Δομές Δεδομένων

8η Διάλεξη:

Ταξινόμηση

Ε. Μαρκάκης

Υπενθύμιση

- Εργαστήρια την επόμενη εβδομάδα
- Πρόγραμμα εργαστηρίων αναρτημένο στο eclass
- Εργασία 1 θα αναρτηθεί την Τρίτη, παράδοση 20/11

Περίληψη

- ΑΤΔ για πολυώνυμα
- Ταξινόμηση: οι κανόνες του παιχνιδιού
- Ταξινόμηση με εισαγωγή
- Γενικές Υλοποιήσεις

ΑΤΔ για πολυώνυμα

- Εφαρμογή: Symbolic Mathematics. Προγράμματα σαν τα Matlab, Mathematica, Maple χρησιμοποιούν μεθόδους για διαχείριση πολυωνύμων
- Συμβολική επεξεργασία, π.χ.:

$$\left(1 - x + \frac{x^2}{2} - \frac{x^3}{6}\right)\left(1 + x + x^2 + x^3\right) = 1 + \frac{x^2}{2} + \frac{x^3}{3} - \frac{2x^4}{3} + \frac{x^5}{3} - \frac{x^6}{6}$$

- Θέλουμε μεθόδους για
  - Πρόσθεση πολυωνύμων
  - Πολλαπλασιασμό πολυωνύμων
  - Αποτίμηση ενός πολυωνύμου σε συγκεκριμένο x

ΑΤΔ για πολυώνυμα

- Διασύνδεση ΑΤΔ:

```
class Poly
{ // υλοποιήσεις και ιδιωτικά μέλη κρυμμένα
   Poly(int, int) //φτιάχνει το πολυώνυμο cx^N
   double eval(double) //αποτίμηση
   void add(Poly) // πρόσθεση πολυωνύμων
   void mult(Poly) //πολλαπλασιασμός
   public String toString()
}
```

ΑΤΔ για πολυώνυμα

- Έστω ότι θέλουμε να υπολογίζουμε τις εκφράσεις $(x+1)^N$, π.χ.
  - $(x+1)^2 = x^2 + 2x + 1$
  - $(x+1)^4 = x^4 + 4x^3 + 6x^2 + 4x + 1$
- Πρόγραμμα πελάτη:

```
public class Binomial {
    public static void main(String[] args)
    { int N = Integer.parseInt(args[0]);
      double p = Double.parseDouble(args[1]);
      Poly y = new Poly(1, 0); //y=1
      Poly t = new Poly(1, 0); //t=1
      t.add(new Poly(1, 1)); //t = x+1
      for (int i = 0; i < N; i++)
         y.mult(t);
}
```

ΑΤΔ για πολυώνυμα

- Υλοποίηση: Θα δούμε την eval (υπόλοιπες μέθοδοι στο βιβλίο)
- Εσωτερική αναπαράσταση πολυωνύμου βαθμού N:
  - πίνακας N+1 στοιχείων με τους συντελεστές
- Αποτίμηση πολυωνύμου, Απλοϊκή μέθοδος: $O(N^2)$ πολλαπλ/μοί
- Αλγόριθμος του Horner: O(N)

$$a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 = (((a_4x + a_3)x + a_2)x + a_1)x + a_0$$

ΑΤΔ για πολυώνυμα

```
class Poly {
   private int n; //βαθμός πολυωνύμου + 1
   private int[] a; //συντελεστές
   Poly(int c, int N) { //δημιουργεί το πολυώνυμο
     a = new int[N+1];  // cx^N
     n = N+1; a[N] = c;
     for (int i = 0; i < N; i++) a[i] = 0;
   }
   double eval(double d) { //υπολογισμός της
     double t = 0.0;        // τιμής στο d
     for (int i = n-1; i >= 0; i--)
        t = t*d + (double) a[i];
     return t; }
}
```

ΑΤΔ - Σύνοψη

- Επιτρέπουν το χτίσιμο επιπέδων αφαίρεσης
- Διαχωρισμός μεταξύ πελάτη και υλοποίησης
- Διευκολύνουν τη σχεδίαση modular programming
  - Επαναχρησιμοποίηση κώδικα
- Βολικός μηχανισμός για τη σύγκριση αλγορίθμων

Κεφάλαιο 6

Ταξινόμηση

Οι κανόνες του παιχνιδιού

- Στόχος ταξινόμησης
  - Είσοδος: Δίνεται ένα σύνολο N στοιχείων
    - Υλοποίηση των συνόλων με πίνακες ή λίστες
  - Κάθε στοιχείο περιλαμβάνει ένα κλειδί
  - Έξοδος: Αναδιάταξη των στοιχείων σύμφωνα με τα κλειδιά
    - Αύξουσα / φθίνουσα αριθμητική / λεξικογραφική σειρά
- Είδη ταξινόμησης
  - Εσωτερική ταξινόμηση: τα στοιχεία χωράνε στη μνήμη
  - Εξωτερική ταξινόμηση: τα στοιχεία αποθηκεύονται στο δίσκο
  - Προσαρμοστική ταξινόμηση: εξαρτάται από τα στοιχεία
  - Σε εσωτερική ταξινόμηση έχουμε πιο εύκολη πρόσβαση στα στοιχεία
  - Θα ασχοληθούμε κυρίως με εσωτερική ταξινόμηση

Οι κανόνες του παιχνιδιού

- Ταξινόμηση σε πίνακα
- Κάποιες συμβάσεις:
  - Θέλουμε ταξινόμηση σε αύξουσα σειρά
  - Πρόγραμμα οδήγησης: γεμίζει τον πίνακα και καλεί την sort()
  - Sort(): υλοποιεί έναν αλγόριθμο ταξινόμησης, βασίζεται στις:
  - less(): σύγκριση 2 στοιχείων
  - exch(): αντιμετάθεση 2 στοιχείων
  - compExch(): σύγκριση 2 στοιχείων και αντιμετάθεση αν είναι απαραίτητο
  - αποδεσμεύεται η sort() από τον τύπο δεδομένων

Ταξινόμηση με εισαγωγή (Insertion Sort)

- Η μέθοδος που χρησιμοποιούν οι χαρτοπαίκτες

- Δείτε εδώ:
- **http://www.youtube.com/watch?v=ejpFmtYM8Cw**
- Ή ψάξτε στο youtube “insertion sort”

Ταξινόμηση με εισαγωγή (Insertion Sort)

- Η ιδέα του αλγορίθμου:

- Βρόχος στα δεδομένα εισόδου
- Στην επανάληψη k:
  - κοιτάμε το στοιχείο a[k]
  - Ο αριστερός υποπίνακας a[0],…,a[k-1] είναι ήδη ταξινομημένος (από τις προηγούμενες επαναλήψεις)
  - Βρίσκουμε ποια είναι η σωστή θέση για το a[k]
  - Κανουμε τις απαραίτητες αντιμεταθέσεις

Ταξινόμηση με εισαγωγή (Insertion Sort)

```
class ArraySortBasic {
  static boolean less(double v, double w) {
       return v < w; }
  static void exch(double[] a, int i, int j) {
       double t = a[i]; a[i] = a[j]; a[j] = t; }
  static void compExch(double[] a, int i, int j) {
       if (less(a[j], a[i])) exch (a, i, j); }
  static void sort(double[] a, int p, int r) {
       for (int i = p+1; i <= r; i++)
            for (int j = i; j > p; j--)
                 compExch(a, j-1, j); }
public static void main(String[] args) {
  double a[] = new double[Integer.parseInt(args[0])];
  for (int i = 0; i < N; i++) a[i] = Math.random();
  sort(a, 0, N-1);}
```

Πολυπλοκότητα

- Αξιολόγηση αλγορίθμων ταξινόμησης
  - Βασικές λειτουργίες: συγκρίσεις και αντιμεταθέσεις
    - Η σύγκριση απαιτεί ανάγνωση
    - Η μετάθεση απαιτεί ανάγνωση και εγγραφή
  - Οι πιο απλές ταξινομήσεις είναι συνήθως $O(N^2)$
  - Οι πιο περίπλοκες ταξινομήσεις είναι O(NlogN)
  - Ορισμένες ειδικές ταξινομήσεις είναι O(N)
- Πολυπλοκότητα της Insertion Sort
- Worst case: αν ο πίνακας αρχικά είναι σε φθίνουσα σειρά
  - $1^η$ επανάληψη: 1 σύγκριση και μετάθεση
  - $2^η$ επανάληψη: 2 συγκρίσεις και μεταθέσεις
  - ...
  - Συνολικά $1+2+...+N-1 = N(N-1)/2 = O(N^2)$

Ιδιότητες και επιπλέον είδη ταξινόμησης

- Ευσταθής και μη ταξινόμηση
  - Πολλές φορές έχουμε στοιχεία με πολλά κλειδιά
  - Ευσταθής ταξινόμηση: διατηρεί τη σχετική σειρά ως προς το δεύτερο κλειδί
  - Έστω ότι έχουμε ένα αρχείο με ονόματα και βαθμούς
    - Αν ταξινομούμε με βάση τους βαθμούς, τα ονόματα που ισοβαθμούν μένουν ταξινομημένα σε ευσταθή ταξινόμηση
  - Οι περισσότερες απλές μέθοδοι είναι ευσταθείς
  - Οι περισσότερες από τις εξελιγμένες μεθόδους δεν είναι ευσταθείς
- Έμμεση ταξινόμηση (indirect sort)
  - Αναδιατάσσουμε τους δείκτες αλλά όχι τα ίδια τα στοιχεία

Adams	1
Black	2
Brown	4
Jackson	2
Jones	4
Smith	1
Thompson	4
Washington	2
White	3
Wilson	3

Adams	1
Smith	1
Washington	2
Jackson	2
Black	2
White	3
Wilson	3
Thompson	4
Brown	4
Jones	4

Adams	1
Smith	1
Black	2
Jackson	2
Washington	2
White	3
Wilson	3
Brown	4
Jones	4
Thompson	4

Γενικές υλοποιήσεις

- Αποδέσμευση της ταξινόμησης από τους τύπους
  - Οι αλγόριθμοι ταξινόμησης δεν εξαρτώνται από τα κλειδιά
    - Αρκεί να μπορούν να τα συγκρίνουν
  - Οι αλγόριθμοι ταξινόμησης δεν εξαρτώνται από τα στοιχεία
    - Αρκεί να μπορούν να τα αντιμεταθέσουν
- Γενική βιβλιοθήκη ταξινόμησης
  - Ο πελάτης μεταβιβάζει στη sort() έναν πίνακα στοιχείων
  - Η sort() καλεί τις μεθόδους σύγκρισης και αντιμετάθεσης
  - Η βιβλιοθήκη της Java χρησιμοποιεί τη διεπαφή Comparable
  - Κάθε αντικείμενο μπορεί να υλοποιεί την Comparable
  - Αρκεί να υλοποιεί τη μέθοδο v.compareTo(w)
    - -x αν v<w, +x αν v>w, 0 αν v=w
  - Κάθε πίνακας τέτοιων αντικειμένων μπορεί να ταξινομηθεί

Γενικές υλοποιήσεις

- Μια απλούστερη μορφή της βιβλιοθήκης!
  - Αρχικά ορίζουμε έναν γενικό τύπο αντικειμένου ITEM
  - Η διεπαφή ITEM αρκεί να παρέχει τη μέθοδο less()

```
interface ITEM
  { boolean less(ITEM v); }
```

  - Στη συνέχεια ορίζουμε τη γενική τάξη ταξινόμησης Sort
    - Η τάξη δεν περιέχει κατάσταση
    - Όλες οι μέθοδοι είναι static
  - Βοηθητική μέθοδος less()
    - Παίρνει ως παραμέτρο δύο στοιχεία τύπου ITEM
  - Βοηθητικές μέθοδοι exch() και compExch()
    - Παίρνουν ως παραμέτρους πίνακες με αντικείμενα ITEM
    - Αντιμεταθέτουν στοιχεία τύπου ITEM

Γενικές υλοποιήσεις

- Βοηθητική μέθοδος sort()
  - Καλείται από τους πελάτες με πίνακα τύπου ITEM και δύο όρια για το εύρος του πίνακα

```
class Sort {
  static boolean less(ITEM v, ITEM w) {
       return v.less(w); }
  static void exch(ITEM[] a, int i, int j) {
       ITEM t = a[i]; a[i] = a[j]; a[j] = t; }
  static void compExch(ITEM[] a, int i, int j) {
       if (less(a[j], a[i])) exch (a, i, j); }
  static void sort(ITEM[] a, int p, int r) {
   example(a, p, r); }
  static void example(ITEM[] a, int p, int r) {
       for (int i = p+1; i <= r; i++)
            for (int j = i; j > p; j--)
                 compExch(a, j-1, j); } }
```

Γενικές υλοποιήσεις

- ΑΤΔ στοιχείων
  - Υλοποιεί τη διεπαφή ITEM
  - Πρέπει να παρέχει υλοποίηση της less()
  - Μπορούμε να προσθέσουμε ανάγνωση, τυχαίο ορισμό και εμφάνιση στοιχείων

```
class myItem implements ITEM {
  public boolean less(ITEM) //υλοποιεί τη σύγκριση
  void read() //διαβάζει το στοιχείο από την είσοδο
  void rand() //παράγει ένα τυχαίο στοιχείο
  public String toString()
}
```

Γενικές υλοποιήσεις

- Παράδειγμα 1: τύποι ταξινόμησης ακεραίων
  - Υλοποίηση τάξης myItem
  - Χρησιμοποιεί τις πράξεις των ακεραίων
  - Προσοχή στις μετατροπές τύπων!

```
class myItem implements ITEM {
  private int key; //το στοιχείο ως προς το οποίο
  ταξινομούμε
  public boolean less(ITEM w) {
       return key < ((myItem) w).key; }
  void read() { key = In.getInt(); }
  void rand() { key = (int) (1000 * Math.random()); }
  public String toString() { return key + ""; } }
```

Γενικές υλοποιήσεις

- ΑΤΔ πίνακα στοιχείων
  - Πίνακας στοιχείων άγνωστου τύπου
  - Μπορούμε να προσθέσουμε ανάγνωση και τυχαίο ορισμό στοιχείων πίνακα
  - Παρέχει τη δυνατότητα εμφάνισης και ταξινόμησης (υπο)πινάκων
- Διασύνδεση

```
class myArray {
  myArray(int)
  void rand() //παράγει τυχαία στοιχεία
  void read() //διαβάζει τον πίνακα από την είσοδο
  void show(int, int) //τυπώνει μέρος του πίνακα
  void sort(int, int) /*ταξινομεί (υπο)πίνακα που
  καθορίζουν
                              οι παράμετροι */
}
```

Γενικές υλοποιήσεις

- Παράδειγμα 1: τύποι ταξινόμησης ακεραίων
  - Υλοποίηση τάξης myArray με στοιχεία myItem
  - Υλοποίηση γίνεται με χρήση των μεθόδων της myItem

```
class myArray {
  private myItem[] a; private int N;
  myArray(int N) {
        this.N = N; a = new myItem[N];
        for (int i = 0; i < N; i++)
              a[i] = new myItem(); }
  void rand() {
        for (int i = 0; i < N; i++) a[i].rand(); }
  void read() {
        for (int i = 0; i < N; i++)
              if (!In.empty()) a[i].read(); }
  void sort(int p, int r) { Sort.sort(a, p, r); } }
```

Γενικές υλοποιήσεις

- Πρόγραμμα οδήγησης ταξινόμησης (driver program)

```
class ArraySort {
  public static void main(String[] args)
{
        int N = Integer.parseInt(args[0]);
        myArray A = new myArray(N);
        if (args.length < 2) A.rand();
        else A.read();
        A.sort(0, N-1);
        A.show(0, N-1); }
}
```

Γενικές υλοποιήσεις

- Παράδειγμα 2: τύποι ταξινόμησης εγγραφών
- Έστω μία τραπεζική εφαρμογή με αντικείμενα που περιέχουν:
  - Αριθμό λογαριασμού (int), όνομα (String), υπόλοιπο (double)
  - Αρχικά ορίζουμε την τάξη των εγγραφών
  - Μπορούμε να επιλέξουμε το κλειδί ταξινόμησης
    - Η SortKeyField δείχνει ποιο κλειδί θέλουμε

```
class Record {
  int id; //αριθμός λογαριασμού
  double balance; //υπόλοιπο
  String who; //όνομα
  static int SortKeyField = 0;
  public String toString() {
       return id+" "+balance + " " + who; } }
```

Γενικές υλοποιήσεις

- Παράδειγμα 2: τύποι ταξινόμησης εγγραφών
  - Στη συνέχεια ορίζουμε την τάξη myItem
    - Κληρονομεί τη Record και υλοποιεί την ITEM
    - Η less() ταξινομεί ανάλογα με τη SortKeyField

```
class myItem extends Record implements ITEM {
  public boolean less(ITEM w) {
        myItem r = (myItem) w;
        switch (SortKeyField) {
        case 2: return who.compareTo(r.who) < 0;
        case 1: return balance < r.balance;
        default: return id < r.id; } }
   }
```

Γενικές υλοποιήσεις

- Ιδιότητες γενικής υλοποίησης
  - Λειτουργεί με οποιονδήποτε τύπο αντικειμένων
  - Όχι τόσο αποδοτική για ενσωματωμένους τύπους
    - Απαιτείται ένα πρόσθετο επίπεδο αναφορών
  - Στην πράξη όμως εργαζόμαστε με αντικείμενα (π.χ. εγγραφές) και όχι με στοιχειώδεις τύπους
  - Υλοποιεί ταξινόμηση δεικτών
- Ταξινόμηση δεικτών
  - Οι πίνακες αντικειμένων περιέχουν αναφορές
  - Η αντιμετάθεση γίνεται σε επίπεδο αναφορών
  - Αποφεύγεται η αντιγραφή των αντικειμένων
  - Το κόστος της αντιμετάθεσης είναι μικρό
    - Ανάλογο με την αντιμετάθεση ακεραίων