Δομές Δεδομένων

Απλοί ΑΤΔ - Στοίβα και Ουρά

Α. Συμβώνης
Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Εφαρμοσμένων Μαθηματικών και Φυσικών Επιστημών
Τομέας Μαθηματικών

Στοίβα
[Stack]

Στοίβα [Stack]

- Ο ΑΤΔ στοίβα αποθηκεύει αντικείμενα αυθαίρετου τύπου
- Οι εισαγωγές και οι διαγραφές ακολουθούν το σχήμα LIFO (Last-In First-Out)
- Φυσικό αντίστοιχο: μια στοίβα από πιάτα, κερματοθήκη
- Βασικές μέθοδοι της στοίβας:
  - `push(Object)`: εισαγωγή στοιχείου (στην κορυφή της στοίβας)
  - `Object pop()`: αφαιρεί και επιστρέφει το τελευταίο εισαχθέν στοιχείο της στοίβας (το οποίο βρίσκεται στην κορυφή της στοίβας)
- Βοηθητικές μέθοδοι της στοίβας:
  - `Object top()`: επιστρέφει το τελευταίο εισαχθέν στοιχείο της στοίβας χωρίς να το αφαιρέσει
  - `int size()`: επιστρέφει το πλήθος των στοιχείων που είναι αποθηκευμένα στη στοίβα
  - `boolean isEmpty()`: υποδηλώνει εάν η στοίβα είναι άδεια ή όχι

Στοίβα [Stack]

- Η διαπροσωπία `Stack`

```
1  public interface Stack<E> {
2
3    void push(E e);
4    E pop();
5
6    E top();
7    int size();
8    boolean isEmpty();
9  }
10
```

Δεν είναι ίδια με τη κλάση της βιβλιοθήκης της Java!
java.util.Stack

Στοίβα [Stack]

Παράδειγμα: Χρήση `Stack`

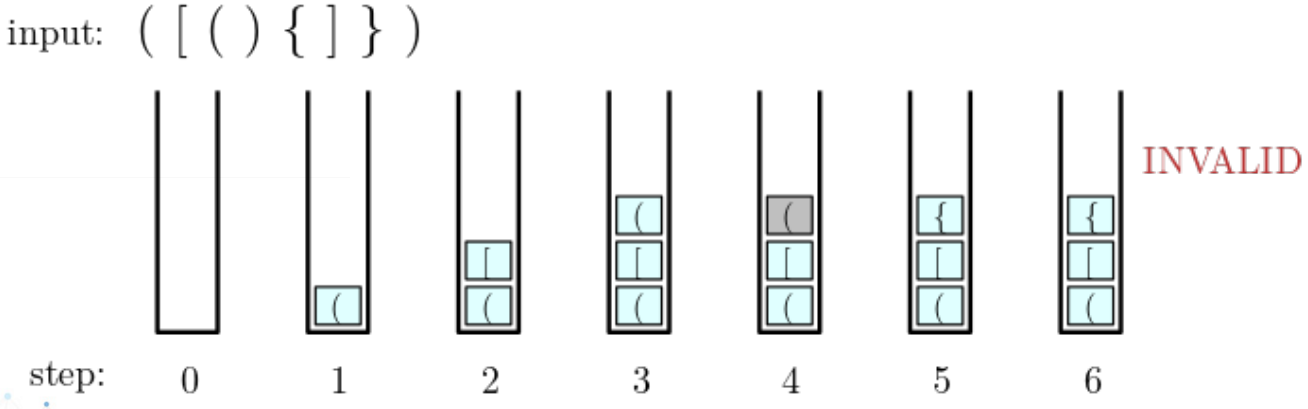
Method	Return value	Περιεχόμενα στοίβας
`push(5)`	-	`( 5 )`
`push(3)`	-	`( 5 , 3 )`
`size()`	2	`( 5 , 3)`
`pop()`	3	`( 5 )`
`isEmpty()`	`false`	`( 5 )`
`pop()`	5	`()`
`isEmpty()`	`true`	`()`
`pop()`	`null`	`()`
`push(7)`	-	`( 7 )`
`push(9)`	-	`( 7, 9 )`
`top()`	9	`( 7, 9 )`
`pop()`	9	`( 7 )`

Στοίβα [Stack]

- Εφαρμογές της στοίβας
  - Ιστορικό σελίδων επίσκεψης σε web browser
  - Αναίρεση ενεργειών σε πρόγραμμα επεξεργασίας κειμένου
  - Αλυσίδα κλήσης μεθόδων στο JVM (Java Virtual Machine)
    - Υλοποίηση αναδρομής -- stackOverflowError
  - Έλεγχος σωστού ζευγαρώματος παρενθέσεων (`()`, `[]`, `{}`)
  - Έλεγχος σωστού ζευγαρώματος HTML λέξεων κλειδιών
  - Υπολογισμός της τιμής μιας αριθμητικής έκφρασης με τελεστές διαφορετικής προτεραιότητας
    - Οι τελεστές */ έχουν μεγαλύτερη προτεραιότητα από τους +-
    - Για διαδοχικούς τελεστές ίδιας προτεραιότητας οι πράξεις γίνονται από τα αριστερά προς τα δεξιά
    - 3+4-2*8*4-5+8/2
  - Αντιστροφή σειράς στοιχείων λίστας

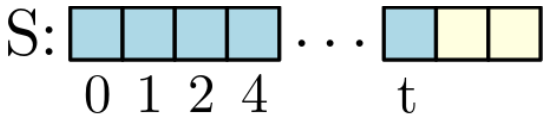
Στοίβα [Stack]

- Έλεγχος σωστού ζευγαρώματος παρενθέσεων
  - Διαβάζουμε διαδοχικά τα σύμβολα
    - Για κάθε σύμβολο '(', '[', '{' που ανοίγει μια παρένθεση, το τοποθετούμε (`push`) στη στοίβα
    - Για κάθε σύμβολο ')', ']', '}' που κλείνει μια παρένθεση ελέγχουμε εάν ταιριάζει με αυτό που είναι στην κορυφή (`top`) της στοίβας.
      - Εάν ΝΑΙ, τότε το αφαιρούμε (`pop`) και συνεχίζουμε με τα υπόλοιπα.
      - Εάν ΟΧΙ, τότε η έκφραση δεν είναι νόμιμη.



Στοίβα [Stack]

- Υλοποίηση στοίβας με διάνυσμα (Array)
  - Απλή υλοποίηση
  - Προσθέτουμε στοιχεία από τη θέση 0 προς το τέλος
  - Μία μεταβλητή διατηρεί το αριθμό της θέσης στην οποία βρίσκεται το κορυφαίο στοιχείο
  - Σταθερό μέγεθος (όριο αποθήκευσης, δέσμευση μνήμης)

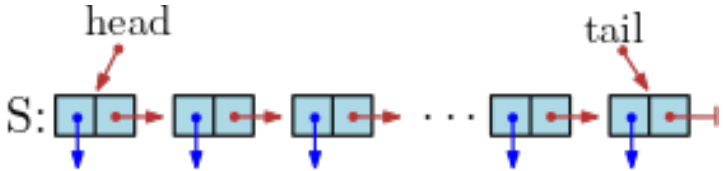



```java
public class ArrayStack<E> implements Stack<E> {
  public static final int CAPACITY=1000;   // default array capacity
  private E[] data;                        // generic array used for storage
  private int t = -1;                      // index of the top element in stack

  public ArrayStack() { this(CAPACITY); }  // constructs stack with default capacity
  public ArrayStack(int capacity) {        // constructs stack with given capacity
    data = (E[]) new Object[capacity];     // safe cast; compiler may give warning
  }

  public int size() { return (t + 1); }
  public boolean isEmpty() { return (t == -1); }

  public void push(E e) throws IllegalStateException {
    if (size() == data.length) throw new IllegalStateException("Stack is full");
    data[++t] = e;                         // increment t before storing new item
  }

  public E top() {
    if (isEmpty()) return null;
    return data[t];
  }

  public E pop() {
    if (isEmpty()) return null;
    E answer = data[t];
    data[t] = null;                        // dereference to help garbage collection
    t--;
    return answer;
  }
```

Στοίβα [Stack]

- Υλοποίηση στοίβας με απλά συνδεδεμένη λίστα (`SinglyLinkedList`)
  - Αξιοποίηση λειτουργιών λίστας
  - Προσθέτουμε στοιχεία στην αρχή της λίστας
  - Το στοιχείο `head` της λίστας είναι στη κορυφή της στοίβας
  - Δυναμικό μέγεθος (δέσμευση μνήμης ίση με το πλήθος των στοιχείων)

```java
public class LinkedStack<E> implements Stack<E> {
  private SinglyLinkedList<E> list = new SinglyLinkedList<>();   // an empty list

  public LinkedStack() { }        // new stack relies on the initially empty list

  public int size() { return list.size(); }

  public boolean isEmpty() { return list.isEmpty(); }

  public void push(E element) { list.addFirst(element); }

  public E top() { return list.first(); }

  public E pop() { return list.removeFirst(); }
```



Στοίβα [Stack]

Σύνοψη

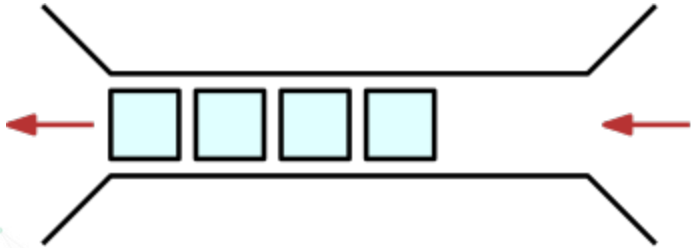
	Υλοποίηση	
Στοίβα	Διάνυσμα	SinglyLinkedList
`push()`	O(1)	O(1)
`pop()`	O(1)	O(1)
`top()`	O(1)	O(1)
`size()`	O(1)	O(1)
`isEmpty()`	O(1)	O(1)
`isFull()`	O(1)	O(1)

Ουρά

[Queue]

Ουρά [Queue]

- Ο ΑΤΔ ουρά αποθηκεύει αντικείμενα αυθαίρετου τύπου
- Οι εισαγωγές και οι διαγραφές ακολουθούν το σχήμα FIFO (First-In First-Out)
- Βασικές μέθοδοι της ουράς:
  - `enqueue(Object)`: Εισάγει στοιχείο στο τέλος της ουράς
  - `Object dequeue()`: Αφαιρεί και επιστρέφει το στοιχείο από την αρχή της ουράς
- Βοηθητικές μέθοδοι της ουράς:
  - `Object first()`: Επιστρέφει το στοιχείο από την αρχή της ουράς χωρίς να το αφαιρέσει
  - `int size()`: Επιστρέφει το πλήθος των στοιχείων που είναι αποθηκευμένα στην ουρά
  - `boolean isEmpty()`: Υποδηλώνει εάν η ουρά είναι άδεια ή όχι



Ουρά [Queue]

- Η διαπροσωπία `Queue`

```
1 public interface Queue<E> {
2
3   void enqueue(E e);
4   E dequeue();
5
6   E first();
7   int size();
8   boolean isEmpty();
9 }
```

Δεν είναι ίδια με τη κλάση της βιβλιοθήκης της Java!
java.util.Queue

Ουρά [Queue]

Παράδειγμα: Χρήση `Queue`

Method	Return value	Περιεχόμενα ουράς
`enqueue(5)`	-	`( 5 )`
`enqueue(3)`	-	`( 5 , 3 )`
`size()`	2	`( 5 , 3)`
`dequeue()`	5	`( 3 )`
`isEmpty()`	`false`	`( 3 )`
`dequeue()`	3	`()`
`isEmpty()`	`true`	`()`
`dequeue()`	`null`	`()`
`enqueue(7)`	-	`( 7 )`
`enqueue(9)`	-	`( 7, 9 )`
`first()`	7	`( 7, 9 )`
`dequeue()`	7	`( 9 )`

Ουρά [Queue]

- Εφαρμογές της ουράς
  - Λίστες αναμονής (Εξυπηρέτηση πελατών)
  - Πρόσβαση σε κοινόχρηστα μέσα (πχ εκτυπωτής)
  - Multiprogramming

Ουρά [Queue]

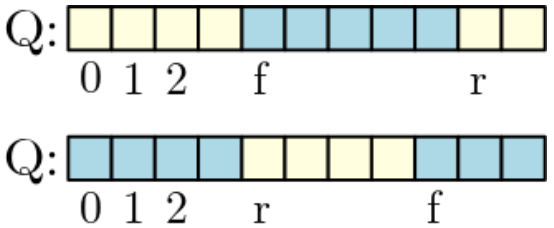
- Υλοποίηση ουράς με κυκλικό-διάνυσμα (circular-array)
  - Σταθερό μέγεθος, έστω `CAPACITY`
  - Μία μεταβλητή διατηρεί τον αριθμό της θέσης του πρώτου στοιχείου της ουράς, έστω `f [front]`
  - Μία μεταβλητή διατηρεί το πλήθος των στοιχείων της ουράς, έστω `sz [size]`

```
1  public class ArrayQueue<E> implements Queue<E> {
2    public static final int CAPACITY = 1000;       // default array capacity
3    private E[] data;                               // generic array used for storage
4    private int f = 0;                              // index of the front element
5    private int sz = 0;                             // current number of elements
6
7    public ArrayQueue() {this(CAPACITY);}          // constructs queue with default capacity
8    public ArrayQueue(int capacity) {              // constructs queue with given capacity
9      data = (E[]) new Object[capacity];           // safe cast; compiler may give warning
10   }
11
12   public int size() { return sz; }
13
14   public boolean isEmpty() { return (sz == 0); }
15
16   public E first() {
17     if (isEmpty()) return null;
18     return data[f];
19   }
```

Ουρά [Queue]

- Υλοποίηση ουράς με κυκλικό-διάνυσμα (circular-array) (*συνέχεια*)
  - Η εισαγωγή στοιχείου γίνεται στην πρώτη διαθέσιμη θέση μετά την `f`, έστω `r [rear]`, η οποία προκύπτει ως: `r = (f + sz) mod CAPACITY`
  - Με παρόμοιο τρόπο υπολογίζεται και η θέση `f` του πρώτου στοιχείο μετά την εξαγωγή στοιχείου

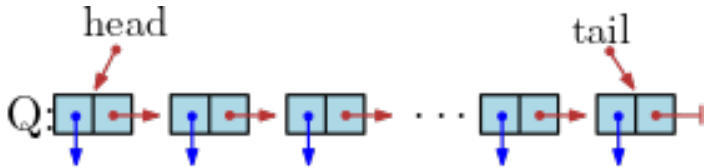
```
21  public void enqueue(E e) throws IllegalStateException {
22    if (sz == data.length) throw new IllegalStateException("Queue is full");
23    int avail = (f + sz) % data.length;   // use modular arithmetic
24    data[avail] = e;
25    sz++;
26  }
27
28  public E dequeue() {
29    if (isEmpty()) return null;
30    E answer = data[f];
31    data[f] = null;              // dereference to help garbage collection
32    f = (f + 1) % data.length;
33    sz--;
34    return answer;
35  }
```



Ουρά [Queue]

- Υλοποίηση ουράς με απλά συνδεδεμένη λίστα (`SinglyLinkedList`)
  - Αξιοποίηση λειτουργιών λίστας
  - Προσθέτουμε στοιχεία στο τέλος της λίστας
  - Το στοιχείο `head` της λίστας είναι η αρχή της ουράς
  - Δυναμικό μέγεθος (δέσμευση μνήμης ίση με το πλήθος των στοιχείων)

```
1  public class LinkedQueue<E> implements Queue<E> {
2    private SinglyLinkedList<E> list = new SinglyLinkedList<>();   // an empty  list
3
4    public LinkedQueue() { }          // new queue relies on the initially empty list
5
6    public int size() { return list.size(); }
7
8    public boolean isEmpty() { return list.isEmpty(); }
9
10   public void enqueue(E element) { list.addLast(element); }
11
12   public E first() { return list.first(); }
13
14   public E dequeue() { return list.removeFirst(); }
```



Ουρά [Queue]

Σύνοψη

	Υλοποίηση		
Ουρά	Διάνυσμα	Κυκλικό διάνυσμα	SinglyLinkedList
`enqueue()`	`O(1)`	`O(1)`	`O(1)`
`dequeue()`	**`O(n)`**	`O(1)`	`O(1)`
`first()`	`O(1)`	`O(1)`	`O(1)`
`size()`	`O(1)`	`O(1)`	`O(1)`
`isEmpty()`	`O(1)`	`O(1)`	`O(1)`
`isFull()`	`O(1)`	`O(1)`	`O(1)`

Κυκλική Ουρά

[Circular Queue]

Κυκλικές Ουρές

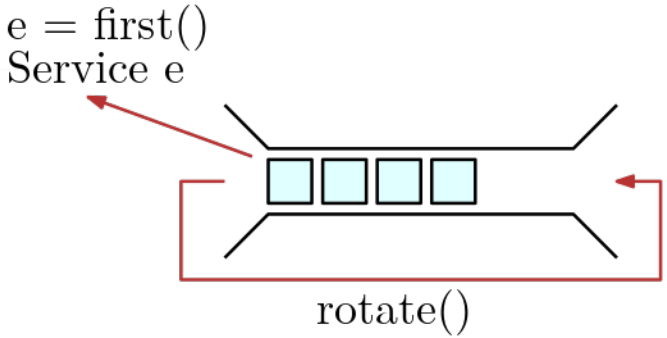
- Επέκταση της απλής ουράς
- Υποστήριξη πράξης `rotate`

```
1 public interface CircularQueue<E> extends Queue<E> {
2   /**
3    * Rotates the front element of the queue to the back of the queue.
4    * This does nothing if the queue is empty.
5    */
6   void rotate();
7 }
```

Η πράξη `rotate` ισοδυναμεί με `enqueue(dequeue())`

Κυκλικές Ουρές

- Εφαρμογή: Round Robin Schedulers
  - Είναι δυνατή η υλοποίηση ενός round robin scheduler χρησιμοποιόντας μια ουρά `q` και εκτελώντας διαδοχικά τα ακόλουθα βήματα:
    - `e = q.first()`
    - `Service element e`
    - `q.rotate()`



Ουρά [Queue]

Σύνοψη

	Υλοποίηση	
Κυκλική Ουρά	Κυκλικό διάνυσμα	CircularlyLinkedList
`enqueue()`	O(1)	O(1)
`dequeue()`	O(1)	O(1)
`first()`	O(1)	O(1)
`size()`	O(1)	O(1)
`isEmpty()`	O(1)	O(1)
`rotate()`	O(1)	O(1)
`isFull()`	O(1)	O(1)