ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS
ΚΕΝΤΡΟ ΕΠΙΜΟΡΦΩΣΗΣ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗΣ

Εφαρμογές Αλγορίθμων σε Πίνακες

Αθ. Ανδρούτσος

Αλγόριθμοι και Πίνακες

- Οι πίνακες είναι μία γραμμική δομή δεδομένων, μία συλλογή στοιχείων
- Είναι μία στατική δομή δεδομένων γιατί από τη στιγμή που θα δημιουργηθεί ένας πίνακες με συγκεκριμένη διάσταση, μετά δεν μπορούμε να αλλάξουμε το μέγεθος (μήκος) του πίνακα
- Οι πίνακες προσφέρονται για την υλοποίηση αλγορίθμων όπως ελάχιστο/μέγιστο στοιχείο πίνακα, ταξινομήσεις, κυκλική μετακίνηση στοιχείων αριστερά (left shift) ή δεξιά (right shift), υλοποίηση μεγάλων ακεραίων και πράξεων επί αυτών, κλπ.

Αλγόριθμοι

- Υπάρχει ειδική επιστημονική περιοχή που μελετά τέτοιου είδους αλγόριθμους σε διάφορες δομές δεδομένων και όχι μόνο σε γραμμικές δομές
- Περισσότερα για δομές δεδομένων και αλγόριθμων μπορείτε να δείτε στο παράρτημα αυτού του κεφαλαίου
- Στη συνέχεια θα δούμε μερικούς αλγόριθμους επί των πινάκων

Circular Rotation (1)


```java
package gr.aueb.cf.ch6;

public class CircularRotationApp {

    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4};

        System.out.println("Initial Array");
        print(arr);
        System.out.println();

        int[] leftRotated = doCircularLeftShiftBy(arr, 2);
        System.out.println("Left Rotated by 2");
        print(leftRotated);
        System.out.println();

        int[] rightRotated = doCircularRightShiftBy(arr, 3);
        System.out.println("Right Rotated by 3");
        print(rightRotated);
    }
```

- Υλοποιεί circular rotation των στοιχείων ενός πίνακα με βάση τις αντίστοιχες μεθλοδους

Circular Rotation (2)


```java
/**
 * Rotates the elements of an array clockwise by an offset.
 *
 * @param arr       the given array of integers.
 * @param offset    the distance to rotate.
 * @return          the rotated array, or null if the
 *                  given array is null.
 */
public static int[] doCircularRightShiftBy(int[] arr, int offset) {
    if (arr == null) return null;
    if (offset < 0) return null;
    int[] rotated = new int[arr.length];

    for (int i = 0; i < arr.length; i++) {
        rotated[(i + offset) % arr.length] = arr[i];
    }

    return rotated;
}
```

- Πρώτα ελέγχει αν τα στοιχεία εισόδου είναι valid
- Στη συνέχεια δημιουργεί ένα νέο πίνακα (rotated) που θα είναι ο destination array
- Στη συνέχεια, ο destination array στο δικό του στοιχείο (i + offset) % arr.length εισάγει το στοιχείο i του source array
- Το % arr.length υλοποιεί το circular rotation

Circular Rotation (3)


```java
/**
 * Rotates the elements of an array counterclockwise by an offset.
 *
 * @param arr       the given array of integers.
 * @param offset    the distance to rotate.
 * @return          the rotated array, or null if the
 *                  given array is null.
 */
public static int[] doCircularLeftShiftBy(int[] arr, int offset) {
    if (arr == null) return null;
    if (offset < 0) return null;
    int[] rotated = new int[arr.length];

    for (int i = 0; i < arr.length; i++) {
        rotated[i] = arr[(i + offset) % arr.length];
    }
    return rotated;
}

public static void print(int[] arr) {
    if (arr == null) return;
    for (int item : arr)
        System.out.print(item + " ");
}
}
```

- Όπως πριν, μόνο που τώρα υλοποιείται το αντίθετο circular rotation
- Πρώτα ελέγχει αν τα στοιχεία εισόδου είναι valid
- Στη συνέχεια δημιουργεί ένα νέο πίνακα (rotated) που θα είναι ο destination array
- Στη συνέχεια, ο destination array στο δικό του στοιχείο (i) εισάγει το στοιχείο (i + offset) % arr.length του source array
- Το % arr.length υλοποιεί το circular rotation

Array Integers

- Έστω ότι αναπαριστούμε ένα μεγάλο integer (όπως ο BigInteger) με πίνακα
- Για παράδειγμα έχουμε τον 1918 ως:

1	9	1	8

- Θα θέλαμε να κάνουμε πράξεις όπως **να προσθέσουμε την μονάδα** ή **να προσθέσουμε δύο ακεραίους**

Add One to array int


```java
52  @   public static int[] addOne(int[] arr) {
53          int[] arrOut;
54          int currentSum;
55          int carry = 1;
56
57          if (arr == null) throw new IllegalArgumentException();
58
59          arrOut = new int[arr.length + 1];
60
61          for (int i = arr.length - 1; i >= 0; i--) {
62              currentSum = arr[i] + carry;
63
64              arrOut[i + 1] = currentSum % 10;
65              carry = currentSum / 10;
66          }
67
68          arrOut[0] = (carry == 1) ? 1 : 0;
69          return arrOut;
70      }
```

- Η πρόσθεση της μονάδας ξεκινάει από το τελευταίο ψηφίο. Αλλά αν το τελευταίο ψηφίο είναι το 9, τότε θα έχουμε και κρατούμενο (carry) το 1 και μπορεί και ο αριθμός που θα προκύψει να έχει ένα ψηφίο επιπλέον (για παράδειγμα 9999 + 1 = 10000)
- Θα πρέπει λοιπόν για κάθε ψηφίο του array (μία for που ξεκινάει από το τελευταίο ψηφίο) να προσθέτουμε το ψηφίο και το κρατούμενο και να αποθηκεύουμε σε ένα πίνακα εξόδου (arrOut)
- Αρχικά αρχικοποιούμε το κρατούμενο (carry) να είναι 1, αφού στην αρχή προσθέτουμε τη μονάδα)

Main


```java
public static void main(String[] args) {
    int[] arr = {9, 9, 9};
    int[] arrOut;
    arrOut = addOne(arr);

    for (int item : arrOut) {
        System.out.print(item + " ");
    }
}
```

- Το αναμενόμενο αποτέλεσμα θα είναι το 1000

Add Two Bigints with array


```java
public static int[] addTwoInts(int[] arr1, int[] arr2) {
    int[] arrOut;
    int currentSum;
    int carry = 0;

    if ((arr1 == null) || (arr2 == null) || (arr1.length != arr2.length)) {
        throw new IllegalArgumentException();
    }

    arrOut = new int[arr1.length + 1];
    for (int i = arr1.length - 1; i >= 0; i--) {
        currentSum = arr1[i] + arr2[i] + carry;

        arrOut[i + 1] = currentSum % 10;
        carry = currentSum / 10;
    }

    arrOut[0] = carry;
    return arrOut;
}
```

- Η ιδέα είναι η ίδια όπως πριν μόνο που τώρα το άθροισμα είναι τα δύο αντίστοιχα ψηφία συν το κρατούμενο
- **Αρχικά θεωρούμε το carry ότι είναι 0** και μετά πάλι παίρνουμε το νέο κρατούμενο με div ενώ το ψηφίο του αποτελέσματος με mod

Binary to decimal


```java
package gr.aueb.cf.ch6;

public class BinaryToDecimalApp {

    public static void main(String[] args) {
        int[] vector = {1, 0, 0, 0, 1, 1, 1, 0};    // 142
        int decimal = binaryToDecimal(vector);
        System.out.println("Value: " + decimal);
    }
```

- Υποθέτουμε ότι ο αριθμός αποθηκεύεται σε big endian μορφή, δηλαδή το LSB (Least Significant Bit) βρίσκεται δεξιά (τελευταίο)
- Μετατρέπουμε μία δυαδική ακολουθία bits, π.χ. $a_7a_6a_5a_4a_3a_2a_1a_0$ στο δεκαδικό σύστημα ως $a_7*2^7 + a_6*2^6 + a_5*2^5 + a_4*2^4 + a_3*2^3 + a_2*2^2 + a_1*2^1 + a_0*2^0$

Binary to decimal - Big Endian


```java
/**
 * Converts a binary vector to decimal.
 * Assume that the binary vector is given in
 * big-endian form --MSB (Most Significant Bit) first,
 * and LSB (Least Significant Bit) last--.
 *
 * @param binaryVector  The source vector
 * @return              the decimal representation of the vector.
 */
public static int binaryBigEndianToDecimal(int[] binaryVector) {
    int decimal = 0;
    int n;

    if (binaryVector == null) {
        throw new IllegalArgumentException();
    }

    n = binaryVector.length;
    for (int i = n - 1; i >= 0; i--) {
        decimal = decimal + binaryVector[i] * (int) Math.pow(2, n - 1 - i);
    }

    return decimal;
}
```

- Στην Big Endian μορφή αναπαράστασης binary vectors, το ελάχιστο σημαντικό bit (LSB) βρίσκεται δεξιά και το πιο σημαντικό (MSB) βρίσκεται αριστερά
- Αντίστοιχα εκτελείται και η for η οποία υπολογίζει το δεκαδικό άθροισμα των bit του binary vector ως άθροισμα των δυνάμεων του δύο ανάλογα με τη θέση του bit

Binary to decimal - Little Endian


```java
/**
 * Converts a binary vector to decimal.
 * Assume that the binary vector is given in
 * little-endian form --LSB (Least Significant Bit) first,
 * and MSB (Most Significant Bit) last--.
 *
 * @param binaryVector  The source vector
 * @return              the decimal representation of the vector.
 */
public static int binaryLittleEndianToDecimal(int[] binaryVector) {
    int decimal = 0;
    int n;

    if (binaryVector == null) {
        throw new IllegalArgumentException();
    }

    n = binaryVector.length;
    for (int i = 0; i < n; i++) {
        decimal = decimal + binaryVector[i] * (int) Math.pow(2, i);
    }

    return decimal;
}
}
```

- Στην Little Endian μορφή αναπαράστασης binary vectors, το ελάχιστο σημαντικό bit (LSB) βρίσκεται αριστερά στην αρχή και το πιο σημαντικό (MSB) βρίσκεται δεξιά στο τέλος του vector
- Αντίστοιχα εκτελείται και η for η οποία εδώ υπολογίζει το δεκαδικό άθροισμα των bit του binary vector ως άθροισμα των δυνάμεων του δύο ανάλογα με τη θέση του bit, όπου εδώ η 1η θέση (LSB) είναι αριστερά δηλαδή στη θέση 0 του πίνακα

Bitwise Operators

- Η Java παρέχει τους bitwise operators που κάνουν πράξεις επί των ψηφίων μίας δυαδικής ακολουθίας ψηφίων
- Οι bitwise operators είναι οι:

&, |	Λογικό ΚΑΙ, Λογικό Ή
^	XOR (που επαληθεύεται μόνο όταν τα δύο bits έχουν διαφορετικές τιμές 0, 1 ή 1, 0
~	Συμπλήρωμα ως προς ένα (One's compliment)
<<, >>, >>>	Shift left, shift right (σέβεται το πρόσημο), shift right (left padding με zeros)

Εφαρμογές με bitwise operators

- Θα προσομοιώσουμε στα επόμενα κάποιους bitwise operators ώστε να επενεργούν επί δυαδικών ακολουθιών bits που απεικονίζονται με πίνακες ακεραίων ή σε boolean μόνο με πράξεις &&, ||, !

One Bit XOR


```java
/**
 * Implements the logical XOR between two booleans.
 *
 * @param b1      the first boolean.
 * @param b2      the second boolean.
 * @return        the logical XOR output.
 */
public static boolean XOR(boolean b1, boolean b2) {
    return (b1 && !b2) || (!b1 && b2);
}
```

- Η λογική της XOR (eXclusive OR) είναι πως οι δύο boolean θα πρέπει να είναι διαφορετικοί (0/1 ή 1/0 – true/false ή false/true)

One’s Compliment


```java
/**
 * Converts a binary number to one's compliment format.
 * That is 1 -> 0 and 0 -> 1. That is the ~ bitwise operator.
 *
 * @param vector    the source binary number.
 * @return          the one's compliment.
 */
public static int[] ComplimentByOne(int[] vector) {
    int[] binOut;

    if (vector == null) throw new IllegalArgumentException();

    binOut = new int[vector.length];
    for (int i = 0; i < vector.length; i++) {
        binOut[i] = (vector[i] == 0) ? 1 : 0;
    }

    return binOut;
}
```

- Το **συμπλήρωμα ως προς ένα (One’s Compliment)** υλοποιείται αντιστρέφοντας (flipped) την τιμή του κάθε bit, από 0 σε 1 και από 1 σε 0
- Πρόκειται για τον ~ bitwise operator

Συνδυασμοί αριθμών (1)

- Έστω ότι θέλουμε να πάρουμε τους συνδυασμούς όλων των τριάδων από 5 αριθμούς. Έστω ότι έχουμε ένα ταξινομημένο πίνακα [1, 2, 3, 4, 5] και θέλουμε να πάρουμε: *1, 2, 3 – 1, 2, 4 – 1, 2, 5 – 1, 3, 4 – 1, 3, 5 – 1, 4, 5* και *2, 3, 4 – 2, 3, 5 – 2, 4, 5* και *3, 4, 5*
- Εφόσον έχουμε 5 αριθμούς και θέλουμε 3άδες, τότε ο πρώτος αριθμός κάθε τριάδας δεν μπορεί να είναι μεγαλύτερος από το 3
- Γενικά αν έχουμε n αριθμούς και θέλουμε συνδυασμούς των m, τότε ο πρώτος αριθμός θα είναι από 0 έως n-m θέση του αρχικού πίνακα, δηλαδή στην περίπτωσή μας από 0 έως 5-3=2. Η θέση 2 του αρχικού πίνακα είναι το 3
- Με την ίδια λογική στην $2^{η}$ θέση του τελικού πίνακα θα έχουμε τους αριθμούς των θέσεων 1 – 3 του αρχικού πίνακα, πάμε μία θέση δεξιά δηλαδή σε σχέση με την $1^{η}$ θέση, κ.ο.κ.

Συνδυασμοί αριθμών

- Για να πάρουμε τους συνδυασμούς m από n αριθμούς, όπου οι n αριθμοί είναι αποθηκευμένοι σε μία ArrayList<Integer> θα πρέπει να έχουμε m nested for όπου το i του 1ου for θα ξεκινάει από 0 και θα φτάνει μέχρι n – m (δηλ. ArrayList.size() - m), το j του 2ου for θα ξεκινάει από το i + 1 και θα φτάνει μέχρι size() – m + 1, κλπ. δηλαδή κάθε for θα αναφέρεται σε ένα διάστημα (range) αριθμών του πίνακα που θα μετακινείται σε κάθε for μία θέση δεξιά
- Η ArrayList είναι μία κλάση που γίνεται backed από πίνακα αλλά μπορεί να μεγαλώνει και να μικραίνει το μέγεθός της, ενώ ο πίνακας δεν μπορεί

Συνδυασμοί Τεσσάρων (1)


```java
package gr.aueb.cf.chapter6;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintStream;
import java.util.ArrayList;
import java.util.Scanner;

public class CombinationsFourDemo {

    public static void main(String[] args) throws FileNotFoundException {
        final int THRESHOLD = 4;
        File inFile = new File("C:/Users/a8ana/OneDrive/CodingFactory-REBOOT/Projects/JAVA/numbers.txt");
        File outFile = new File("C:/Users/a8ana/OneDrive/CodingFactory-REBOOT/Projects/JAVA/combinations.txt");
        Scanner in = new Scanner(inFile);
        PrintStream ps = new PrintStream(outFile);
        int n = 4;
        int[] row = new int[4];



        ArrayList<Integer> numbers = new ArrayList<>();

        while (in.hasNextInt()) {
            numbers.add(in.nextInt());
        }
```

- Διαβάζουμε από το αρχείο εισόδου (όσο έχει next, η hasNextInt() είναι state-checking method) και εισάγουμε στην ArrayList με την μέθοδο .add() που εισάγει στην επόμενη ελεύθερη θέση
- Η File είναι μία κλάση της Java που αναπαριστά αρχεία και μετά μπορούμε με Scanner να διαβάσουμε από το αρχείο

Συνδυασμοί Τεσσάρων (2)


```
27        for (int i = 0; i <= numbers.size() - n; i++) {
28            for (int j = i + 1; j <= numbers.size() - n + 1; j++) {
29                for (int k = j + 1; k <= numbers.size() - n + 2; k++) {
30                    for (int m = k + 1; m < numbers.size(); m++) {
31                        row[0] = numbers.get(i);
32                        row[1] = numbers.get(j);
33                        row[2] = numbers.get(k);
34                        row[3] = numbers.get(m);
35
36                        if (!isEven(row, THRESHOLD)) {
37                            ps.printf("%d\t%d\t%d\t%d\n", numbers.get(i), numbers.get(j), numbers.get(k), numbers.get(m));
38                        }
39                    }
40                }
41            }
42        }
43    }
```

- Η κάθε for αναφέρεται σε ένα αριθμό της τετράδας, η 1η for στον 1ο αριθμό, η 2η for στον 2ο αριθμό κλπ.
- Πριν εκτυπώσει στο αρχείο, ελέγχει αν η τετράδα δεν έχει πάνω από THRESHOLD άρτιους (π.χ. πάνω από 3 άρτιους)

Άρτιοι


```
45      /**
46       * Checks if a 6-number array contains more than
47       * a certain amount of evens.
48       *
49       * @param row       the source array of ints.
50       * @return          true, if the count of evens
51       *                  id greater than the threshold.
52       */
53  @   public static boolean isEven(int[] row, int threshold) {
54          int count = 0;
55
56          for (int num : row) {
57              if (num % 2 == 0) count++;
58          }
59
60          return (count > threshold);
61      }
62  }
```

- Ελέγχουμε αν μία τετράδα έχει ζυγούς και πόσους
- Αν έχει πάνω από threshold άρτιους επιστρέφουμε true

Άθροισμα 15 (1)


```java
package gr.aueb.cf.ch6;

public class Array15App {

    public static void main(String[] args) {
        int[][] arr = {{2, 7, 6},
                       {9, 5, 1},
                       {4, 3, 8}
        };

        System.out.printf("It is%s arr15\n", (isArr15(arr) ? "" : " not"));
    }

    public static boolean isArr15(int[][] arr) {
        int hCount = 0;
        int vCount = 0;
        int diagonal1Sum = 0;
        int diagonal2Sum = 0;
        int[] vSum = new int[3];
        int[] hSum = new int[3];
        boolean isDiagonal1 = false;
        boolean isDiagonal2 = false;

        if (arr == null) throw new RuntimeException();
```

- Το πρόγραμμα ελέγχει αν ένας πίνακας έχει άθροισμα 15, σε όλες τις πλευρές του και τις διαγώνιους

Άθροισμα 15 (2)


```
26      for (int i = 0; i < arr.length; i++) {
27          for (int j = 0; j < arr[i].length; j++) {
28              hSum[i] += arr[i][j];
29          }
30
31          diagonal1Sum += arr[i][i];
32          diagonal2Sum += arr[i][arr.length-i-1];
33
34          vSum[0] += arr[i][0];
35          vSum[1] += arr[i][1];
36          vSum[2] += arr[i][2];
37      }
38
39      for (int num : hSum) {
40          if (num == 15) hCount++;
41      }
42
43      for (int num : vSum) {
44          if (num == 15) vCount++;
45      }
46
47      if (diagonal1Sum == 15) isDiagonal1 = true;
48      if (diagonal2Sum == 15) isDiagonal2 = true;
49
```

- Η 1η for διατρέχει τις γραμμές του πίνακα και υπολογίζει διαγώνια και κάθετα αθροίσματα, ενώ η nested for διατρέχει τα στοιχεία των γραμμών και υπολογίζει τα οριζόντια αθροίσματα
- Στη συνέχεια ελέγχουμε αν τα αθροίσματα γραμμών και στηλών είναι τρία, όσα οι γραμμές/στήλες
- Καθώς και αν τα διαγώνια αθροίσματα είναι 15

Άθροισμα 15 (3)


```
50          return (isDiagonal1 && isDiagonal2 && (hCount == 3) && (vCount == 3));
51      }
52  }
```

- Στη συνέχεια επιστρέφουμε το λογικό AND των συνθηκών που ελέγχουν αν όλα τα διαγώνια, οριζόντια και κάθετα αθροίσματα, είναι 15

Άθροισμα 15 (4)


```java
package gr.aueb.cf.chapter6;

/**
 * Ελέγχει αν ένας πίνακας έχει άθροισμα οριζόντιο,
 * κάθετο και διαγωνίων == 15.
 */
public class Grid15Demo {

    public static void main(String[] args) {
        int[][] arr = { {2, 7, 6},
                        {9, 5, 1},
                        {4, 3, 8}};

        System.out.printf("It is%s arr15", (isArr15(arr)) ? "" : " not");
    }
```

- Το παραπάνω αναμένουμε να δώσει ότι ο πίνακας είναι arr15

Στοίβα (Stack) – Ουρά (Queue)

- Δύο σημαντικές δομές δεδομένων είναι οι **στοίβες (stack) και οι ουρές (queues)**
- Η στοίβα υλοποιεί τη λογική **LIFO** (Last In First Out)
- Η ουρά υλοποιεί τη λογική **FIFO** (First In First Out)
- Θα δούμε στα επόμενα πως μπορούμε να χρησιμοποιήσουμε πίνακες ως αποθηκευτικό χώρο για να υλοποιήσουμε τις παραπάνω δομές δεδομένων stack και queue

Stack Implementation (1)


```java
package gr.aueb.cf.chapter6;

public class StackDemo {

    static int[] stack = new int[50];
    static int top = -1;

    public static void main(String[] args) {
        int num;

        push(1);
        push(2);
        push(3);
        push(4);
        push(5);
        push(6);

        printStack();
        num = pop();
        System.out.printf("Num %d was popped\n", num);
        System.out.println("Contents of the stack");
        printStack();
    }
```

- Ξεκινάμε με ένα πίνακα έστω 50 θέσεων και ένα ακέραιο top που λειτουργεί ως δείκτης που δείχνει στην κορυφή της στοίβας (αρχικά -1)
- Η main απλά καλεί τις βασικές μεθόδους της Στοίβας, push() για να εισάγει στο τέλος της στοίβας και pop() για να εξάγει από το τέλος της στοίβας
- Θα δούμε στα επόμενα την υλοποίηση των βασικών καθώς και βοηθητικών μεθόδων της στοίβας

Stack Push


```
23      /**
24       * Inserts a num on top of stack.
25       *
26       * @param num   the input num.
27       */
28      public static void push(int num) {
29          if (isFull()) {
30              throw new RuntimeException("Stack is full");
31          }
32          stack[++top] = num;     // same as:  top++; stack[top] = num;
33      }
```

- Εισάγει ένα αριθμό στο τέλος του πίνακα (της στοίβας)
- Πρώτα αυξάνει το top κατά ένα και μετά εισάγει

Stack Pop


```
35      /**
36       * Gets (and essentially removes)  a num from
37       * the top of the stack.
38       *
39       * @return      the num from the top of the stack.
40       */
41      public static int pop() {
42          if (isEmpty()) {
43              throw new RuntimeException("Stack is empty");
44          }
45          return stack[top--];
46      }
```

- Εξάγει (και διαγράφει από τη στοίβα) το τελευταίο στοιχείο της στοίβας
- Πρώτα εξάγει και μετά μειώνει το top κατά ένα

isFull & isEmpty


```
53      /**
54       * Returns true if the stack is full.
55       *
56       * @return      true if the stack is full,
57       *              false otherwise.
58       */
59      public static boolean isFull() {
60          return (top == stack.length - 1);
61      }
62
63      /**
64       * Returns true if the stack is empty.
65       *
66       * @return      true if the stack is empty,
67       *              false otherwise.
68       */
69      public static boolean isEmpty() {
70          return (top == -1);
71      }
```

- Βοηθητικές μέθοδοι που ελέγχουν αν η στοίβα είναι γεμάτη ή άδεια

Prints stack elements


```
73      /**
74       * Prints the elements of the stack.
75       */
76      public static void printStack() {
77          if (isEmpty()) {
78              throw new RuntimeException("Empty Stack");
79          }
80
81          for (int i = top; i >= 0; i--) {
82              System.out.println(stack[i]);
83          }
84          System.out.println();
85      }
86  }
```

- Εκτυπώνει σε αντίστροφη σειρά (από πάνω προς τα κάτω) τα στοιχεία της στοίβας

Queue


```
1  package gr.aueb.cf.chapter6;
2
3  import java.util.Arrays;
4
5  public class QueueDemo {
6
7      static int[] queue = new int[10];
8      static int top = -1;
9
10     public static void main(String[] args) {
11         int num;
12
13         enQueue(1);
14         enQueue(2);
15         enQueue(3);
16         printQueue();
17
18         num = deQueue();
19         printQueue();
20     }
```

- Η ουρά μπορεί να αναπαρασταθεί με ένα πίνακας έστω 10 θέσεων (ουρά 10 θέσεων)
- Και με ένα δείκτη που δείχνει στο τέλος της ουράς (αρχικά -1)

enQueue (Εισαγωγή στο τέλος)


```
22      /**
23       * Inserts a num at the end of the queue.
24       *
25       * @param val   the num to be inserted.
26       */
27      public static void enQueue(int val) {
28          if (isFull()) {
29              throw new RuntimeException("Queue is full");
30          }
31          queue[++top] = val;
32      }
```

- Εισάγουμε στο τέλος της ουράς. Πρώτα αυξάνουμε κατά 1 το top και μετά εκχωρούμε

deQueue (Εξαγωγή από την αρχή)


```
34      /**
35       * Returns (and essentially deletes) the first
36       * queue element.
37       *
38       * @return  the element t the front of teh queue.
39       */
40      public static int deQueue() {
41          int num = 0;
42
43          if (isEmpty()) throw new RuntimeException("Queue is empty");
44          num = queue[0];
45
46          // Arrays.copyOfRange returns an array with (to - from)
47          // elements.  If 'to' which is not inclusive is greater
48          // than length, then the returned array is right padded
49          // with zeroes.
50          queue = Arrays.copyOfRange(queue, 1, queue.length + 1);
51          top--;
52          return num;
53      }
```

- Εξάγουμε από την αρχή
- Για να μην έχουμε οπή (hole) κάνουμε shift μία θέση αριστερά όλα τα περιεχόμενα του πίνακα και μειώνουμε τον top κατά 1

isFull & isEmpty


```
57      /**
58       * Returns true if the queue is full.
59       *
60       * @return      true if the queue is full,
61       *              false otherwise.
62       */
63      public static boolean isFull() {
64          return (top == queue.length - 1);
65      }
66
67      /**
68       * Returns true if the queue is empty.
69       *
70       * @return      true if the queue is empty,
71       *              false otherwise.
72       */
73      public static boolean isEmpty() {
74          return (top == -1);
75      }
```

- Βοηθητικές μέθοδοι που ελέγχουν αν η ουρά είναι γεμάτη ή άδεια

Print Queue elements


```
77      /**
78       * Prints the elements of the queue.
79       */
80      public static void printQueue() {
81          if (isEmpty()) {
82              throw new RuntimeException("Empty Queue");
83          }
84
85          for (int i = 0; i <= top; i++) {
86              System.out.print(queue[i] + " ");
87          }
88
89          System.out.println();
90      }
91  }
```

- Εκτυπώνει ξεκινώντας από την αρχή της ουράς και φτάνει μέχρι όχι το τέλος του πίνακα αλλά μέχρι το στοιχείο top

Άσκηση 1

- Έστω ένας **ταξινομημένος** πίνακας με επαναλαμβανόμενα στοιχεία. Γράψτε μία μέθοδο *int[] getLowAndHighIndexOf(int[] arr, int key)* που να υπολογίζει και να επιστρέφει τα low και high index ενός πίνακα arr, για ένα ακέραιο key που λαμβάνει ως παράμετρο.

- Γράψτε και μία main() που να βρίσκει το low και high index για τον πίνακα {0, 1, 4, 4, 4, 6, 7, 8, 8, 8, 8, 8}. Για παράδειγμα, αν δώσουμε ως τιμή το 8, θα πρέπει να επιστρέφει {7, 11}.

- **Hint**. Ελέγξτε αν το key περιέχεται στον πίνακα και σε ποια θέση. Αν ναι, τότε από τη θέση αυτή μετρήστε τα στοιχεία όσο υπάρχουν στοιχεία με ίδια τιμή και μέχρι να βρείτε το τέλος του πίνακα.

Άσκηση 2

- Έστω ένας δισδιάστατος πίνακας που περιέχει τα στοιχεία άφιξης και αναχώρησης αυτοκινήτων σε μορφή *arr[][] = {{1012, 1136}, {1317, 1417}, {1015, 1020}}* Αναπτύξτε μία εφαρμογή που διαβάζει να εκτυπώνει τον μέγιστο αριθμό αυτοκινήτων που είναι σταθμευμένα το ίδιο χρονικό διάστημα. Για παράδειγμα στον παραπάνω πίνακα το αποτέλεσμα θα πρέπει να είναι: 2. Το 1ο αυτοκίνητο αφίχθη στις 10:12 και αναχώρησε στις 11:36, το 3ο αυτοκίνητο αφίχθη στις 10:15 και αναχώρησε στις 10:20. Επομένως, το 1ο και το 3ο αυτοκίνητο ήταν παρόντα το ίδιο χρονικό διάστημα.

- **Hint**. Με βάση τον αρχικό πίνακα, δημιουργήστε ένα δυσδιάστατο πίνακα που σε κάθε γραμμή θα περιέχει δύο πεδία int. Στο πρώτο πεδίο θα εισάγεται η ώρα άφιξης ή αναχώρησης από τον αρχικό πίνακα και στο 2ο πεδίο θα εισάγεται ο αριθμός 1 αν πρόκειται για άφιξη και 0 αν πρόκειται για αναχώρηση.

- Ταξινομήστε τον πίνακα σε αύξουσα σειρά με βάση την ώρα. Στη συνέχεια υπολογίστε το μέγιστο αριθμό αυτοκινήτων που είναι σταθμευμένα το ίδιο χρονικό διάστημα με ένα πέρασμα του πίνακα.