ΕΙΣΑΓΩΓΗ ΣΤΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟ (2018-19)

Εργασία 1

Ένας ακέραιος αριθμός μεγαλύτερος του 1 ονομάζεται *πρώτος* (prime) όταν έχει σαν μόνους διαιρέτες το 1 και τον εαυτό του. Για παράδειγμα, το 13 είναι πρώτος αριθμός, ενώ το 15 ($= 3 \times 5$) δεν είναι. Ένας αριθμός που δεν είναι πρώτος ονομάζεται *σύνθετος* (composite). Δεν είναι ιδιαίτερα δύσκολο να γράψουμε έναν αλγόριθμο που να ελέγχει αν ένας αριθμός είναι πρώτος, ο οποίος θα βασίζεται στον προηγούμενο ορισμό, μόνο που για μεγάλους αριθμούς ο αλγόριθμος αυτός θα είναι εξαιρετικά δαπανηρός σε χρόνο. Θα ονομάσουμε ένα τέτοιο αλγόριθμο *ντετερμινιστικό* (deterministic), υπονοώντας ότι μπορεί να αποφασίσει με απόλυτη βεβαιότητα αν ένας αριθμός είναι πρώτος ή σύνθετος.

Από την άλλη πλευρά, υπάρχει και η μέθοδος Miller-Rabin για τον έλεγχο αν ένας αριθμός είναι πρώτος. Πρόκειται για ένα *πιθανοτικό* αλγόριθμο, ο οποίος μπορεί να αποφασίσει αν ένας αριθμός είναι πρώτος, αλλά υπάρχει ένα μικρό ενδεχόμενο να κάνει λάθος, θεωρώντας έναν αριθμό ως πρώτο, ενόσω δεν είναι στην πραγματικότητα. Αναλυτική περιγραφή της μεθόδου μπορείτε να δείτε στην ιστοσελίδα `https://en.wikipedia.org/wiki/Miller-Rabin_primality_test` και ιδιαίτερα στην παράγραφο `https://en.wikipedia.org/wiki/Miller-Rabin_primality_test#Computational_complexity` όπου δίνεται μία διατύπωση του αλγορίθμου σε ψευδοκώδικα.

Παρατηρήστε ότι στον αλγόριθμο, για να ελεγχθεί αν ένας περιττός ακέραιος n είναι πρώτος (οι άρτιοι δεν έχει νόημα να ελεγχθούν, γιατί κανείς, πλην του 2, δεν είναι πρώτος), γίνονται k επιλογές τυχαίων ακεραίων a, καθένας από τους οποίους ελέγχεται αν πληροί κάποιες ιδιότητες που περιγράφονται στον ψευδοκώδικα. Αν όλοι πληρούν τις ιδιότητες αυτές, τότε ο ακέραιος n είναι πιθανότατα πρώτος. Μάλιστα, όσο πιο μεγάλο είναι το k, δηλαδή όσο περισσότεροι τυχαίοι ακέραιοι a επιλεγούν, τόσο λιγότερο πιθανό είναι να κάνει ο αλγόριθμος λάθος, δηλαδή να θεωρήσει ένα σύνθετο αριθμό ως πρώτο.

Όμως, έχει αποδειχθεί ότι αν ο αριθμός n που θέλουμε να ελέγξουμε αν είναι πρώτος δεν υπερβαίνει κάποιο συγκεκριμένο όριο, τότε μπορούμε να κάνουμε τους ελέγχους του αλγορίθμου Miller-Rabin για συγκεκριμένους αριθμούς a και όχι για τυχαία επιλεγμένους και να υπάρχει απόλυτη βεβαιότητα για το αν ο αριθμός n είναι πρώτος ή σύνθετος. Δείτε στην παραπάνω ιστοσελίδα και την παράγραφο `https://en.wikipedia.org/wiki/Miller-Rabin_primality_test#Deterministic_variants` και ιδιαίτερα τη γραμμή

> if n $< 4,759,123,141$, it is enough to test $a = 2$, 7, and 61

Το παραπάνω όριο για το n είναι αρκετό για να μπορούν να ελεγχθούν αν είναι πρώτοι ή όχι όλοι οι ακέραιοι αριθμοί των 4 bytes, ακόμα και `unsigned`, χωρίς να υπάρχει ενδεχόμενο να έχει γίνει λάθος.

Γράψτε ένα πρόγραμμα C (έστω ότι το πηγαίο αρχείο του ονομάζεται "`milrab.c`") το οποίο να υπολογίζει το πλήθος των πρώτων αριθμών που βρίσκονται μέσα σε δεδομένο διάστημα, τόσο με τον βασικό ντετερμινιστικό τρόπο, όσο και με την εκδοχή του αλγορίθμου Miller-Rabin που αναφέρθηκε προηγουμένως και δεν κάνει λάθος αν ο ελεγχόμενος αριθμός είναι μικρότερος του 4759123141. Το διάστημα να καθορίζεται από δύο συμβολικές σταθερές `MINNUM` και `MAXNUM` που θα ορίσετε μέσα στο πρόγραμμά σας. Και οι δύο αλγόριθμοι θα πρέπει να χρονομετρηθούν, ο καθένας συνολικά για όλους τους αριθμούς του διαστήματος που ελέγχεται.

Προτάσεις/Υποδείξεις/Απαγορεύσεις:

1. Θα πρέπει στην έκδοση του προγράμματος που θα παραδώσετε να θέσετε ως τιμές των συμβολικών σταθερών `MINNUM` και `MAXNUM` τις `3990000000` και `4010000000`, αντίστοιχα, γιατί με αυτές τις τιμές θα ελεγχθεί το πρόγραμμά σας ως προς την αποδοτικότητά του. Αρχικά, βέβαια,

μπορείτε να χρησιμοποιήσετε μικρότερες τιμές για να ελέγξετε την ορθότητα του προγράμματός σας.

2. Για διευκόλυνσή σας, η διατύπωση της μη πιθανοτικής εκδοχής του αλγορίθμου Miller-Rabin σε ψευδοκώδικα είναι:

   Μία μη πιθανοτική εκδοχή του αλγορίθμου Miller-Rabin
   Είσοδος: Ένας περιττός ακέραιος n στο διάστημα (2,4759123141)
   Έξοδος: "πρώτος" ή "σύνθετος"

   Γράψε τον $n-1$ στη μορφή $2^r \cdot d$, όπου d περιττός
   Για κάθε a μέσα από το σύνολο $\{2, 7, 61\}$
   Θέσε $x \leftarrow a^d \bmod n$
   Αν $x = 1$ ή $x = n-1$ τότε
   Συνέχισε με το επόμενο a, αν υπάρχει
   Επανάλαβε $r-1$ φορές
   Θέσε $x \leftarrow x^2 \bmod n$
   Αν $x = n-1$ τότε
   Συνέχισε με το επόμενο a, αν υπάρχει
   Επίστρεψε "σύνθετος"
   Επίστρεψε "πρώτος"

3. Η χρονομέτρηση των αλγορίθμων να γίνεται μέσω της συνάρτηση `clock`. Περισσότερες λεπτομέρειες για τη χρήση της συνάρτησης αυτής μπορείτε να πάρετε μέσω της εντολής "`man 3 clock`" στα μηχανήματα Linux του Τμήματος, ή να δείτε τη χρήση της στο πρόγραμμα `http://www.di.uoa.gr/~ip/cprogs/sorting.c`, στις σελίδες 167–169 των διαφανειών του μαθήματος.

4. Είναι προφανές ότι αν δοκιμάσετε να υπολογίσετε τα $a^d \bmod n$ και $x^2 \bmod n$ για μεγάλα d, x και n υπολογίζοντας πρώτα τις δυνάμεις, τα αποτελέσματα δεν θα είναι σωστά λόγω υπερχείλισης. Για να λύσετε αυτό το πρόβλημα, σκεφτείτε ότι ισχύουν τα εξής:[1]

$$p^{2k+1} \bmod m = (p \cdot (p^{2k} \bmod m)) \bmod m$$

$$p^{2k} \bmod m = (p^2 \bmod m)^k \bmod m$$

5. Για να μπορέσετε να εφαρμόσετε τον αλγόριθμο Miller-Rabin για τους αριθμούς του διαστήματος που ορίζεται από τις τιμές των συμβολικών σταθερών που προτείνονται, θα χρειαστείτε, τουλάχιστον σε κάποιες ενδιάμεσες πράξεις, να χρησιμοποιήσετε απροσήμαστους ακεραίους των 8 bytes, κάτι που στη C σας εξασφαλίζει μόνο η χρήση του τύπου `unsigned long long`.

6. Στην άσκηση αυτή **απαγορεύεται αυστηρά η χρήση πινάκων**.

Μία ενδεικτική εκτέλεση[2] του προγράμματος φαίνεται στη συνέχεια:

```
$ ./milrab
Checking range [3990000000,4010000000] for primes
Deterministic algorithm: Found 904533 primes in 50.59 secs
Miller-Rabin algorithm:  Found 904533 primes in  5.16 secs
```

Η παράδοση της άσκησης αυτής συνίσταται στην υποβολή του πηγαίου αρχείου `milrab.c` με διαδικασία που θα ανακοινωθεί σύντομα.

[1] Δείτε και το `http://en.wikipedia.org/wiki/Modular_exponentiation#Right-to-left_binary_method`

[2] Η συγκεκριμένη εκτέλεση έγινε σε μηχάνημα Linux του Τμήματος.