Δομές Δεδομένων

7η Διάλεξη

Αφηρημένοι Τύποι Δεδομένων

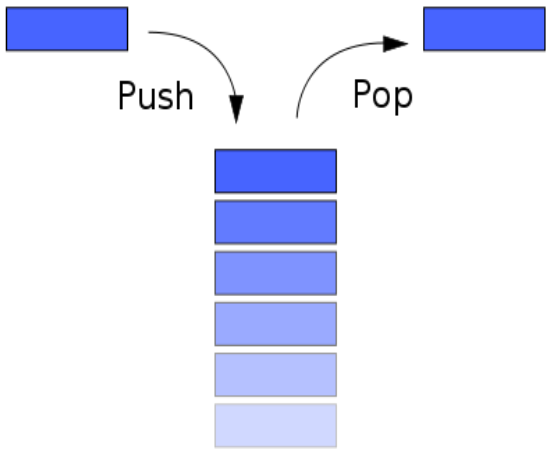
Ε. Μαρκάκης

Περίληψη

- Στοίβα ώθησης προς τα κάτω
- Παραδείγματα πελατών για στοίβες
- Υλοποιήσεις στοίβας με πίνακες και λίστες
- Γενικές υλοποιήσεις
- Ουρές FIFO
- Υλοποιήσεις ουράς με πίνακες και λίστες
- ΑΤΔ για προβλήματα συνδετικότητας

Υπενθύμιση από προηγούμενο μάθημα

- Πράξεις στοίβας - Ορολογία
  - `Pop()` : απομάκρυνση και επιστροφή του αντικειμένου που εισήχθη τελευταίο
  - `Push()` : εισαγωγή ενός αντικειμένου στη στοίβα
  - `isEmpty()` : είναι άδεια η στοίβα;
- Διεπαφή στοίβας ακεραίων ή χαρακτήρων
  - 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() }
```

```
class charStack {
  charStack(int)
  boolean isEmpty()
  void push(char)
  char pop() }
```

Παραδείγματα πελατών για στοίβες

- Πολλές σημαντικές εφαρμογές
  - Υπολογισμός αριθμητικών παραστάσεων
  - Εικονική μηχανή Java
  - Κλήσεις μεθόδων κατά την εκτέλεση ενός προγράμματος
  - 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

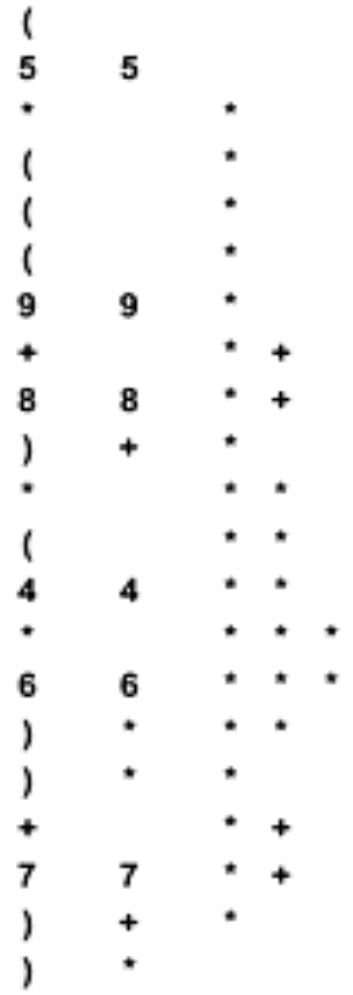
Παράδειγμα 1: Μετατροπή σε μεταθεματική μορφή

- Μεταθεματική αναπαράσταση
  - Φυσικός τρόπος οργάνωσης σειράς υπολογισμών
  - Χρησιμοποιείται και από την PostScript (παράδειγμα στο βιβλίο)
- Μετατροπή ενθεματικής σε μεταθεματική
- Είσοδος: μία παράσταση σε ενθεματική μορφή
- Έξοδος: να τυπώσουμε την αντίστοιχη μεταθεματική μορφή
  - Ιδέα: χρησιμοποιούμε μία στοίβα χαρακτήρων για τους τελεστές. Όταν διαβάζουμε:
  - Τελεστέο: εκτύπωση
  - Τελεστή: push τελεστή
  - Αριστερή παρένθεση: αγνοείται
  - Δεξιά παρένθεση: pop τελεστή και εκτύπωση

Παράδειγμα 1: Μετατροπή σε μεταθεματική μορφή

- Μετατροπή ενδοθεματικής σε μεταθεματική

```
class InfixToPostfix {
  public static void main(String[] args) {
    char[] a = args[0].toCharArray();
    int N = a.length;
    charStack s = new charStack(N);
    for (int i = 0; i < N; i++) {
       if (a[i] == ')')
              Out.print(s.pop() + " ");
       if ((a[i] == '+') || (a[i] == '*'))
              s.push(a[i]);
       if ((a[i] >= '0') && (a[i] <= '9'))
              Out.print(a[i] + " "); }
    Out.println(""); } }
```



Παράδειγμα 2: Υπολογισμός αριθμητικής παράστασης

- Είσοδος: μία παράσταση σε μεταθεματική μορφή
- Έξοδος: Υπολογισμός της τιμής της παράστασης
  - Ιδέα: Χρησιμοποιούμε μία στοίβα ακεραίων για τους τελεστέους. Όταν διαβάζουμε:
  - Τελεστέο: push τελεστέου
  - Τελεστή: pop δύο τελεστέων, πράξη, push αποτελέσματος

Παράδειγμα 2: Υπολογισμός αριθμητικής παράστασης

- Υπολογισμός μεταθεματικής παράστασης

```
class Postfix {
 public static void main(String[] args) {
        char[] a = args[0].toCharArray();
        int N = a.length;
        intStack s = new intStack(N);
        for (int i = 0; i < N; i++) {
            if (a[i] == '+') s.push(s.pop() + s.pop());
            if (a[i] == '*') s.push(s.pop() * s.pop());
            if ((a[i] >= '0') && (a[i] <= '9'))
            /* Αν έχουμε >1 ψηφία θέλουμε
               κάτι σαν την Integer.parseint */
                    s.push(0);
             while((a[i] >= '0') && (a[i] <= '9'))
                    s.push(10*s.pop() + (a[i++]-'0')); }
        Out.println(s.pop() + ""); } }
```

5	5			
9	5	9		
8	5	9	8	
+	5	17		
4	5	17	4	
6	5	17	4	6
*	5	17	24	
*	5	408		
7	5	408	7	
+	5	415		
*	2075			

Υλοποιήσεις στοίβας

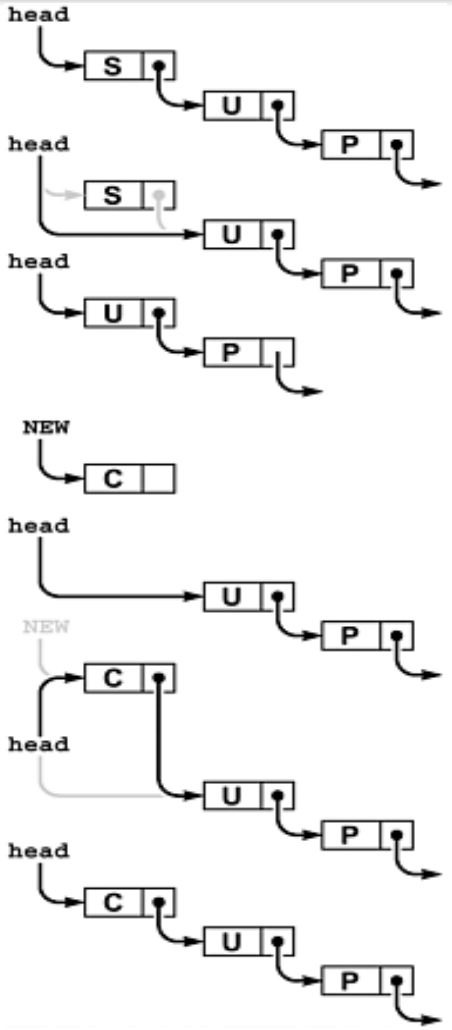
- Υλοποίηση στοίβας με πίνακα
  - Υποθέτουμε στοίβα ακεραίων

```
class intStack {
  private int[] s;
  private int N; //= αριθμός στοιχείων στη στοίβα
  intStack(int maxN) { s = new int[maxN]; N = 0; }
  boolean isEmpty() { return (N == 0); }
  void push(int item) { s[N++] = item; }
  int pop() { return s[--N]; }}
```

- Μεγέθυνση και σμίκρυνση
  - Ιδανικά θα θέλαμε να έχουμε αυτόματη αλλαγή μεγέθους
  - Απλός κανόνας αλλαγής μεγέθους
    - Όταν ο πίνακας φτάσει στο 100%, τον διπλασιάζουμε
    - Όταν ο πίνακας πέσει στο 25%, τον ημιδιπλασιάζουμε
    - Σε κάθε περίπτωση απαιτείται αντιγραφή στοιχείων

Υλοποιήσεις στοίβας

- Υλοποίηση στοίβας με λίστα
  - Κόμβοι με στοιχείο και δείκτη
  - Διατηρούμε έναν δείκτη στην κεφαλή
  - Push: εισάγει ένα στοιχείο στην αρχή
  - Pop: απομακρύνει ένα στοιχείο από την αρχή
- Σύγκριση υλοποιήσεων:
  - Εύκολη μεγέθυνση και σμίκρυνση σε λίστες
  - Η μεγέθυνση κοστίζει σε πίνακες
  - Επιβάρυνση σε λίστες ο χειρισμός αναφορών
    - Σε μνήμη και επεξεργασία



Υλοποιήσεις στοίβας

- Υλοποίηση στοίβας με λίστα

```
class intStack {
  private Node head;
  private class Node {
        int item; Node next;
        Node(int item, Node next) {
              this.item = item; this.next = next; } }
  intStack(int maxN) { head = null; }
  boolean isEmpty() { return (head == null); }
  void push(int item) { head = new Node(item, head);}
  int pop() {
        int v = head.item; Node t = head.next;
        head = t; //head δείχνει στον επόμενο
        return v; } }
```

Παρατηρήσεις

- Σχεδόν ίδιος κώδικας για στοίβες χαρακτήρων ή άλλων τύπων δεδομένων
- Στην υλοποίηση με λίστες η κατασκευάστρια μέθοδος αγνοεί την παράμετρό της
- Και στις 2 υλοποιήσεις, οι λειτουργίες push και pop γίνονται σε χρόνο O(1) (εκτός αν χρειαστεί να κάνουμε μεγέθυνση/σμίκρυνση στην υλοποίηση με πίνακες)
- Πρέπει πάντα να προσθέτουμε και τα κατάλληλα exceptions (π.χ. αν γίνει pop σε κενή στοίβα ή push σε γεμάτη στοίβα)

Γενικές υλοποιήσεις

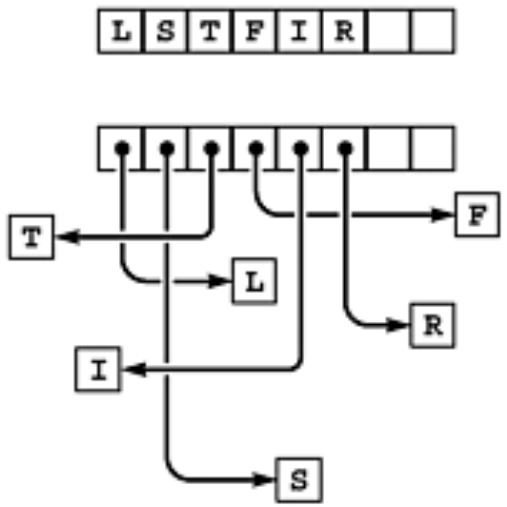
- Υλοποίηση στοίβας για διαχείριση οποιουδήποτε τύπου
  - Χρήση του ίδιου κώδικα ως βάση
  - Διαφοροποίηση μόνο του τύπου των στοιχείων
  - Μπορούμε να εισάγουμε οποιοδήποτε αντικείμενο χρησιμοποιώντας τον τύπο Object
  - Κληρονομικότητα για μετατροπή σε άλλους τύπους

```
class Stack {
  private Object[] s;
  private int N;
  Stack(int maxN) { s = new Object[maxN]; N = 0; }
  boolean isEmpty() { return (N == 0); }
  void push(Object item) { s[N++] = item; }
  Object pop() {
       Object t = s[--N]; s[N] = null; return t; }
}
```

Γενικές υλοποιήσεις

- Περιορισμοί του τύπου Object
  - Οι στοιχειώδεις τύποι θέλουν πρόσθετη μετατροπή με χρήση κλάσεων περιτύλιξης (wrapper classes)
    - Αντί για `s.push(x)` έχουμε `s.push(new Integer(x))`
    - Αντί για `s.pop()` έχουμε `((Integer) s.pop().intValue() )`
  - Οι στοιχειώδεις τύποι επιβαρύνονται πολύ
  - Αδυναμία εντοπισμού σφαλμάτων τύπου
    - Δεν ελέγχονται οι τύποι των στοιχείων

```
Apple a = new Apple();
Orange b = new Orange();
s.push(a); s.push(b);
...
a = ((Apple) s.pop()); //εξαίρεση
b = ((Orange) s.pop());
```



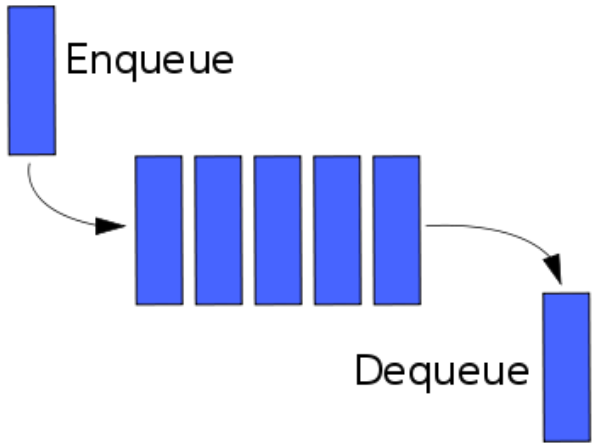
Γενικές υλοποιήσεις

- Εναλλακτική λύση: Χρήση κλάσεων προσαρμογής (adapter classes)
  - Μετατροπή της γενικής κλάσης Stack σε εξειδικευμένη
  - Ρητή μετατροπή τύπων γίνεται στην υλοποίηση και όχι από τον πελάτη

```
class intStack {
  private Stack S;
  intStack(int maxN) { S = new Stack(maxN); }
  boolean isEmpty() { return S.isEmpty(); }
  void push(int item) { S.push(new
  Integer(item));}
  int pop() {return ((Integer)
  S.pop()).intValue();}}
```

Ουρές FIFO

- Απομακρύνουμε πάντα το στοιχείο που έχει παραμείνει στην ουρά το μεγαλύτερο χρονικό διάστημα
- Πράξεις ουράς – Ορολογία:
  - `Get()` ή `Dequeue()`: απομάκρυνση του αντικειμένου που εισήχθη πρώτο
  - `Put()` ή `Enqueue()`: εισαγωγή ενός αντικειμένου στην ουρά
  - `isEmpty()`: είναι άδεια η ουρά;



Ουρές FIFO

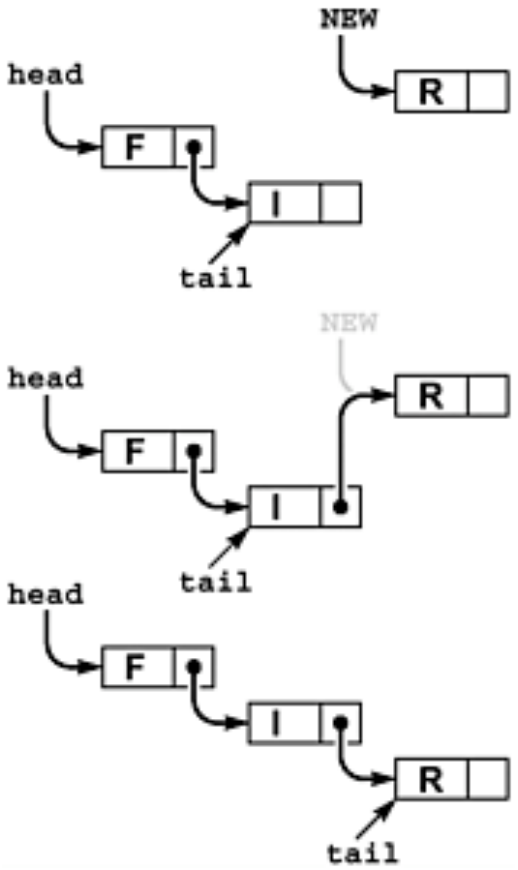
- Παραδείγματα:
  - Δρομολόγηση διεργασιών, δρομολόγηση σε δίκτυα
  - Λίστες αναμονής
- Διεπαφή ουράς ακεραίων
  - Δίνουμε και το μέγιστο μέγεθος

```
class intQueue {
  intQueue(int)
  boolean isEmpty()
  void put(int)
  int get() }
```

Ουρές FIFO

- Υλοποίηση ουράς με λίστα
  - Χρειαζόμαστε δύο δείκτες
  - head: το παλιότερο στοιχείο
    - Χρήση από την get()
  - tail: το νεότερο στοιχείο
    - Χρήση από την put()
- Ορισμός κόμβων λίστας
  - Εσωτερική τάξη της ουράς

```
private class Node {
      int item; Node next;
      Node(int item) {
            this.item = item;
            next = null; } }
```



Ουρές FIFO

- Υλοποίηση ουράς με λίστα

```
class intQueue {
  private Node head, tail;
  intQueue(int max) { head = null; tail = null; }
  boolean isEmpty() { return (head == null); }
  void put(int item) {
       Node t = tail;
       tail = new Node(item);
       if (isEmpty()) head = tail;
       else t.next = tail; }
  int get() {
       int v = head.item; Node t = head.next;
       head = t;
       return v; } }
```

Ουρές FIFO

- Υλοποίηση ουράς με πίνακα
  - Κοστίζει πολύ να μετακινούμε στοιχεία μέσα στον πίνακα
  - Αντί αυτού θα μετακινούμε τους δείκτες head, tail
  - Χρήση πίνακα N στοιχείων με κυκλικό τρόπο
    - Χρήση αριθμητικής modulo
    - Τα στοιχεία της ουράς είναι από το head μέχρι το tail με αναδίπλωση στο 0
  - Head: δείχνει το πρώτο στοιχείο της ουράς
    - Πρώτα modulo και μετά αύξηση
  - Tail: δείχνει **μετά** το τελευταίο στοιχείο
    - Πρώτα αύξηση και μετά modulo
  - Head % N == Tail αν η ουρά είναι άδεια
  - Πρέπει να προστεθεί και έλεγχος αν η ουρά είναι γεμάτη: (Head – 1) % N == Tail
  - Μεγέθυνση ουράς: αντιγραφή στοιχείων

Ουρές FIFO

- Υλοποίηση ουράς με πίνακα
  - Μπορούμε να έχουμε και μέθοδο isFull()

```
class intQueue {
  private int[] q;
  private int N, head, tail;
  intQueue(int maxN) {
        q = new int[maxN + 1];
        N = maxN + 1; head = N; tail = 0; }
  boolean isEmpty() { return (head % N == tail); }
  void put(int item) {
        q[tail++] = item;
        tail = tail % N; } //όταν tail=N πάει στο 0
  int get() {
        head = head % N; // αναδίπλωση όταν head = N
        return q[head++]; } }
```

Αφηρημένοι Τύποι Δεδομένων για προβλήματα συνδετικότητας

Στο Κεφάλαιο 1 είδαμε 4 αλγορίθμους για το πρόβλημα συνδετικότητας

1. Γρήγορη εύρεση
2. Γρήγορη ένωση
3. Σταθμισμένη γρήγορη ένωση
4. Σταθμισμένη γρήγορη ένωση με συμπίεση μονοπατιών

Αφαίρεση: όλοι οι παραπάνω αλγόριθμοι στηρίζονται σε λειτουργίες εύρεσης και ένωσης

Αφηρημένοι Τύποι Δεδομένων για προβλήματα συνδετικότητας

- Διασύνδεση για ΑΤΔ Union-Find

```
class UF  {
      UF(int)
      boolean find(int, int) //βρίσκει αν συνδέονται 2 στοιχεία
      void unite(int, int)   }
```

- Πρόγραμμα πελάτη

```
class connectivity {
       public static void main(String[] args)
       { int p, q, N = Integer.parseInt(args[0]);
         UF info = new UF(N);
         for (In.init(); !In.empty(); )
            { p = In.getInt(); q = In.getInt();
              if (!info.find(p, q))
                    { info.unite(p, q);
                      Out.println(p + "-" + q);
                     } } } }
```

Αφηρημένοι Τύποι Δεδομένων για προβλήματα συνδετικότητας

- Υλοποίηση του ΑΤΔ UF με σταθμισμένη γρήγορη ένωση

```
class UF  {
  private int[] id, sz;
  private int find(int x) //βρίσκει τη ρίζα του x
   { while (x != id[x]) x = id[x]; return x; }
  UF(int N)
  { id = new int[N]; sz = new int[N];
    for (int i = 0; i < N; i++)
      { id[i] = i; sz[i] = 1; } }
  boolean find(int p, int q) //ελέγχει αν p,q συνδέονται
  { return (find(p) == find(q)); }
  void unite(int p, int q)
  { int i = find(p), j = find(q);
    if (i == j) return;
    if (sz[i] < sz[j])
       { id[i] = j; sz[j] += sz[i]; }
    else { id[j] = i; sz[i] += sz[j]; } } }
```

Διασυνδέσεις ΑΤΔ

- Τι μηχανισμούς παρέχει η Java για το διαχωρισμό υλοποίησης/προγραμμάτων πελάτη?

1. abstract κλάσεις
   - Δημόσιες μέθοδοι μιας abstract κλάσης πρέπει να επανοριστούν σε μια extended κλάση που θα κάνει την υλοποίηση
2. Interfaces
   - Στο παράδειγμά μας μπορούμε να έχουμε:

```
interface uf {
        int find(int x);
        boolean find(int p, int q);
        void unite(int p, int q);
 }
```

   - Η κλάση UF δηλώνεται ως: `class UF implements uf`
   - Δεν επιτρέπονται constructors μέσα σε ορισμό interface