



Εφαρμογές και παραδείγματα στον Δομημένο Προγραμματισμό

Αθ. Ανδρούτσος



Armstrong Numbers (1)

Προγραμματισμός με Java

- 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)

Προγραμματισμός με Java

```
1 package testbed.review;
2
3 import ...
4
5
6
7 /**
8  * Βρίσκει αν το άθροισμα των δυνάμεων κάθε ψηφίου εις τη (πλήθος ψηφίων)
9  * είναι ίσο με τον ίδιο τον αριθμό. Για παράδειγμα, ο αριθμός 153 = 1^3 + 5^3 + 3^3.
10 * Επομένως, το 153 είναι Armstrong number. Επίσης, τα 370, 371, 407.
11 */
12 public class ArmstrongApp {
13
14     public static void main(String[] args) {
15         final Scanner in = new Scanner(System.in);
16         final List<Integer> digits = new ArrayList<>();
17         int inputNumber = 0;
18         int digitsCount = 0;
19         int num = 0;
20         int sum = 0;
21         int digit = 0;
22         boolean isArmstrong = false;
```

- Δεδομένου ότι δεν γνωρίζουμε πόσα ψηφία θα δώσει ο χρήστης, χρησιμοποιούμε μία `List<>` που υλοποιείται ως `ArrayList<Integer>` για να αποθηκεύουμε τα ψηφία του αριθμού
- Η `ArrayList` είναι απλά είναι δυναμικός πίνακας, δηλαδή πίνακας που μπορεί όμως να αυξάνει ή να μικραίνει αυτόματα το μέγεθός του



Armstrong Numbers (3)

Προγραμματισμός με Java

```
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)

Προγραμματισμός με Java

```
1 package testbed.review;
2
3 import java.util.Scanner;
4
5 /**
6  * Μετατρέπει σε κεφαλαία-πεζά εναλλάξ με το 1ο κεφαλαίο.
7  * Παράδειγμα: "athanassios" -> "AtHaNaSsIoS"
8  */
9 public class UpperLowerCaseApp {
10
11     public static void main(String[] args) {
12         Scanner in = new Scanner(System.in);
13         StringBuilder sb = new StringBuilder();
14         String sentence;
15         char ch;
16
17         System.out.println("Please insert a string");
18         sentence = in.nextLine();
19         for (int i = 0; i < sentence.length(); i++) {
20             ch = sentence.charAt(i);
21             if (Character.isLetter(ch)) {
22                 ch = ((i % 2) == 0) ? Character.toUpperCase(ch)
23                     : Character.toLowerCase(ch);
24                 sb.append(ch);
25             }
26         }
27
28         for (int i = 0; i < sb.length(); i++) {
29             System.out.print(sb.charAt(i));
30         }
31     }
32 }
```

- 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)

Προγραμματισμός με Java

```
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

Προγραμματισμός με Java

- 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)

Προγραμματισμός με Java

- Κωδικοποίησε τον 1^ο 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)

Προγραμματισμός με Java

- 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)

Προγραμματισμός με Java

```
1 package testbed.review;
2
3 import java.util.ArrayList;
4
5 /**
6  * Κωδικοποίησε τον 1ο χαρακτήρα του μηνύματος με την ακέραια τιμή που αντιστοιχεί σε αυτόν
7  * (από τον κώδικα ASCII). Κωδικοποίησε του επόμενους χαρακτήρες: (α) προσθέτοντας την ακέραια
8  * ASCII τιμή του καθένα από αυτούς με τον κωδικό του προηγούμενου του,
9  * (β) παίρνοντας το υπόλοιπο της διαίρεσης του αθροίσματος αυτού διά μία σταθερά.
10 * Υποθέτουμε πως τα μηνύματα τελειώνουν με τον χαρακτήρα #
11 * Γράψτε ένα πρόγραμμα java που να υλοποιεί τον αλγόριθμο κρυπτογράφησης έτσι ώστε το
12 * κωδικοποιημένο μήνυμα που προκύπτει να είναι μία ακολουθία ακεραίων που τελειώνει με -1
13 * Γράψτε και τον αλγόριθμο αποκρυπτογράφησης που λαμβάνει ως είσοδο μία ακολουθία ακεραίων
14 * που τελειώνει με -1 και υπολογίζει το αρχικό μήνυμα
15 */
16 a8ana *
```



Cryptography (2)

Προγραμματισμός με Java

```
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)

Προγραμματισμός με Java

```
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)

Προγραμματισμός με 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)

Προγραμματισμός με Java

```
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)

Προγραμματισμός με Java

- 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)

Προγραμματισμός με Java

```
1 package testbed.challenges;
2
3 import java.util.Scanner;
4
5 public class MaxZerosShift {
6
7     public static void main(String[] args) {
8         Scanner in = new Scanner(System.in);
9         int n;
10        int count = 0;
11        boolean oneFlag = false;
12        int max = 0;
13
14        System.out.println("Please insert a number");
15        n = in.nextInt();
16    }
```

```
17
18        // if rightmost bit is one
19        if (n % 2 != 0) {
20            oneFlag = true;
21            if (count > max) {
22                max = count;
23            }
24            count = 0;
25        } else if (oneFlag) {
26            count++;
27        }
28    }
29    System.out.println("Zeros: " + max);
30 }
31 }
```

- 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)

Προγραμματισμός με Java

```
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

Προγραμματισμός με Java

- 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

Προγραμματισμός με Java

- 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

```
20 public class CyclicRotation {
21
22     public static void main(String[] args) {
23         int[] arr = {1, 2, 3};
24         int[] rotated = doCircularShiftRightBy(arr, 2);
25         for (int i : rotated) {
26             System.out.print(i + " ");
27         }
28     }
29
30     @ public static int[] doCircularShiftRightBy(int[] arr, int offset) {
31         int[] rotated = new int[arr.length];
32
33         // We compute the modulus of arr.length, so
34         // the equal or greater indexes wrap around
35         // circularly.
36         for (int i = 0; i < arr.length; i++) {
37             rotated[(i + offset) % arr.length] = arr[i];
38         }
39         return rotated;
40     }
41 }
```



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)

```
30 ▶ public class NotPaired2 {  
31  
32 ▶   public static void main(String[] args) {  
33       int[] arr = new int[] {1, 1, 1};  
34       int result = 0;  
35  
36       for (int num : arr) {  
37           // Το XOR ακυρώνει τα ζεύγη όμοιων αριθμών  
38           result ^= num;  
39       }  
40       System.out.println("Result: " + result);  
41   }  
42 }
```

- Η γενική ιδέα θα ήταν να έχουμε ένα βοηθητικό πίνακα (*backedArray*) που να εισάγουμε το στοιχείο που βλέπουμε για 1^η φορά και να το διαγράφουμε αν το δούμε και 2^η 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 ακυρώνονταν (το XOR θα δώσει 0 στα ζεύγη) και θα παραμείνει ως result ο αριθμός χωρίς ζεύγος



Frog Jumps (1)

Προγραμματισμός με Java

- 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)

Προγραμματισμός με Java

```
23 ▶ public class FrogJumps {
24
25 ▶   public static void main(String[] args) {
26       int x = 10, y = 85, d = 30;
27       System.out.println(getJumps(x, y, d));
28   }
29
30   public static int getJumps(int startPos, int endPos, int jmp) {
31       int jmpPoint;
32       int jumpCount = 0;
33
34       jmpPoint = startPos;
35       while (jmpPoint < endPos) {
36           jmpPoint += jmp;
37           jumpCount++;
38       }
39
40       return jumpCount;
41   }
42 }
```

- 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)

Προγραμματισμός με Java

```
1 package testbed.challenges;
2
3 public class FrogJumps2 {
4
5     public static void main(String[] args) {
6         int x = 10;
7         int y = 85;
8         int d = 20;
9
10        System.out.println(getJumps(x, y, d));
11    }
12
13    public static int getJumps(int startPos, int endPos, int jmp) {
14
15        return (int) Math.ceil((endPos - startPos) / (double) jmp);
16    }
17 }
```




Missing στοιχείο πίνακα (1)

Προγραμματισμός με Java

- 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)

Προγραμματισμός με Java

```
1 package testbed.challenges;
2
3 public class MissingArrayElement2 {
4
5     public static void main(String[] args) {
6         int[] arr = {1, 2, 4, 5, 6, 7, 8, 9, 10};
7         System.out.println(getMissingElement(arr));
8     }
9
10    public static int getMissingElement(int[] arr) {
11        int expectedSum = 0;
12        int actualSum = 0;
13        int n;
14
15        n = arr.length + 1;
16        expectedSum = n * (n + 1) / 2;
17        for (int num : arr) {
18            actualSum += num;
19        }
20        return expectedSum - actualSum;
21    }
22 }
```

- 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)

Προγραμματισμός με Java

- 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 γ . Έστω ότι ένας frog θέλει να περάσει στην απέναντι όχθη μόλις συμπληρωθεί ένα μονοπάτι από το σημείο 1 έως το σημείο γ , από φύλλα.
- Έστω ένας πίνακας όπου περιέχει τα σημεία που πέφτουν κάθε sec τα φύλλα. Για παράδειγμα έστω ότι το $\gamma=4$ 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^ο 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)

Προγραμματισμός με Java

- 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)

Προγραμματισμός με Java

```
10 @ public static int getMinTime(int y, int[] arr) {
11     int[] helperArray = new int[y+1];
12     int remainingSteps = y;
13     int time = -1;
14
15     for (int i = 0; i < arr.length; i++) {
16         // If the leave has not set (0), set it (1) at the
17         // helper array. The index of helper array is the
18         // position where the leave falls.
19         if (helperArray[arr[i]] == 0) {
20             helperArray[arr[i]] = 1;
21             remainingSteps--;
22         } else continue;
23
24         if (remainingSteps == 0) {
25             time = i;
26             break;
27         }
28     }
29     return time;
30 }
31 }
```

- 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)

Προγραμματισμός με Java

```
1 package testbed.challenges;  
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
```