Δομές Δεδομένων

6η Διάλεξη

Αναδρομικές Εξισώσεις και Αφηρημένοι Τύποι Δεδομένων

Ε. Μαρκάκης

Περίληψη

- Χρήση αναδρομικών εξισώσεων στην ανάλυση αλγορίθμων
- Αφηρημένοι τύποι δεδομένων
- Συλλογές στοιχείων
- Στοίβα ώθησης προς τα κάτω
- Παραδείγματα πελατών για στοίβες
- Υλοποιήσεις στοίβας
- Γενικές υλοποιήσεις

Παραδείγματα ανάλυσης αλγορίθμων

- Σύγκριση ακολουθιακής και δυαδικής αναζήτησης
  - Έστω ένας πίνακας ακεραίων με N στοιχεία
  - Ψάχνουμε τη θέση ενός στοιχείου στον πίνακα
  - Πόσες πράξεις σύγκρισης χρειάζονται;
- Ακολουθιακή αναζήτηση (Sequential Search)
  - Ψάχνουμε από το ένα άκρο του πίνακα ως το άλλο
  - Κατάλληλη και για μη ταξινομημένο πίνακα
  - Το εύρος του πίνακα που ψάχνουμε δίνεται με τις παραμέτρους p, r

```
static int search(int a[], int v, int p, int r) {
  int i;
  for (i = p; i <= r; i++)
       if (v == a[i]) return i;
  return -1;
}
```

Παραδείγματα ανάλυσης αλγορίθμων

- Ανάλυση ακολουθιακής αναζήτησης
  - Έστω N το εύρος του πίνακα που ψάχνουμε (N = r-p+1)
  - Ο χρόνος εκτέλεσης εξαρτάται από τα δεδομένα
  - Χειρότερη δυνατή περίπτωση: η αναζήτηση ανεπιτυχής ή το στοιχείο είναι στην τελευταία θέση: O(N) βήματα
  - Μέσο κόστος επιτυχούς αναζήτησης: Υποθέτουμε ότι όλες οι θέσεις είναι εξίσου πιθανοί στόχοι
  - Μέσο κόστος = (1 + 2 + … + N) / N = (N+1)/2 ≈ N/2
  - Καλύτερο από τη χειρότερη περίπτωση αλλά και πάλι O(N)
- Αν ο πίνακας είναι ταξινομημένος;
  - Η αναζήτηση δεν εξαντλεί απαραίτητα τον πίνακα
  - Μέσο κόστος N/2 βήματα, μέγιστο κόστος N βήματα

Παραδείγματα ανάλυσης αλγορίθμων

- Δυαδική αναζήτηση (Binary Search)
  - Μόνο για ταξινομημένο πίνακα
  - Εξετάζει το μεσαίο στοιχείο του πίνακα
  - Συνεχίζει με το κατάλληλο μισό του πίνακα (αναδρομή)

```
static int search(int a[], int v, int p, int r)
  {
  while (r >= p) {
        int m = (p+r)/2;
        if (v == a[m]) return m;
        if (v < a[m]) r = m-1;
        else p = m+1; }
  return -1; }
```

Παραδείγματα ανάλυσης αλγορίθμων

- Ανάλυση δυαδικής αναζήτησης
  - Σε κάθε έλεγχο μένει ο μισός πίνακας
  - Χειρότερη περίπτωση:

$$T_N \leq T_{\lfloor N/2 \rfloor} + O(1)$$

  - Για να βρούμε την πολυπλοκότητα πρέπει να λύσουμε την αναδρομική σχέση
  - Όχι πάντα εύκολο αλλά επιλύσιμο για πολλές κατηγορίες αναδρομικών εξισώσεων

1488	1488			
1578	1578			
1973	1973			
3665	3665			
4426	4426			
4548	4548			
5435	5435	5435	5435	5435
5446	5446	5446	5446	
6333	6333	6333		
6385	6385	6385		
6455	6455	6455		
6504				
6937				
6965				
7104				
7230				
8340				
8958				
9208				
9364				
9550				
9645				
9686				

Βασικές αναδρομικές εξισώσεις

- Ακολουθούν την αποσύνθεση ενός προβλήματος
  - Ο χρόνος εκτέλεσης ορίζεται ως συνάρτηση του χρόνου εκτέλεσης σε μικρότερα προβλήματα
  - Τελικά φτάνουμε σε στοιχειώδη προβλήματα
  - Η λύση δίνει έναν τύπο για το χρόνο εκτέλεσης
- Παράδειγμα 1
  - Βρόχος στα δεδομένα εισόδου και απαλοιφή ενός στοιχείου πριν την επόμενη επανάληψη
  - $C_N=C_{N-1}+N$, $C_1=1$
  - Λύση: $C_N = C_{N-1}+N = C_{N-2}+(N-1)+N = C_{N-3}+ (N-2)+ (N-1)+ N$
  - $= 1+2+3+...+N = N(N+1)/2$, δηλαδή $O(N^2)$

Βασικές αναδρομικές εξισώσεις

- Παράδειγμα 2
  - Κάθε βρόχος διχοτομεί τα δεδομένα, επόμενη επανάληψη κοιτάζει τα μισά δεδομένα
  - $C_N=C_{N/2}+1$, $C_1=1$
  - Τέτοιες σχέσεις είναι χρήσιμο να τις αναλύουμε πρώτα για δυνάμεις του 2, έστω ότι $N = 2^n$

$$C_N = C_{N/2} + 1 = C_{2^{n-1}} + 1 = (C_{2^{n-2}} + 1) + 1$$

$$= \ldots = C_1 + 1 + 1 + \ldots + 1 = n + 1$$

  - Άρα $C_N = \log N+1$, δηλαδή $O(\log N)$
  - Όταν το N δεν είναι δυναμη του 2, αποδεικνύεται με επαγωγή ότι $C_N \leq \lceil \log N+1 \rceil$
  - Άρα για κάθε N, $C_N = O(\log N)$
  - **Πόρισμα:** Ο αλγόριθμος δυαδικής αναζήτησης τρέχει σε χρόνο $O(\log N)$

Βασικές αναδρομικές εξισώσεις

- Με ίδια τεχνική αναλύονται πολλά άλλα προβλήματα
- Παράδειγμα 3 (προσοχή: λάθος τύπος στο βιβλιο, σελ. 74!)
  - Κάθε βρόχος διχοτομεί τα στοιχεία αφού τα εξετάσει
  - Ορισμός: $C_N=C_{N/2}+N$, $C_1=1$
  - Λύση: $C_N = 2N$, δηλαδή $O(N)$
- Παράδειγμα 4
  - Κάθε βρόχος εξετάζει, διχοτομεί και κάνει αναδρομικές κλήσεις και στα 2 υποπροβλήματα μεγέθους N/2
  - Ορισμός: $C_N=2C_{N/2}+N$, $C_1=1$
  - Λύση: $C_N = NlogN$, δηλαδή $O(NlogN)$
- Παράδειγμα 5
  - Κάθε βρόχος διχοτομεί και κάνει αναδρομική κλήση και για τα 2 υποπροβλήματα
  - Ορισμός: $C_N=2C_{N/2}+1$, $C_1=1$
  - Λύση: $C_N = 2N$, δηλαδή $O(N)$

Όρια ανάλυσης αλγορίθμων

- Εγγύηση: άνω φράγμα χρόνου εκτέλεσης
  - Χρόνος εκτέλεσης στη χειρότερη περίπτωση
  - Χρησιμοποιούμε το χειρότερο δυνατό σύνολο εισόδου
  - Πόσο πιθανή είναι η χειρότερη περίπτωση;
    - Ένας αλγόριθμος μπορεί να είναι σχεδόν πάντα γρήγορος
- Πρόβλεψη: μέσος χρόνος εκτέλεσης
  - Χρόνος εκτέλεσης κατά μέσο όρο
  - Χρησιμοποιούμε ένα «τυχαίο» σύνολο εισόδου
  - Πόσο αντιπροσωπευτικό είναι το σύνολο αυτό;
    - Πόσο τυχαία είναι πραγματικά η είσοδος;
- Περιορισμός: κάτω φράγμα χρόνου εκτέλεσης
  - Απόδειξη ότι κάθε αλγόριθμος θέλει τουλάχιστον τόσο χρόνο
  - Γενικά πολύ πιο δύσκολη η απόδειξη κάτω φραγμάτων

Κεφάλαιο 4

Αφηρημένοι Τύποι Δεδομένων

Αφηρημένοι τύποι δεδομένων

- Όλα τα υπολογιστικά συστήματα βασίζονται σε στρώσεις αφαίρεσης (layers of abstraction)
  - Το μοντέλο του bit: 0 ή 1 (χωρίς να μας απασχολεί πως επιτυγχάνεται, π.χ. τσιπ πυριτίου)
  - Γλώσσα προγραμματισμού: ο προγραμματιστής χρειάζεται να γνωρίζει τις διαθέσιμες δομές δεδομένων και εντολές της γλώσσας και όχι απαραίτητα πώς υλοποιείται ο compiler
  - Σχεδίαση αλγορίθμων: ανεξάρτητο από γλώσσα προγραμματισμού (συνήθως)
  - Σε κάθε επίπεδο υπάρχει διαχωρισμός μεταξύ υλοποίησης και χρήσης

Αφηρημένοι Τύποι Δεδομένων

- ΑΤΔ: πελάτης, υλοποίηση και διεπαφή (ή διασύνδεση)
  - Διεπαφή: περιγραφή τύπου δεδομένων και πράξεων
  - Πελάτης: πρόγραμμα που χρησιμοποιεί τις πράξεις
  - Υλοποίηση: κώδικας που υλοποιεί τις πράξεις
  - Προσοχή: Τα προγράμματα πελάτες δεν μπορούν να προσπελάσουν τα δεδομένα παρά μόνο μέσω των μεθόδων που παρέχει η διασύνδεση
- Διαχωρισμός διεπαφής και υλοποίησης
  - Επιτρέπει το χτίσιμο επιπέδων αφαίρεσης
  - Επαναχρησιμοποίηση λογισμικού
  - Παραδείγματα: στοίβες, ουρές, πίνακες συμβόλων

Αφηρημένοι Τύποι Δεδομένων

- ΑΤΔ: πελάτης, υλοποίηση και διεπαφή (ή διασύνδεση)
  - Διεπαφή: περιγραφή τύπου δεδομένων και πράξεων
  - Πελάτης: πρόγραμμα που χρησιμοποιεί τις πράξεις
  - Υλοποίηση: κώδικας που υλοποιεί τις πράξεις
  - Προσοχή: Τα προγράμματα πελάτες δεν μπορούν να προσπελάσουν τα δεδομένα παρά μόνο μέσω των μεθόδων που παρέχει η διασύνδεση
- Πλεονεκτήματα ΑΤΔ
  - Ο πελάτης μπορεί να χρησιμοποιήσει διάφορες υλοποιήσεις
  - Η υλοποίηση μπορεί να χρησιμοποιηθεί από διάφορους πελάτες
  - Σχεδίαση αρθρωτού (άρα επαναχρησιμοποιήσιμου) κώδικα
  - Επιτρέπει τη χρήση της πλέον βελτιστοποιημένης υλοποίησης

Αφηρημένοι τύποι δεδομένων

- Παράδειγμα: ας επανέλθουμε στην τάξη Point για επεξεργασία σημείων στο επίπεδο
  - Περιέχει δεδομένα και μεθόδους

```
class Point {
  double x, y;
  Point() { x = Math.random(); y = Math.random(); }
  Point(double x, double y) {this.x = x;this.y = y;}
  double r() { return Math.sqrt(x*x + y*y); }
  double theta() { return Math.atan2(y, x); }
  double distance(Point p) {
       double dx = x - p.x, dy = y - p.y;
       return Math.sqrt(dx*dx + dy*dy); }
  public String toString() {
       return "(" + x + "," + y + ")"; }
}
```

Αφηρημένοι τύποι δεδομένων

- Οι χρήστες (προγράμματα-πελάτες) δεν χρειάζεται να γνωρίζουν πώς αποθηκεύονται τα σημεία
- Χειρισμός σημείων μέσω μεθόδων και μόνο
  - Γνωρίζουμε ποιες είναι οι μέθοδοι αλλά όχι πώς λειτουργούν
- Διεπαφή ΑΤΔ σημείων
  - Περιλαμβάνει μόνο τα δημόσια μέλη της τάξης

```
class Point {
  Point()
  Point(double, double)
  double x()
  double y()
  double r()
  double theta()
  double distance(Point)
  public String toString() }
```

Αφηρημένοι τύποι δεδομένων

- Υλοποίηση 1: χρήση καρτεσιανών συντεταγμένων

```
class Point {
  private double x, y;
  Point() { x = Math.random(); y = Math.random(); }
  Point(double x, double y) {this.x = x; this.y = y;}
  double x() { return x; }
  double y() { return y; }
  double r() { return Math.sqrt(x*x + y*y); }
  double theta() { return Math.atan2(y, x); }
  double distance(Point p) {
       double dx = x - p.x();
       double dy = y - p.y();
       return Math.sqrt(dx*dx + dy*dy); }
  public String toString() {
       return "(" + x + ","+y+")"; } }
```

Αφηρημένοι τύποι δεδομένων

- Διαφορές με πριν:
  - Τα δεδομένα είναι `private`
  - 2 νέες μέθοδοι `x(), y()`
- Σε ιδιωτικά μέλη μπορεί να γίνει αναφορά μόνο μέσα από την ίδια την κλάση
- Πρόσβαση στα δεδομένα από το πρόγραμμα πελάτη γίνεται μόνο μέσω των μεθόδων `x(), y()`
- Έστω τώρα ότι θέλουμε να αλλάξουμε την αναπαράσταση σε πολικές συντεταγμένες αντί για καρτεσιανές
- Πρέπει να αλλάξουν και όλα τα προγράμματα-πελάτες?

Αφηρημένοι τύποι δεδομένων

- Υλοποίηση 2: χρήση πολικών συντεταγμένων

```
class Point {
    private double r, theta;
    Point() {
      double x = Math.random(), y = Math.random();
      this = new Point(x, y); }
    Point(double x, double y) {
      r = Math.sqrt(x*x + y*y); theta = Math.atan2(y, x); }
    double r() { return r; }
    double theta() { return theta; }
    double x() { return r*Math.cos(theta); }
    double y() { return r*Math.sin(theta); }
    double distance(Point p) {
      double dx = x - p.x(); double dy = y - p.y();
      return Math.sqrt(dx*dx + dy*dy); }
    public String toString() {
      return "(" + x() + ", " + y() + ")"; } }
```

Αφηρημένοι τύποι δεδομένων

- Όσοι πελάτες χρησιμοποιούσαν τις `x()`, `y()` πριν, μπορούν να εξακολουθήσουν να τις χρησιμοποιούν και τώρα
- Καμία απολύτως αλλαγή στον κώδικα του πελάτη
- Μας επιτρέπει να κάνουμε αλλαγές στην αναπαράσταση προς βελτιστοποίηση της υλοποίησης
- Διόρθωση σφαλμάτων χωρίς αλλαγές στους πελάτες

Συλλογές στοιχείων

- Γενικευμένες ουρές: θεμελιώδεις δομές δεδομένων
  - Τιμές: σύνολα αντικειμένων (άγνωστος τρόπος αποθήκευσης), μπορεί να είναι `int, char, Object`,...
  - Πράξεις: εισαγωγή νέου στοιχείου, απομάκρυνση στοιχείου, έλεγχος αν το σύνολο είναι κενό
  - Υλοποίηση με λίστες ή πίνακες
  - Η σημασία της εισαγωγής είναι προφανής
  - Ποιο αντικείμενο επιστρέφει η απομάκρυνση; (π.χ. Αυτό που έχει μείνει περισσότερη ώρα, λιγότερη ώρα, το 3^{ο} από το τέλος,....)
  - Κάθε κριτήριο απομάκρυνσης $\rightarrow$ νέα δομή δεδομένων

Συλλογές στοιχείων

- Στοίβα (Stack)
  - Απομακρύνουμε το αντικείμενο που εισήχθη πιο πρόσφατα
  - Αναλογία: πλύσιμο πιάτων, στοίβα των δίσκων στο εστιατόριο
  - Ορολογία: push (εισαγωγή) και pop (απομάκρυνση)
  - Αναφέρεται και ως ουρά LIFO (Last In First Out)

Συλλογές στοιχείων

- Ουρά (Queue)
  - Απομακρύνουμε το αντικείμενο που εισήχθη πιο παλιά
  - Αναλογία: ουρά στο κυλικείο
  - Ορολογία: enqueue (εισαγωγή) και dequeue (απομάκρυνση)
  - Αναφέρεται συνήθως ως ουρά FIFO (First In First Out)

Στοίβα ώθησης προς τα κάτω

- Πράξεις στοίβας
  - Pop(): απομάκρυνση του αντικειμένου που εισήχθη τελευταίο
  - Push(): εισαγωγή ενός αντικειμένου στη στοίβα
  - isEmpty(): είναι άδεια η στοίβα;
- Παράδειγμα: στοίβα χαρακτήρων
  - Χαρακτήρας: Push, Αστερίσκος: Pop

Στοίβα ώθησης προς τα κάτω

- Διεπαφή στοίβας ακεραίων
  - Ο πελάτης μπορεί να δίνει και το μέγιστο πλήθος στοιχείων

```
class intStack {
 // υλοποιήσεις και ιδιωτικά μέλη κρυμμένα
  intStack(int)
  boolean isEmpty()
  void push(int)
  int pop() }
```

Παραδείγματα πελατών για στοίβες

- Πολλές σημαντικές εφαρμογές
  - Υπολογισμός αριθμητικών παραστάσεων
  - Εικονική μηχανή Java
  - Λειτουργίες Undo και Back
  - Γλώσσα περιγραφής σελίδων PostScript
- Υπολογισμός αριθμητικών παραστάσεων
  - Έστω ότι θέλουμε να υπολογίσουμε την παράσταση
    - 5 * (((9 + 8) * (4 * 6)) + 7)
  - Η παράσταση αυτή είναι σε ενθεματική μορφή (infix)
  - Μετατροπή σε μεταθεματική μορφή (postfix)
    - 5 9 8 + 4 6 * * 7 + *
  - Κάθε τελεστής ακολουθεί τα ορίσματά του
  - Δεν χρειάζονται παρενθέσεις