Εισαγωγή στην Επιστήμη Υπολογιστών

2η Σειρά Ασκήσεων

Ονοματεπώνυμο: Λιαροκάπης Αλέξανδρος
Αριθμός Μητρώου: 03114860

Άσκηση 1

(α) Εκφράστε τον αριθμό μετακινήσεων δίσκων που κάνει ο αναδρομικός αλγόριθμος για τους πύργους του Hanoi, σαν συνάρτηση του αριθμού των δίσκων n.

(β) Δείξτε ότι ο αριθμός των μετακινήσεων του αναδρομικού αλγορίθμου είναι ο ελάχιστος μεταξύ όλων των δυνατών αλγορίθμων για το πρόβλημα αυτό.

(γ) Βρείτε τη σχέση ανάμεσα στον αριθμό των μετακινήσεων του αναδρομικού και τον αριθμό των κινήσεων του επαναληπτικού αλγόριθμου.

Λύση

(α) Σύμφωνα με τον αναδρομικό αλγόριθμο, ο αριθμός μετακινήσεων που χρειάζεται για να μετακινήσουμε μία στοίβα n δίσκων στην τελική θέση, ισούται με τον αριθμό μετακινήσεων που χρειαζόμαστε για να μετακινήσουμε τους πάνω $n-1$ δίσκους στην ενδιάμεση θέση, να μετακινήσουμε τον κάτω δίσκο στην τελική θέση και να μετακινήσουμε ξανά τους $n-1$ δίσκους από την ενδιάμεση στην τελική θέση. Άρα αν με $f(n)$ ορίσουμε τον αριθμό μετακινήσεων που χρειάζεται για να λύσουμε το πρόβλημα του Hanoi με n δίσκους, τότε αναδρομικά

$$h(1) = 1$$
$$h(n) = 2h(n-1) + 1$$

Λύνοντας την αναδρομή προκύπτει η σχέση

$$h(n) = 2^n - 1$$

(β) Για να λύσουμε το πρόβλημα του Hanoi για $k+1$ δίσκους, παρατηρούμε το εξής: Για να μετακινηθεί ο κάτω δίσκος στη τελική θέση θα πρέπει να μην υπάρχει δίσκος από πάνω του και κανένας στην τελική θέση. Άρα θα πρέπει να έχουν μετακινηθεί οι k δίσκοι από πάνω του στην ενδιάμεση θέση. Άμα ο αναδρομικός αλγόριθμος για k δίσκους είναι βέλτιστος, τότε αν τον χρησιμοποιήσουμε για να μετακινήσουμε τους k πάνω δίσκους στην ενδιάμεση θέση, μετά μετακινήσουμε τον κάτω δίσκο στην τελευταία θέση και τελικά τον χρησιμοποιήσουμε ξανά για να μετακινήσουμε τους k δίσκους πάνω του, θα έχουμε τον βέλτιστο αλγόριθμο για $k+1$ δίσκους. Επειδή για $n = 1$ ο αλγόριθμος είναι προφανώς βέλτιστος, τότε επαγωγικά θα είναι βέλτιστος για κάθε n.

(γ) Για να λύσουμε το πρόβλημα του Hanoi για $k+1$ δίσκους, παρατηρούμε πως για να μετακινηθεί κάποια στιγμή ο μεγαλύτερος δίσκος στην τελική θέση, θα πρέπει όλοι οι άλλοι δίσκοι να βρίσκονται στην ενδιάμεση θέση. Έστω πως αυτό μπορεί να γίνει σε $2^k - 1$ κινήσεις με τον επαναληπτικό αλγόριθμο. Τότε ο τελευταίος δίσκος που θα κουνήθηκε για να πάει στην ενδιάμεση θέση θα είναι ο μικρότερος και σύμφωνα με τον επαναληπτικό αλγόριθμο θα πρέπει να κουνηθεί ο μεγαλύτερος δίσκος στην τελική θέση γιατί δεν μπορεί να κουνηθεί άλλος. Άμα συνεχίσουμε τον επαναληπτικό αλγόριθμο θα μας πάρει άλλες $2^k - 1$ κινήσεις και θα έχουμε λύσει το πρόβλημα σε $2^{k+1} - 1$ κινήσεις. Επειδή για $n = 1$ το πρόβλημα με τον επαναληπτικό τύπο λύνεται σε 1 κίνηση, τότε το πρόβλημα θα χρειάζεται $2^n - 1$ κινήσεις για n δίσκους με τον επαναληπτικό αλγόριθμο.

Άσκηση 2

(α) Γράψτε πρόγραμμα σε γλώσσα της επιλογής σας που να ελέγχει αν ένας αριθμός είναι πρώτος με τον έλεγχο (test) του Fermat:
Αν n πρώτος τότε για κάθε α τ.ω $1 < \alpha < n-1$, ισχύει

$$\alpha^{n-1} \bmod n = 1$$

Αν λοιπόν, δεδομένου ενός n βρεθεί α ώστε να μην ισχύει η παραπάνω ισότητα, τότε ο αριθμός n είναι οπωσδήποτε σύνθετος. Αν η ισότητα ισχύει, τότε το n είναι πρώτος με μεγάλη πιθανότητα (για τους περισσότερους αριθμούς $\geq \frac{1}{2}$). Για να αυξήσουμε σημαντικά την πιθανότητα μπορούμε να επαναλάβουμε μερικές φορές (τυπικά 40 φορές) με διαφορετικό α. Αν όλες τις φορές βρεθεί να ισχύει η παραπάνω ισότητα, τότε λέμε ότι το n 'περνάει το test' και ανακηρύσσουμε το n πρώτο αριθμό αν έστω και μία φορά αποτύχει ο έλεγχος, τότε ο αριθμός είναι σύνθετος.
Η συνάρτησή σας θα πρέπει να δουλεύει σωστά για αριθμούς έως και 1000 ψηφίων, π.χ, να μπορεί να ελέγξει αν ο αριθμός $2^x - 1$ είναι πρώτος για κάθε $x \leq 1000$. Δοκιμάστε την για $x = 100i, i \leq i \leq 10$.

Λύση

(α) Για να υλοποιήσουμε αποδοτικά τον έλεγχο, γράφουμε συνάρτηση ύψωσης με τετραγωνισμό η οποία εκμεταλλεύεται τις ιδιότητες της ομάδας $(\mathbb{Z}/n\mathbb{Z})^\times$. Παρακάτω παρατίθεται μία υλοποίηση γραμμένη σε Haskell.

```haskell
{-# LANGUAGE BangPatterns #-}

import Control.Monad
import System.Random

fastExpMod :: Integer -> Integer -> Integer -> Integer
fastExpMod b e m = fastExpMod' b e 1
  where
    fastExpMod' !b 0 !a = a
    fastExpMod' b e a
      | even e = fastExpMod' (b'^2) (e `div` 2) a
      | odd  e = fastExpMod' (b'^2) ((e-1) `div` 2) (a*b' `mod` m)
      where b' = b `mod` m

check :: Integer -> IO Bool
check n = do
  alphas <- replicateM 40 $ randomRIO (2, n-1)
  return $ all (\a -> fastExpMod a (n-1) n == 1) alphas
```

Άσκηση 3

Περιγράψτε σε ψευδοκώδικα όσο το δυνατόν πιο αποδοτικό αλγόριθμο που να δέχεται σαν είσοδο έναν γράφο (δοσμένο με μορφή πίνακα γειτνίασης) και να επιστρέφει έναν κύκλο Euler ή μία διαδρομή Euler (αν υπάρχει κάτι από τα δύο). Ποια η πολυπλοκότητα του του αλγόριθμου σας και γιατί;

Λύση

Από τον πίνακα γειτνίασης βρίσκουμε αν όλοι οι κόμβοι είναι άρτιου βαθμού ή αν υπάρχουν μόνο 2 κόμβοι περιττού βαθμού. Σε κάθε άλλη περίπτωση δεν υπάρχει κύκλος ή μονοπάτι Euler. Αυτή η διαδικασία έχει πολυπλοκότητα $O(|V|^2)$

Αν οι κόμβοι είναι όλοι άρτιου βαθμού, τότε μπορούμε αυθαίρετα να διαλέξουμε ένα κόμβο και να διαλέγουμε διαδοχικά γειτονικές ακμές που δεν υπάρχουν στο τωρινό μονοπάτι. Επειδή δε γίνεται να παγιδευτούμε σε κάποιον κόμβο αφού όλοι είναι άρτιου βαθμού, κάποια στιγμή θα επιστρέψουμε στον αρχικό κόμβο έχοντας βρει έναν κύκλο. Έπειτα μπορούμε να επαναλάβουμε τη διαδικασία για όσους κόμβους του κύκλου έχουν ακμές που δεν έχουμε περάσει και να συνδέσουμε τους κύκλους τους με τον αρχικό μας.

Αν υπάρχουν 2 κόμβοι περιττού βαθμού τότε μπορούμε να αρχίσουμε από τον ένα κόμβο και να επαναλάβουμε την ίδια διαδικασία. Η διαφορά είναι πως κάποια στιγμή θα βρεθούμε σε αδιέξοδο η οποία θα είναι το τέλος του μονοπατιού.

Όλες οι ενέργειες μπορούν να γίνουν σε $O(1)$ χρόνο π.χ με τη χρήση hash table του οποίου η δημιουργία παίρνει πάλι $O(|V|^2)$ χρόνο. Μία άλλη προσέγγιση είναι με διπλές λίστες των οποίων οι κόμβοι κρατάνε αναφορές στους γειτονικούς κόμβους των άλλων συγγενικών λιστών ώστε να μπορεί να γίνεται ταυτόχρονη διαγραφή κόμβων από όλες τις σχετικές λίστες σε O(1) χρόνο. Η δημιουργία μίας τέτοιας δομής πάλι θέλει $O(|V|^2)$ χρόνο. Άρα η συνολική πολυπλοκότητα του αλγορίθμου θα είναι $O(|V|^2)$.

Άσκηση 4

(α) Αποδείξτε την ορθότητα του αλγόριθμου Bellman-Ford όταν δεν υπάρχουν αρνητικοί κύκλοι στον γράφο.

(β) Βρείτε τι ακριβώς πρέπει να κάνει ο αλγόριθμος Bellman-Ford για να εντοπίζει αρνητικούς κύκλους. Αποδείξτε την ορθότητα της προτεινόμενος τροποποίησης.

Λύση

(α) Παρατηρούμε πως αν μία διαδρομή $u \to \cdots \to v' \to v$ είναι η μικρότερη διαδρομή από το u στο v απόστασης το πολύ k ακμών, τότε η διαδρομή $u \to v'$ θα είναι η μικρότερη διαδρομή από το u στο v' απόστασης το πολύ $k-1$ ακμών.

Έστω πως ο αλγόριθμος για κάποιο k, έχει βρει για κάθε κόμβο v απόστασης το πολύ k ακμών, τις βέλτιστες διαδρομές για τους γείτονες του, απόστασης από την αρχή το πολύ $k-1$ ακμών. Τότε διαλέγει τη βέλτιστη διαδρομή από την αρχή στο v, αφού μεταξύ όλων των βέλτιστων γειτονικών διαδρομών, αυτή θα έχει το μικρότερο άθροισμα μήκους υπό-διαδρομής και αντίστοιχης τελευταίας ακμής που συνδέεται με το v. Έτσι, επειδή ο αλγόριθμος προφανώς αρχίζει βρίσκοντας τη βέλτιστη διαδρομή για όλους τους κόμβους που απέχουν το πολύ 0 ακμές από την αρχή, επαγωγικά θα βρίσκει μετά από όλες τις επαναλήψεις τις βέλτιστες διαδρομές προς κάθε κόμβο.

(β) Μετά από $n-1$ επαναλήψεις υπό κανονικές συνθήκες ο αλγόριθμος πρέπει να έχει βρει τις βέλτιστες διαδρομές ακόμα και αν τα βάρη είναι αρνητικά. Αν όμως υπάρχει αρνητικός κύκλος, τότε για κάποιους κόμβους δε θα υπάρχει βέλτιστη διαδρομή αφού με επιπλέον περάσματα από τον κύκλο, η τιμή της θα μειώνεται διαρκώς. Έτσι ένας τρόπος να ελέγξουμε αν υπάρχει αρνητικός κύκλος είναι να κάνουμε μία επιπλέον επανάληψη στο τέλος και να δούμε αν κάποια διαδρομή μπορεί να βελτιωθεί περαιτέρω.