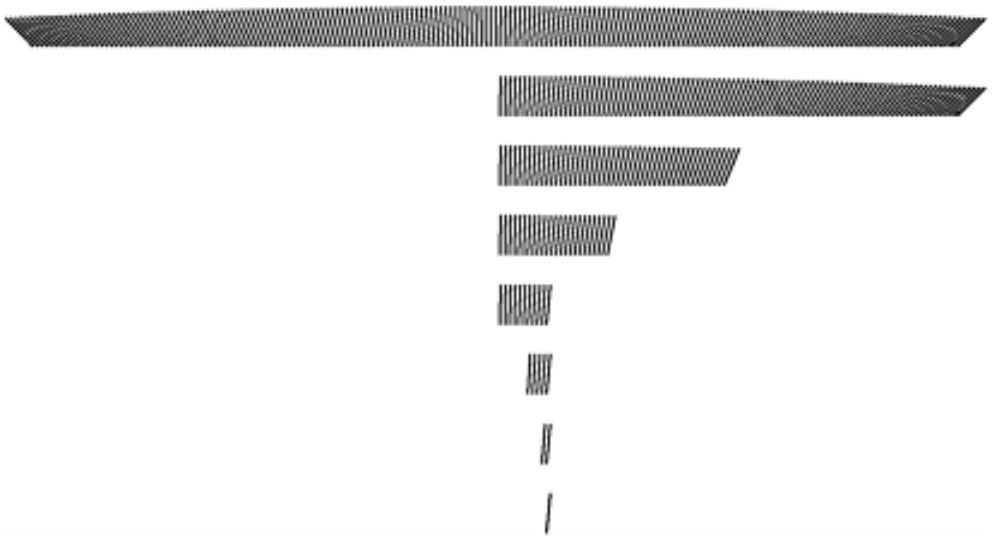
Δυαδική αναζήτηση

- Τακτική αναζήτησης διαίρει και βασίλευε
  - Διαιρούμε τον πίνακα συμβόλων σε δύο ίσα τμήματα
    - Πιθανόν να διαφέρουν κατά 1 στοιχείο
  - Συνεχίζουμε στο τμήμα που περιέχει το ζητούμενο κλειδί
  - Χρησιμοποιεί ταξινομημένο πίνακα

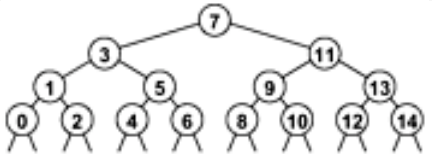
```
private ITEM searchR(int l, int r, KEY v) {
  if (l > r) return null;
  int m = (l+r)/2;
  if (equals(v, st[m].key())) return st[m];
  if (less(v, st[m].key()))
       return searchR(l, m-1, v);
  else return searchR(m+1, r, v); }
ITEM search(KEY v) {
  return searchR(0, N-1, v); }
```

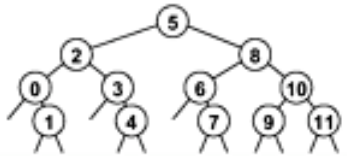
Δυαδική αναζήτηση

- Απόδοση δυαδικής αναζήτησης
  - Το πολύ floor(logN)+1 συγκρίσεις, επιτυχείς ή μη αναζητήσεις
- Πρόβλημα: διατήρηση του πίνακα ταξινομημένου
  - Αν τα στοιχεία εισάγονται δυναμικά έχουμε πρόβλημα
    - Χρειάζεται συνεχής μετακίνηση των στοιχείων του πίνακα
  - Η συνδεδεμένη λίστα εμποδίζει την άμεση προσπέλαση



Δυαδική αναζήτηση





- Προκαθορισμένη ακολουθία συγκρίσεων
  - Εξαρτάται μόνο από το μέγεθος του αρχείου
  - Ακολουθούμε μοναδική διαδρομή στο δένδρο
- Χειρισμός διπλών κλειδιών
  - Η ταξινόμηση τοποθετεί τα κλειδιά σε συνεχόμενες θέσεις
  - Η αναζήτηση οδηγεί σε κάποια από αυτές τις θέσεις
  - Απλές προσθήκες για σάρωση των γειτονικών θέσεων
- Επιτάχυνση με παρεμβολή
  - Υπολογίζουμε πού περίπου πρέπει να βρίσκεται το κλειδί
    - Υποθέτουμε ομοιόμορφη κατανομή κλειδιών
  - Διασπάμε τον πίνακα εκεί και όχι στο μέσο
    - m = l + (v – a[l].key()) * (r – l) / (a[r].key() – a[l].key())