ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν Πανεπιστήμιον Αθηνών
ΙΔΡΥΘΕΝ ΤΟ 1837

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Κατατακτήριες Εξετάσεις 2023

Εισαγωγή στον Προγραμματισμό (11/12/2023)

Απαντήστε και στα 4 θέματα, τα οποία είναι βαθμολογικά ισοδύναμα. Τα προγράμματα που θα γράψετε πρέπει να είναι δομημένα, διατυπωμένα ευκρινώς και με επαρκή τεκμηρίωση ώστε να είναι κατανοητά.

Θέμα 1 - Κάντο όπως ο Βιετά (25 Μονάδες)

Ο Γάλλος μαθηματικός Βιετά (François Viète) υπήρξε ο πρώτος μαθηματικός που χρησιμοποίησε ευρέως σύμβολα για να εκφράσει αριθμητικές ποσότητες. Το 1593 κατάφερε να εκφράσει και να υπολογίσει τον αριθμό π με ακρίβεια 9 δεκαδικών, βελτιώνοντας έτσι το σχετικό αποτέλεσμα του Αρχιμήδη. Για να υπολογίσει το π, ο Βιετά χρησιμοποίησε μια σχέση που χρησιμοποιεί ένα απειρογινόμενο όρων t_i, γνωστό ως φόρμουλα Βιετά προς τιμήν του:

$$\frac{2}{\pi} = \underbrace{\frac{\sqrt{2}}{2}}_{t_1} \cdot \underbrace{\frac{\sqrt{2+\sqrt{2}}}{2}}_{t_2} \cdot \underbrace{\frac{\sqrt{2+\sqrt{2+\sqrt{2}}}}{2}}_{t_3} \cdots$$

Λύνοντας την παραπάνω σχέση ως προς π, μπορούμε και εμείς να προσεγγίσουμε το π όπως ο Βιετά. Μάλιστα, όσους περισσότερους όρους t_i χρησιμοποιούμε, τόσο καλύτερη η προσέγγισή μας. Γράψτε ένα πρόγραμμα C το οποίο παίρνει έναν θετικό ακέραιο ως όρισμα από την γραμμή εντολών που αντιπροσωπεύει πόσους πρώτους όρους t_i να χρησιμοποιήσει στον υπολογισμό του π και στην συνέχεια εκτυπώνει την προσέγγιση του π με 9 δεκαδικά ψηφία. Παραδείγματα εκτέλεσης είναι τα εξής:

```
$ ./vieta 2
Multiplied first 2 ti terms, pi = 3.061467459
$ ./vieta 5
Multiplied first 5 ti terms, pi = 3.140331157
$ ./vieta 50
Multiplied first 50 ti terms, pi = 3.141592654
```

Θέμα 2 - Εαυτοί Αριθμοί (25 Μονάδες)

Στην θεωρία αριθμών, ένας φυσικός αριθμός n λέγεται *εαυτός* (self) όταν δεν υπάρχει κάποιος φυσικός αριθμός m, έτσι ώστε το n να ισούται με το άθροισμα του m και των ψηφίων του m (με βάση το 10 σε αυτό το θέμα). Έστω $F(n)$ η συνάρτηση που υπολογίζει το άθροισμα ενός φυσικού n και των ψηφίων του. Τότε ένας αριθμός n είναι εαυτός αν και μόνο αν $\nexists m.F(m) = n$. Για παράδειγμα, $F(15) = 15 + 1 + 5 = 21$ και επομένως ο αριθμός 21 δεν είναι εαυτός. Αντίθετα, ο αριθμός 20 είναι εαυτός καθώς δεν υπάρχει φυσικός αριθμός m έτσι ώστε $F(m) = 20$. Υπάρχουν μόλις 13 εαυτοί αριθμοί μικρότεροι του 100:

$$1, 3, 5, 7, 9, 20, 31, 42, 53, 64, 75, 86, 97$$

Γράψτε ένα πρόγραμμα C το οποίο βρίσκει και τυπώνει όλους τους εαυτούς αριθμούς σε ένα εύρος φυσικών αριθμών $[low, high]$ (το εύρος είναι κλειστό, δηλαδή τα άκρα συμπεριλαμβάνονται), όπου οι αριθμοί δίνονται από την γραμμή εντολών. Ακολουθεί παράδειγμα εκτέλεσης για να βρούμε όλους τους εαυτούς αριθμούς στο διάστημα $[9900, 10000]$:

```
$ ./self 9900 10000
Self numbers: 9903 9914 9925 9927 9938 9949 9960 9971 9982 9993
Found 10 total
```

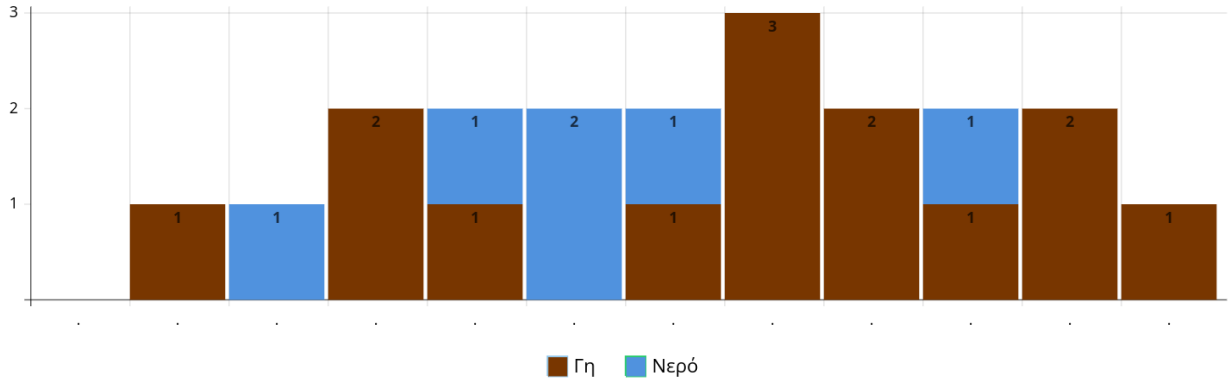
Θέμα 3 - Σκαλί-Σκαλί (25 Μονάδες)

Ένα παιδί ανεβαίνει μια σκάλα και σε κάθε βήμα του ανεβαίνει 1, 2 ή 3 σκαλιά. Έστω ότι η σκάλα έχει 5 σκαλιά. Ένας τρόπος να τα ανέβει είναι 1-1-1-1-1, δηλαδή ένα σκαλί την φορά. Ένας άλλος είναι ο 2-2-1 (δύο σκαλιά στα πρώτα δύο βήματα και μετά ένα σκαλί). Άλλοι πιθανοί τρόποι είναι οι 3-2, 2-3, 3-1-1, κ.ο.κ - σε κάθε περίπτωση ο συνολικός αριθμός των σκαλιών που ανέβηκε πρέπει να ισούται με τον αριθμό των σκαλιών της σκάλας. Γράψτε ένα πρόγραμμα C το οποίο διαβάζει τον συνολικό αριθμό των σκαλοπατιών και επιστρέφει τον αριθμό των διαφορετικών τρόπων με τους οποίους το παιδί μπορεί να ανέβει την σκάλα. Το πρόγραμμά σας θέλουμε να βγάζει σωστά αποτελέσματα για σκάλες μέχρι 50 σκαλιά. Λάβετε υπόψη σας ότι στην C δεν μπορούν να αναπαρασταθούν ακέραιοι μεγαλύτεροι από το $2^{64} - 1$, δεδομένου ότι ο ευρύτερος τύπος ακεραίου που μπορούμε να έχουμε είναι ο `unsigned long long` (των 8 bytes). Ο αλγόριθμός σας πρέπει να έχει χρονική πολυπλοκότητα γρηγορότερη από $O(n^2)$, όπου n ο αριθμός των σκαλιών. Ακολουθούν ενδεικτικές εκτελέσεις:

```
$ ./step
Please provide the number of steps: 4
There are 7 different ways to climb the ladder
$ ./step
Please provide the number of steps: 5
There are 13 different ways to climb the ladder
$ ./step
Please provide the number of steps: 6
There are 24 different ways to climb the ladder
$ ./step
Please provide the number of steps: 50
There are 10562230626642 different ways to climb the ladder
```

Θέμα 4 - Το Νερό Νεράκι (25 Μονάδες)

Το νερό είναι πολύτιμο και είναι σημαντικό να γνωρίζουμε ποια είναι τα διαθέσιμα αποθέματά μας. Δοθέντος του υψομετρικού χάρτη μιας περιοχής, θέλουμε να μπορούμε να υπολογίζουμε την μέγιστη ποσότητα νερού της βροχής που μπορεί να διατηρηθεί μέσα στις φυσικές δεξαμενές που σχηματίζονται από την διαμόρφωση του εδάφους. Για απλοποίηση, θεωρούμε ότι ο υψομετρικός χάρτης είναι μονοδιάστατος και μας δίνεται ως μια σειρά θετικών ακεραίων (μονάδες ύψους με πλάτος 1) και η ποσότητα νερού μετριέται επίσης στις ίδιες μονάδες ύψους. Το Σχήμα 1 δείχνει ένα παράδειγμα.



Σχήμα 1: Οπτικοποίηση της ποσότητας νερού που θα "παγιδευτεί" μετά την βροχή για έναν ενδεικτικό υψομετρικό χάρτη. Στο παράδειγμα, μέχρι 6 μονάδες νερού μπορούν να συγκρατηθούν από αυτόν τον χάρτη (1 μονάδα στην 3η θέση, 1 στην 5η θέση, 2 στην 6η θέση, 1 στην 7η θέση και 1 στην 10η θέση).

Γράψτε ένα πρόγραμμα C που διαβάζει τον πίνακα των υψών του υψομετρικού χάρτη και επιστρέφει την μέγιστη ποσότητα του νερού της βροχής που μπορεί να συγκρατηθεί από αυτόν. Το πρόγραμμά σας πρέπει να έχει χρονική πολυπλοκότητα καλύτερη από $O(n^2)$, όπου n ο αριθμός των δοθέντων υψών και την ελάχιστη δυνατή χρήση μνήμης. Αφού γράψετε το πρόγραμμά σας, καταγράψτε και εξηγήστε την πολυπλοκότητά του ως προς τον χρόνο και τον χώρο μνήμης που απαιτεί. Ακολουθούν ενδεικτικές εκτελέσεις:

```
$ ./water
Provide the number of heights: 12
Provide the heights: 0 1 0 2 1 0 1 3 2 1 2 1
Up to 6 units of water can be trapped
$ ./water
Provide the number of heights: 12
Provide the heights: 4 1 0 2 1 0 1 3 2 3 2 1
Up to 14 units of water can be trapped
./water
Provide the number of heights: 20
Provide the heights: 0 2 3 1 4 4 4 4 3 8 9 3 9 0 6 4 6 0 9 5
Up to 38 units of water can be trapped
```