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

Εφαρμογές και παραδείγματα στον Δομημένο Προγραμματισμό

Αθ. Ανδρούτσος

Armstrong Numbers (1)

- Οι αριθμοί Armstrong είναι ένα είδος ειδικών αριθμών στον τομέα των μαθηματικών. Ένας αριθμός Armstrong για μια δεδομένη βάση είναι **ένας αριθμός που είναι ίσος με το άθροισμα των ψηφίων του, όπου κάθε ψηφίο έχει υψωθεί στη δύναμη του αριθμού των ψηφίων**.
- Για παράδειγμα, στο δεκαδικό σύστημα, ένας αριθμός Armstrong με τρία ψηφία θα ήταν:
- $abc=a^3+b^3+c^3$
- Εδώ, *a*, *b* και *c* είναι τα τρία ψηφία του αριθμού.
- Παράδειγμα: Ο αριθμός 153 είναι ένα παράδειγμα αριθμού Armstrong στο δεκαδικό σύστημα, γιατί:
- $1^3+5^3+3^3=1+125+27=153$

Armstrong Numbers (2)


```java
package testbed.review;

import ...

/**
 * Βρίσκει αν το άθροισμα των δυνάμεων κάθε ψηφίου εις τη (πλήθος ψηφίων)
 * είναι ίσο με τον ίδιο τον αριθμό. Για παράδειγμα, ο αριθμός 153 = 1^3 + 5^3 + 3^3.
 * Επομένως, το 153 είναι Armstrong number. Επίσης, τα 370, 371, 407.
 */
public class ArmstrongApp {

    public static void main(String[] args) {
        final Scanner in = new Scanner(System.in);
        final List<Integer> digits = new ArrayList<>();
        int inputNumber = 0;
        int digitsCount = 0;
        int num = 0;
        int sum = 0;
        int digit = 0;
        boolean isArmstrong = false;
```

- Δεδομένου ότι δεν γνωρίζουμε πόσα ψηφία θα δώσει ο χρήστης, χρησιμοποιούμε μία List<> που υλοποιείται ως ArrayList<> για να αποθηκεύουμε τα ψηφία του αριθμού

Armstrong Numbers (3)


```
25          System.out.println("Please insert a number (int)");
26          inputNumber = in.nextInt();
27
28          num = inputNumber;
29
30          // Find digits count
31          do {
32              digitsCount++;
33              digit = num % 10;
34              digits.add(digit);
35              num = num / 10;
36          } while (num != 0);
37
38          // Find the sum
39          for (int item : digits) {
40              sum += Math.pow(item, digitsCount);
41          }
42
43          isArmstrong = (sum == inputNumber);
44
45          System.out.println("Number is: " + inputNumber);
46          System.out.println("Sum is: " + sum);
47          System.out.printf("%d is Armstrong: %s", inputNumber,  (isArmstrong) ? "YES" : "NO");
48      }
49  }
```

- Διαιρούμε τον αριθμό με το 10 όσο δεν είναι 0, και παίρνουμε ένα-ένα τα ψηφία του, με num % 10
- Κρατάμε επίσης το άθροισμα των ψηφίων
- Στη συνέχεια βρίσκουμε το άθροισμα των δυνάμεων και συγκρίνουμε

UpperLowerCaseApp (1)


```java
package testbed.review;

import java.util.Scanner;

/**
 * Μετατρέπει σε κεφαλαία-πεζά εναλλάξ με το 1ο κεφαλαίο.
 * Παράδειγμα: "athanassios" -> "AtHaNaSsIoS"
 */
public class UpperLowerCaseApp {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        StringBuilder sb = new StringBuilder();
        String sentence;
        char ch;

        System.out.println("Please insert a string");
        sentence = in.nextLine();
        for (int i = 0; i < sentence.length(); i++) {
            ch = sentence.charAt(i);
            if (Character.isLetter(ch)) {
                ch = ((i % 2) == 0) ? Character.toUpperCase(ch)
                        : Character.toLowerCase(ch);
                sb.append(ch);
            }
        }

        for (int i = 0; i < sb.length(); i++) {
            System.out.print(sb.charAt(i));
        }
    }
}
```

- Χρησιμοποιούμε StringBuilder για να αποθηκεύσουμε το string εξόδου δεδομένου ότι δεν ξέρουμε εκ των προτέρων το μήκος του string εισόδου που δίνει ο χρήστης και αν είχαμε string θα είχαμε πολλά concat που θα ήταν inefficient.
- Ο StringBuilder όπως γνωρίζουμε είναι mutable

UpperLowerCaseApp (2)


```
24          for (int i = 0; i < sb.length(); i++) {
25              System.out.print(sb.charAt(i));
26          }
27      }
28  }
```

- Μετατρέπει σε κεφαλαία – πεζά εναλλάξ.

Κρυπτογράφηση

- Μία από τις πρώτες εφαρμογές των Η/Υ ήταν η κρυπτογράφηση
- Ένας απλός τρόπος κρυπτογράφησης είναι η κωδικοποίηση κάθε χαρακτήρα με ένα ακέραιο με βάση ένα **κλειδί κρυπτογράφησης**
- Μία **μέθοδος κρυπτογράφησης** περιγράφεται στη συνέχεια

Challenge – Κρυπτογράφηση (1)

- Κωδικοποίησε **τον 1ο χαρακτήρα του μηνύματος με την ακέραια τιμή που αντιστοιχεί σε αυτόν** (από τον κώδικα ASCII)
- Κωδικοποίησε του επόμενους χαρακτήρες: (α) **προσθέτοντας** την ακέραια ASCII τιμή του καθένα από αυτούς με τον κωδικό του προηγούμενού του, (β) παίρνοντας **το υπόλοιπο της διαίρεσης του αθροίσματος** αυτού διά μία σταθερά
- Η σταθερά ονομάζεται **κλειδί (key)** κρυπτογράφησης και (υποτίθεται πως) είναι μυστική

Challenge – Κρυπτογράφηση (2)

- Υποθέτουμε πως τα μηνύματα τελειώνουν με τον χαρακτήρα #
- Γράψτε ένα πρόγραμμα java που να υλοποιεί τον αλγόριθμο κρυπτογράφησης έτσι ώστε το κωδικοποιημένο μήνυμα που προκύπτει να είναι μία ακολουθία ακεραίων που τελειώνει με -1
- Γράψτε και τον αλγόριθμο αποκρυπτογράφησης που λαμβάνει ως είσοδο μία ακολουθία ακεραίων που τελειώνει με -1 και υπολογίζει το αρχικό μήνυμα

Cryptography (1)


```java
package testbed.review;

import java.util.ArrayList;

/**
 * Κωδικοποίησε τον 1ο χαρακτήρα του μηνύματος με την ακέραια τιμή που αντιστοιχεί σε αυτόν
 * (από τον κώδικα ASCII). Κωδικοποίησε του επόμενους χαρακτήρες: (α) προσθέτοντας την ακέραια
 * ASCII τιμή του καθένα από αυτούς με τον κωδικό του προηγούμενού του,
 * (β) παίρνοντας το υπόλοιπο της διαίρεσης του αθροίσματος αυτού διά μία σταθερά.
 * Υποθέτουμε πως τα μηνύματα τελειώνουν με τον χαρακτήρα #
 * Γράψτε ένα πρόγραμμα java που να υλοποιεί τον αλγόριθμο κρυπτογράφησης έτσι ώστε το
 * κωδικοποιημένο μήνυμα που προκύπτει να είναι μία ακολουθία ακεραίων που τελειώνει με -1
 * Γράψτε και τον αλγόριθμο αποκρυπτογράφησης που λαμβάνει ως είσοδο μία ακολουθία ακεραίων
 * που τελειώνει με -1 και υπολογίζει το αρχικό μήνυμα
 */
```

a8ana *

Cryptography (2)


```
18      public static void main(String[] args) {
19          // The key should be large enough to give deciphers 0 to 127
20          final int KEY = 128;
21          String s = "Apollo 17 was the final mission of NASA's Apollo program.#";
22          String s1 = encrypt(s, KEY).toString();
23          System.out.println(s1);
24
25          String s2 = decrypt(encrypt(s, KEY), KEY).toString();
26          System.out.println(s2);
27      }
```

Cryptography (3)


```
29 @    public static ArrayList<Integer> encrypt(String s, int key) {
30          ArrayList<Integer> encrypted = new ArrayList<>();
31          char ch;
32          int i;
33
34          int prev = cipher(s.charAt(0), -1, key);
35          encrypted.add(prev);
36
37          i = 1;
38          while ((ch = s.charAt(i)) != '#') {
39              encrypted.add(cipher(ch, prev, key));
40              prev = cipher(ch, prev, key);
41              i++;
42          }
43          encrypted.add(-1);
44          return encrypted;
45      }
```

- Ακολουθούμε τη λογική που περιγράφεται στην εκφώνηση
- Για κάθε γράμμα που διαβάζουμε βρίσκουμε τον cipher, με την cipher() που θα δούμε στη συνέχεια

Cryptography (4)


```java
47  @   public static ArrayList<Character> decrypt(ArrayList<Integer> encrypted, int key) {
48          ArrayList<Character> decrypted = new ArrayList<>();
49          int token;
50          int i;
51          int prevToken;
52
53          prevToken = decipher(encrypted.get(0), -1, key);
54          decrypted.add((char) prevToken);
55
56          i = 1;
57          while ((token = encrypted.get(i)) != -1) {
58              decrypted.add(decipher(token, prevToken, key));
59              prevToken = token;
60              i++;
61          }
62
63          return decrypted;
64      }
```

- Ακολουθούμε κι εδώ τη λογική που περιγράφεται στην εκφώνηση. Για κάθε αριθμό που διαβάζουμε βρίσκουμε τον decipher, με την decipher() που θα δούμε στη συνέχεια

Cryptography (5)


```
66      public static int cipher(char ch, int prev, int key) {
67          if (prev == -1) return ch;
68
69          // Το cipher θα είναι μεταξύ 0 και (key-1)
70          return (ch + prev) % key;
71      }
72
73      public static char decipher(int cipher, int prev, int key) {
74          int m;
75
76          if (prev == -1) return (char) cipher;
77
78          // Αν cipher-prev είναι αρνητικό θα πρέπει να διορθώσουμε με + key για να
79          // πάρουμε θετικό και άρα να τρέξει σωστά το modulo
80          m = (cipher - prev + key) % key;
81          return (char) m;
82      }
83  }
```

Max συνεχόμενα zeros (1)

- Έστω η δυαδική αναπαράσταση ενός ακεραίου αριθμού. Βρείτε τα μέγιστο αριθμό συνεχόμενων 0 που βρίσκονται ανάμεσα σε δύο 1.
- Για παράδειγμα ο αριθμός 5 έχει δυαδική αναπαράσταση 101 και ο μέγιστος αριθμός μηδενικών μεταξύ των 1, είναι 1
- Ο αριθμός 81 έχει δυαδική αναπαράσταση 1010001 και ο μέγιστος αριθμός συνεχόμενων 0, είναι 3.
- Ο 80 έχει αναπαράσταση 1010000 και ο μέγιστος αριθμός μηδενικών μεταξύ των 1, είναι 1.
- Γράψτε ένα αλγόριθμο maxZerosBetweenOnes(int n) που να δίνει το αποτέλεσμα σε γραμμικό χρόνο (όχι εμφωλιασμένη for ή while που δίνει πολυπλοκότητα χρόνου $O(n^2)$

Max συνεχόμενα zeros (2)


```java
package testbed.chalenges;

import java.util.Scanner;

public class MaxZerosShift {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int n;
        int count = 0;
        boolean oneFlag = false;
        int max = 0;

        System.out.println("Please insert a number");
        n = in.nextInt();

        for (; n > 0; n >>= 1) {
            // if rightmost bit is one
            if (n % 2 != 0) {
                oneFlag = true;
                if (count > max) {
                    max = count;
                }
                count = 0;
            } else if (oneFlag) {
                count++;
            }
        }
        System.out.println("Zeros: " + max);
    }
}
```

- Η βασική ιδέα είναι να κάνουμε shift κατά 1 δεξιά τον αριθμό και μόλις το rightmost bit είναι 1 να έχουμε ένα flag ότι ‘είδαμε 1’ και όταν το επόμενο είναι 0 (αφού έχουμε δει 1, όταν flag είναι 1) να ξεκινήσουμε να μετράμε τα zeros (count++). Αν ξαναδούμε 1, τότε if (count > max) max = count και count = 0 πάλι για να ξαναμετρήσουμε επόμενα zeros αν υπάρχουν

Max συνεχόμενα zeros (4)


```
23  public class MaxZeros {
24
25      public static void main(String[] args) {
26          int n = 5;
27          //System.out.println(Integer.toBinaryString(n));
28          int max = maxZeros(n);
29          System.out.println(max);
30      }
31
```

Circular shift right by an offset

- Δοθέντος ενός πίνακα με ακεραίους, θέλουμε μία μέθοδο, έστω int[] doCircularShiftBy(int[] arr, int n) που να μεταθέτει δεξιά κατά n θέσεις όλα τα στοιχεία του πίνακα. Τα τελευταία στοιχεία μετακινούνται στην αρχή του πίνακα. Για παράδειγμα ένα shift κατά 2 του πίνακα [1, 2, 3, 4, 5] δίνει [4, 5, 1, 2, 3]

Circular shift right


```java
public class CyclicRotation {

    public static void main(String[] args) {
        int[] arr = {1, 2, 3};
        int[] rotated = doCircularShiftRightBy(arr, 2);
        for (int i : rotated) {
            System.out.print(i + " ");
        }
    }

    public static int[] doCircularShiftRightBy(int[] arr, int offset) {
        int[] rotated = new int[arr.length];

        // We compute the modulus of arr.length, so
        // the equal or greater indexes wrap around
        // circularly.
        for (int i = 0; i < arr.length; i++) {
            rotated[(i + offset) % arr.length] = arr[i];
        }
        return rotated;
    }
}
```

- Το modulus (%) η λειτουργεί ως ένα καλούπι, που δίνει αριθμούς στο διάστημα έως (n-1)
- Αν θέλουμε επομένως από το length και μετά του πίνακά μας (που είναι η θέση που υπερβαίνει κατά 1 τη θέση του τελευταίου στοιχείου του πίνακα) να εισάγουμε τα στοιχεία αυτά στην αρχή κάνουμε mod με το length, οπότε θα εισάγονται πάντα στο διάστημα του πίνακα 0 – length - 1

Pairs (1)

- Έστω ένας πίνακας με περιττό αριθμό στοιχείων, όπου όλα τα στοιχεία είναι σε ζεύγη εκτός από ένα αριθμό
- Γράψτε μία efficient μέθοδο που να βρίσκει και να επιστρέφει αυτόν τον αριθμό
- Αν για παράδειγμα ο πίνακας είναι ο [1, 2, 3, 4, 3, 2, 1] πρέπει να επιστραφεί το 4

Pairs (2)


```java
public class NotPaired2 {

    public static void main(String[] args) {
        int[] arr = new int[] {1, 1, 1};
        int result = 0;

        for (int num : arr) {
            // Το XOR ακυρώνει τα ζεύγη όμοιων αριθμών
            result ^= num;
        }
        System.out.println("Result: " + result);
    }
}
```

- Η γενική ιδέα θα ήταν να έχουμε ένα βοηθητικό πίνακα (*backedArray*) που να εισάγουμε το στοιχείο που βλέπουμε για 1η φορά και να το διαγράφουμε αν το δούμε και 2η φορά. Το στοιχείο που θα παραμείνει είναι το στοιχείο που δεν έχει pair. Κάτι τέτοιο θα είχε κακή πολυπλοκότητα χώρου.
- Ένα hack θα ήταν να κάνουμε XOR όλα τα στοιχεία του πίνακα μεταξύ τους. Σε αυτή την περίπτωση τα ζεύγη θα ακυρώνοντας (το XOR θα δώσει 0 στα ζεύγη) και θα παραμείνει ως result ο αριθμός χωρίς ζεύγος

Frog Jumps (1)

- Έστω ένας βάτραχος που κάνει jumps συγκεκριμένου μήκους jmp. Γράψτε πόσα jumps χρειάζεται για να ξεκινήσει από ένα σημείο x, και να φτάσει σε ή να ξεπεράσει ένα σημείο y
- Για παράδειγμα αν κάθε jump είναι 25 και ξεκινήσει από x=10 και είναι y=55, χρειάζεται 2 jumps (10+25=35, 35+25=60)

Frog Jumps (2)


```java
public class FrogJumps {

    public static void main(String[] args) {
        int x = 10, y = 85, d = 30;
        System.out.println(getJumps(x, y, d));
    }

    public static int getJumps(int startPos, int endPos, int jmp) {
        int jmpPoint;
        int jumpCount = 0;

        jmpPoint = startPos;
        while (jmpPoint < endPos) {
            jmpPoint += jmp;
            jumpCount++;
        }

        return jumpCount;
    }
}
```

- Μία ιδέα θα ήταν μία while
- Σε κάθε iteration προσθέτουμε το jmp και αυξάνουμε κατά 1 τον μετρητή (jumpCount)
- Η πολυπλοκότητα χρόνου είναι γραμμική δεδομένου ότι έχουμε την while
- Μπορούμε να έχουμε καλύτερη λύση;

Frog Jumps (3)


```java
package testbed.chalenges;

public class FrogJumps2 {

    public static void main(String[] args) {
        int x = 10;
        int y = 85;
        int d = 20;

        System.out.println(getJumps(x, y, d));
    }

    public static int getJumps(int startPos, int endPos, int jmp) {

        return (int) Math.ceil((endPos - startPos) / (double) jmp);
    }
}
```

Missing στοιχείο πίνακα (1)

- Έστω ένας array με N στοιχεία από 1 έως N+1
- Γράψτε ένα αποδοτικό αλγόριθμο που βρίσκει το στοιχείο που λείπει
- Για παράδειγμα, αν έχουμε [1, 2, 3, 5] ο αλγόριθμος θα πρέπει να επιστρέψει 4

Missing στοιχείο πίνακα (2)


```java
package testbed.chalenges;

public class MissingArrayElement2 {

    public static void main(String[] args) {
        int[] arr = {1, 2, 4, 5, 6, 7, 8, 9, 10};
        System.out.println(getMissingElement(arr));
    }

    public static int getMissingElement(int[] arr) {
        int expectedSum = 0;
        int actualSum = 0;
        int n;

        n = arr.length + 1;
        expectedSum = n * (n + 1) / 2;
        for (int num : arr) {
            actualSum += num;
        }
        return expectedSum - actualSum;
    }
}
```

- Η ιδέα είναι να βρούμε το αναμενόμενο άθροισμα των n+1 στοιχείων από 1 .. n+1, που είναι n(n+1) / 2
- Στη συνέχεια να υπολογίσουμε το πραγματικό άθροισμα των n στοιχείων
- Η διαφορά θα είναι το στοιχείο που λείπει

Min time to pass the river (1)

- Έστω ότι σε ένα ποτάμι κάθε 1 sec πέφτει ένα φύλλο από ένα δένδρο σε ένα σημείο έστω p που βρίσκεται μεταξύ δύο σημείων 1 και y. Έστω ότι ένας frog θέλει να περάσει στην απέναντι όχθη μόλις συμπληρωθεί ένα μονοπάτι από το σημείο 1 έως το σημείο y, από φύλλα.
- Έστω ένας πίνακας όπου περιέχει τα σημεία που πέφτουν κάθε sec τα φύλλα. Για παράδειγμα έστω ότι το y=4 και T = [1, 1, 2, 2, 4, 3, 1] δηλ. T[0] = 1, T[1] = 1, T[2] = 2, T[3] = 2, T[4] = 4, T[5] = 3, T[6]=1
- Τη στιγμή 5 (στο 5^{o} second) είναι ο ελάχιστος χρόνος που μπορεί ο frog να περάσει απέναντι γιατί έχει συμπληρωθεί όλο το μονοπάτι από 1 έως 4

Min time to pass the river (2)

- Γράψτε μία μέθοδο που να επιστρέφει τον ελάχιστον χρόνο ή -1 αν ο frog δεν μπορεί να περάσει απέναντι

Min time to pass the river (3)


```java
public static int getMinTime(int y, int[] arr) {
    int[] helperArray = new int[y+1];
    int remainingSteps = y;
    int time = -1;

    for (int i = 0; i < arr.length; i++) {
        // If the leave has not set (0), set it (1) at the
        // helper array. The index of helper array is the
        // position where the leave falls.
        if (helperArray[arr[i]] == 0) {
            helperArray[arr[i]] = 1;
            remainingSteps--;
        } else continue;

        if (remainingSteps == 0) {
            time = i;
            break;
        }
    }
    return time;
}
}
```

- Η ιδέα είναι ένας helper array που αποθηκεύει στις αντίστοιχες index θέσεις αν έχει πέσει φύλλο ή όχι
- Μπορούμε κάθε φορά που πέφτει φύλλο σε θέση που δεν υπήρχε φύλλο, να μειώνουμε τη συνολική διαδρομή κατά 1
- Μόνο αν η συνολική υπολειπόμενη διαδρομή (*remainingSteps*) είναι 0, θα μπορεί ο frog να περάσει απέναντι. Ο χρόνος είναι το *i* του αρχικού πίνακα τη στιγμή που συμπληρώνεται η διαδρομή

Min time to pass the river (4)


```
1   package testbed.chalenges;
2
3   public class MinTimeToPassTheRiver {
4
5       public static void main(String[] args) {
6           int[] T = {1, 1, 2, 2, 4, 1, 3};
7           System.out.println(getMinTime(4, T));
8       }
9
```

Registers (1)

- Έστω ένας πίνακας από n registers (register array) που αρχικά είναι όλοι μηδέν. Έστω ένας άλλος πίνακας που περιέχει τη θέση ενός register στο reg-array και τότε θα πρέπει ο αντίστοιχος register να αυξήσει την τιμή του κατά 1, ενώ αν δεν περιέχει τη θέση ενός register, τότε όλοι οι registers λαμβάνουν την μέγιστη τιμή των registers
- Γράψτε ένα πρόγραμμα int[] getRegisterValues(int n, int[] arr) που να υπολογίζει τα register values μετά από όλες τις πράξεις

Registers (2)

- Για παράδειγμα αν έχουμε 5 registers και ένα πίνακα arr = [0, 1, 2, 4, 4, 6, 1, 1, 1, 1, 7, 0] τότε αναμένουμε να πάρουμε τον πίνακα αρχικά είναι [0, 0, 0, 0, 0] και τελικά [7, 6, 6, 6, 6]
- Στο πρόβλημα αυτό θα χρησιμοποιήσουμε 'functional programming' δηλαδή αντί να κάνουμε for loop για να διατρέξουμε τον πίνακα, θα μετατρέψουμε τον πίνακα σε 'stream' –μία ειδική δομή της Java- πάνω στην οποία μπορούμε να εφαρμόζουμε μεθόδους όπως max(), min() κλπ.
- Το Stream API θα το δούμε πιο αναλυτικά σε επόμενα μαθήματα

Registers (3)


```java
    /**
     * The goal of this method is to create an array of n
     * registers, and then based on the input-array to manage
     * the state of each register element in the register-array.
     * If the value of the input-array is equal to a register position
     * in the register-array, then the state of the register will be
     * increased by one, otherwise the state of all registers will be
     * increased and get equal to the max register state.
     *
     * @param n     the length of the register-array.
     * @param arr   the input-array.
     * @return      the register-array with the updated values.
     */
    public static int[] getRegisterValues(int n, int[] arr) {
        int[] registers = new int[n];
        int max;

        for (int val : arr) {
            if ((val >= 0) && (val <= n-1)) {
                registers[val]++;
            } else {
                max = Arrays.stream(registers).max().orElse(0);
                Arrays.fill(registers, max);
            }
        }
        return registers;
    }
}
```

- Ο στόχος αυτής της μεθόδου είναι να δημιουργήσει έναν πίνακα n και στη συνέχεια –με βάση ένα input-array- να διαχειρίζεται, την κατάσταση κάθε στοιχείου στον πίνακα registers
- Εάν η τιμή ενός στοιχείου του πίνακα εισόδου είναι ίση με μια θέση του πίνακα των registers, τότε στον πίνακα των registers, η τιμή του αντίστοιχου register θα αυξηθεί κατά 1, διαφορετικά η κατάσταση όλων των θέσεων του πίνακα registers θα αυξηθεί και θα ισούται με τη μέγιστη τιμή του των στοιχείων register-array

Registers (4)


```java
package testbed.chalenges;

import java.util.Arrays;

public class Registers {

    public static void main(String[] args) {
        int[] arr = {0, 1, 2, 4, 4, 6, 1, 1, 1, 1, 7, 0};
        int n = 5;
        int[] regs = getRegisterValues(n, arr);
        for (int reg : regs) {
            System.out.print(reg + " ");
        }
    }
```

Μικρότερος missing θετικός (1)

- Δοθέντος ενός πίνακα μήκους N με ακεραίους, γράψτε ένα πρόγραμμα int getSmallestMissingPositive(int[] arr) που επιστρέφει τον μικρότερο θετικό (>0) που δεν υπάρχει στον πίνακα
- Για παράδειγμα αν ο πίνακας είναι [-1, -2, -3, -4, 0] θα επιστρέψει 1, αν είναι [2, 1, 6, 3, 5, 9] θα επιστρέψει 4

Μικρότερος missing θετικός (2)


```java
package testbed.chalenges;

import java.util.Arrays;
import java.util.HashSet;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

public class SmallestPositive2 {

    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5, 6};
        System.out.println(getSmallestPositive(arr));
    }

    public static int getSmallestPositive(int[] arr) {
        HashSet<Integer> set = Arrays.stream(arr).boxed().collect(Collectors.toCollection(HashSet::new));
        return IntStream.rangeClosed(1, arr.length + 1)
                .filter(i -> !set.contains(i))
                .findFirst()
                .orElse(-1);
    }
}
```

- Πρόκειται για το Stream API που θα δούμε αργότερα. Οι πίνακες και άλλα collections στοιχείων μπορούν να μετατραπούν σε streams, ώστε να κάνουμε εύκολα πράξεις στα στοιχεία του stream με functional programming και να μετατρέπουμε ξανά σε collections, όπως το HashSet που είναι σύνολο (set) άρα δεν κρατάει διπλότυπα αλλά και Hash οπότε ο χρόνος αναζήτησης είναι O(1). Ο τελικός χρόνος εύρεσης του missing element είναι O(n)

Μικρότερος missing θετικός (3)


```java
package testbed.chalenges;

import java.util.Arrays;
import java.util.HashSet;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

public class SmallestPositive2 {

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

        // Should return 5
        System.out.println(getSmallestPositive(arr));
    }

```

Mobiles App (1)

- Αναπτύξτε μία εφαρμογή επαφών για ένα κινητό τηλέφωνο, η οποία μπορεί να περιέχει μέχρι 500 επαφές. Κάθε επαφή έχει Όνομα , Επώνυμο και Τηλέφωνο.

- Για να αποθηκεύεται τις επαφές χρησιμοποιήστε ένα δυσδιάστατο πίνακα 500x3 όπου στην 1η θέση κάθε επαφής θα αποθηκεύεται το Όνομα, στη 2η θέση το Επώνυμο και στην 3η θέση το τηλέφωνο, όλα ως String.

Mobiles App (2)

- Χρησιμοποιήστε στρωματοποιημένη αρχιτεκτονική κατηγοριοποιώντας τις μεθόδους σας σε τρία μέρη: CRUD Μέθοδοι, Service Layer, Controllers
- Οι CRUD μέθοδοι θα υλοποιούν τις βασικές **CRUD** πράξεις: Αναζήτηση Επαφής με βάση το τηλέφωνο, Εισαγωγή Επαφής (δεν επιτρέπονται διπλότυπα), Ενημέρωση Επαφής (εάν υπάρχει), Διαγραφή Επαφής (εάν υπάρχει)

Mobiles App (3)

- Το Service Layer θα παρέχει ολοκληρωμένες υπηρεσίες CRUD χρησιμοποιώντας το CRUD Layer. Επιστρέφει data, ανάλογα τι έχει ζητηθεί
- Χρησιμοποιεί exceptions για να επιστρέψει στον caller (Controller) error μηνύματα και να κάνει γενικά διαχείριση λαθών

Mobiles App (4)

- Οι Controllers κάνουν τη 'δρομολόγηση' των αιτημάτων που λαμβάνουν από τον client προς το κατάλληλο service και αποστέλλουν πίσω ένα response (success / failure / data)
- Κάνουν επίσης validation των δεδομένων εισόδου καλώντας ειδικές μεθόδους (validators)

Mobiles App (5)

- Η main() θα είναι ο client που θα κάνει το I/O με τον χρήστη και θα καλεί στη συνέχεια τους controllers.
- Θα πρέπει να εμφανίζει ένα μενού στον χρήστη, οποίος θα επιλέγει την κατάλληλη πράξη ή Έξοδο από την εφαρμογή (με q ή Q) και με την κατάλληλη καθοδήγηση της εφαρμογής θα επιτελεί την επιλεγμένη πράξη

Ενδεικτική λύση

- https://github.com/a8anassis/codingfactory5-java/tree/main/src/gr/aueb/cf/ch10