ΘΕΜΕΛΙΩΔΗ ΘΕΜΑΤΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

2Η ΣΕΙΡΑ ΑΣΚΗΣΕΩΝ

ΣΧΟΛΗ: ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

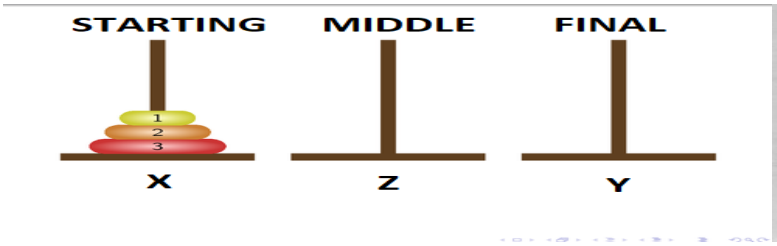
ΟΝΟΜΑ: ΑΛΚΙΒΙΑΔΗΣ ΠΑΝΑΓΙΩΤΗΣ ΕΠΙΘΕΤΟ: ΜΙΧΑΛΙΤΣΗΣ

ΑΜ: el18868 ΕΞΑΜΗΝΟ: 3Ο

1Η ΑΣΚΗΣΗ

(Αναδρομή – Επανάληψη – Επαγωγή)

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



ΑΝΑΔΡΟΜΙΚΟΣ ΑΛΓΟΡΙΘΜΟΣ (HANOI TOWERS)

```
procedure move_anoi(n from X to Y using Z)
begin
    if n = 1 then
        move top disk from X to Y
    else
        move_anoi(n-1 from X to Z using Y);
        move top disk from X to Y;
        move_anoi(n-1 from Z to Y using X)
end
```

Θέτουμε Τ(n) τον αριθμό των μετακινήσεων του παραπάνω αλγορίθμου και προκύπτει:

T(n)=2*T(n-1) +1

Αντίστοιχα

T(n-1)=2*T(n-2) + 1, T(n-2)=2*T(n-3) +1, T(n-k)=2*T(n-(k-1))+1, T(0)=0→

k=3

T(n)=2*(2*(2*T(n-3)+1)+1)+1=2^3*T(n-3) +2^2+2^1+1→

Γενικά για k προκύπτει:

T(n)= 2^k*T(n-k) + $\sum_{n=0}^{k-1} 2^n$

T(0)=0→n-k=0→n=k→ T(n)= 2^n*T(0) + $\sum_{i=0}^{n-1} 2^i$

Προκύπτει μια γεωμετρική πρόοδος η οποία ισούται με→

T(n)=2^n-1 (Συνολικός αριθμός μετακινήσεων του αλγορίθμου)

(β) Δείξτε ότι ο αριθμός κινήσεων του αναδρομικού ισούται με τον αριθμό μετακινήσεων του επαναληπτικού αλγορίθμου

<u>ΕΠΑΝΑΛΗΠΤΙΚΟΣ ΑΛΓΟΡΙΘΜΟΣ HANOI TOWERS</u>

Σύμφωνα με τον επαναληπτικό αλγόριθμο, οι μετακινήσεις των δίσκων στους πύργους Hanoi γίνεται ως εξής. Εναλλάσσονται οι κινήσεις μεταξύ του μικρότερου δίσκου και του μεγαλύτερου από αυτόν, που όμως, μπορεί εκείνη τη στιγμή να κινηθεί (βρίσκεται στην κορυφή ενός πύργου). Μετακινούμε το μικρότερο δίσκο συνεχώς στον επόμενο πύργο με την ίδια κατεύθυνση κάθε φορά: προς τα δεξιά, αν ο αρχικός αριθμός δίσκων είναι άρτιος, ενώ προς τα αριστερά αν ο αρχικός αριθμός δίσκων είναι περιττός. Αν δεν υπάρχει άλλος πύργος στην επιλεγμένη κατεύθυνση, μετακινούμε το δίσκο στο αντίθετο τέλος (με αριστερή κατεύθυνση από τον 1ο πύργο στον 3ο και με δεξιά κατεύθυνση από τον 3ο πύργο στον 1ο) και συνεχίζουμε στη σωστή κατεύθυνση που ορίσαμε πιο πριν.

Για n=1 και στους δύο αλγορίθμους έχουμε μόνο μια κίνηση.

Θα συγκρίνουμε για n=2.

<u>ΜΕ ΕΠΑΝΑΛΗΠΤΙΚΟ:</u>

nmod2=0→n=άρτιος άρα θα κινηθούμε προς τα δεξιά.

1) Ο μικρός δίσκος θα έχει μετατόπιση X→Z

2) Ο μεγάλος δίσκος θα έχει μετατόπιση X→Y

3) Τέλος ο μικρός δίσκος θα έχει μετατόπιση Z→Y

ΜΕ ΑΝΑΔΡΟΜΙΚΟ:

n ≠1→

1) move_anoi(1 from X to Z using Y); , καλείται αναδρομικά ο αλγόριθμος και για όρισμα 1 εκτελεί το if, με αποτέλεσμα τη μετακίνηση του μικρού δίσκου από τον X στον Z.

2) move top disk from X to Y; , μετακίνηση του αμέσως μεγαλύτερου δίσκου από τον X στον Y.

3) move_anoi(1 from Z to Y using X); , καλείται αναδρομικά ο αλγόριθμος και για όρισμα 1 εκτελεί το if, με αποτέλεσμα τη μετακίνηση του μικρού δίσκου από τον Z στον Y.

Παρατηρούμε ότι για n=2 κάνουν τον ίδιο αριθμό κινήσεων. Θα αποδείξουμε μέσω της επαγωγής ότι για n=k δίσκους κάνουν τον ίδιο αριθμό κινήσεων εάν δείξουμε ότι για n=k+1 κάνουν τον ίδιο αριθμό κινήσεων.

Για n=k+1:

ΑΝΑΔΡΟΜΙΚΟΣ ΑΛΓΟΡΙΘΜΟΣ

(Κινήσεις για μετακίνηση k δίσκων από ένα πόλο σε άλλο)

Έχουμε move_anoi(k+1 from X to Y using Z). Ωστόσο n=k+1 ≠1 άρα εκτελείται το else και: move_anoi(k from X to Z using Y) (Μετακίνηση k δίσκων από το X στο Z)=(Μετακίνηση k δίσκων σε άλλο πόλο με τον επαναληπτικό αλγόριθμο)=N

(Μετακίνηση του k+1^{ου} δίσκου από ένα πόλο σε ένα άλλο)

Move top disk from X to Y(μία κίνηση)

(Κινήσεις για μετακίνηση k δίσκων από ένα πόλο σε άλλο)

move_anoi(k from Z to Y using X); , μετακίνηση k δίσκων από τον Z στον Y. Από επαγωγική υπόθεση έχουμε πάλι, ότι ο αριθμός των κινήσεων (N) για την μετακίνηση των k δίσκων θα είναι ίδιος με τον αντίστοιχο του επαναληπτικού αλγορίθμου.

ΕΠΑΝΑΛΗΠΤΙΚΟΣ ΑΛΓΟΡΙΘΜΟΣ

(Κινήσεις για μετακίνηση k δίσκων από ένα πόλο σε άλλο)

Μετακίνηση k δίσκων από τον X στον Z (N κινήσεις). (Να επισημανθεί στο σημείο αυτό ότι, είτε για άρτιο, είτε για περιττό αριθμό k -με αντίστοιχη φορά μετακινήσεων- έχουμε ίδιο αριθμό κινήσεων, οπότε χωρίς βλάβη της γενικότητας συνεχίζουμε την απόδειξη).

(Κίνηση για μετακίνηση (k+1)ου δίσκου από ένα πόλο σε άλλο)

Μόνη επιτρεπτή κίνηση η μετακίνησή του από τον X στον Y (μία κίνηση).

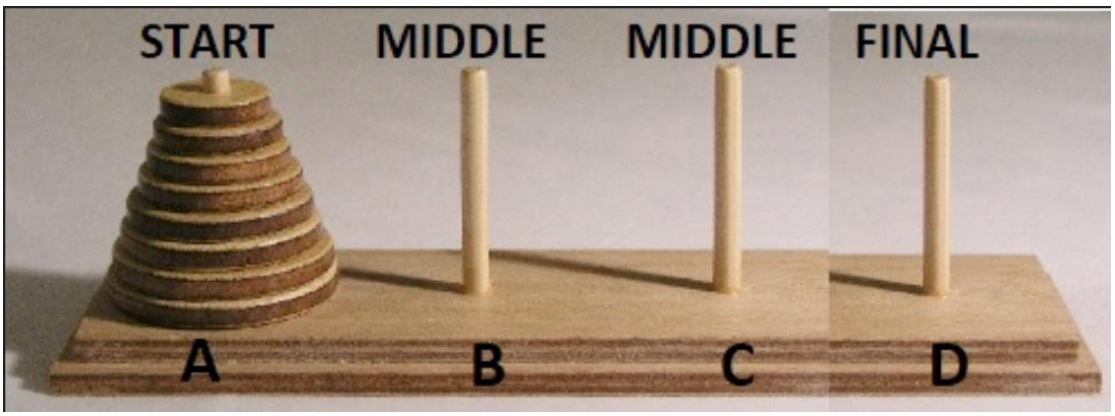
(Κινήσεις για μετακίνηση k δίσκων από ένα πόλο σε άλλο)

Μετακίνηση των k δίσκων από τον Z στον Y (N κινήσεις).

Παρατηρούμε ότι το πρώτο και το τρίτο βήμα του επαναληπτικού αλγορίθμου έχουν ίδιο πλήθος κινήσεων (2N μαζί) με τα αντίστοιχα βήματα του αναδρομικού αλγορίθμου. Επίσης το δεύτερο βήμα έχει μόνο μια μετατόπιση (ίδιο πλήθος επίσης) και για τους δύο αλγορίθμους. Άρα για n=k+1 δίσκους έχουν ίδιο αριθμό μετακινήσεων (2N + 1).

Επομένως οι δύο αλγόριθμοι (αναδρομικός και επαναληπτικός) εκτελούν ίδιο αριθμό μετακινήσεων ∀n∈N .)

(δ)* Θεωρήστε το πρόβλημα των πύργων του Hanoi με 4 αντί για 3 πασσάλους. Σχεδιάστε αλγόριθμο μετακίνησης n δίσκων από τον πάσσαλο 1 στον πάσσαλο 4 ώστε το πλήθος των βημάτων να είναι σημαντικά μικρότερο από το πλήθος των βημάτων που απαιτούνται όταν υπάρχουν μόνο 3 πάσσαλοι. Εκφράστε τον αριθμό των απαιτούμενων βημάτων σαν συνάρτηση του n.



Procedure move_anoi (n from A to D using B and C)

Begin

If (n==0)

Return;

If (n==1) move top_disk from A to D

Else

Move_anoi(n-2 from A to B using C and D)

Move n-1 disk from A to C

Move n disk from A to D

Move n-1 disk from C to D

Move_anoi(n-2 from B to D using A and C)

end

2Η ΑΣΚΗΣΗ

(Επαναλαμβανόμενος Τετραγωνισμός – Κρυπτογραφία)


Fermat


Miller


Rabin

(α) Γράψτε πρόγραμμα σε γλώσσα της επιλογής σας (θα πρέπει να υποστηρίζει πράξεις με αριθμούς 100δων ψηφίων) που να ελέγχει αν ένας αριθμός είναι πρώτος με τον έλεγχο (test) του Fermat: Αν n πρώτος τότε για κάθε a τ.ώ. 1 < a < n − 1, ισχύει a n−1 mod n = 1 Αν λοιπόν, δεδομένου ενός n βρεθεί a ώστε να μην ισχύει η παραπάνω ισότητα τότε ο αριθμός n είναι οπωσδήποτε σύνθετος. Αν η ισότητα ισχύει, τότε το n είναι πρώτος με αρκετά μεγάλη πιθανότητα (για τους περισσότερους αριθμούς ≥ 1/2). Για να αυξήσουμε σημαντικά την πιθανότητα μπορούμε να επαναλάβουμε μερικές φορές (τυπικά 30 φορές) με διαφορετικό a. 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 χιλιάδων ψηφίων. Δοκιμάστε την με τους αριθμούς: 67280421310721, 1701411834604692317316873037158841057 21, 2 2281 − 1, 2 9941 − 1, 2 19939 − 1 Σημείωση 1: το a 2 19939−2 έχει 'αστρονομικά' μεγάλο πλήθος ψηφίων (δεν χωράει να γραφτεί σε ολόκληρο το σύμπαν!), ενώ το a 2 19939−2 mod (219939 − 1) είναι σχετικά "μικρό" (έχει μερικές χιλιάδες δεκαδικά ψηφία μόνο :-)). Σημείωση 2: Υπάρχουν (λίγοι) σύνθετοι που έχουν την ιδιότητα να περνούν τον έλεγχο Fermat για κάθε a που είναι σχετικά πρώτο με το n, οπότε για αυτούς το test θα αποτύχει όσες δοκιμές και αν γίνουν (εκτός αν πετύχουμε κατά τύχη a που δεν είναι σχετικά πρώτο με το n, πράγμα αρκετά απίθανο). Αυτοί οι αριθμοί λέγονται Carmichael.(– δείτε και http://en.wikipedia.org/wiki/Carmichael_ 1 number. Ελέγξτε τη συνάρτησή σας με αρκετά μεγάλους αριθμούς Carmichael που θα βρείτε π.χ. στη σελίδα http://de.wikibooks.org/wiki/Pseudoprimzahlen:_Tabelle_Carmichael-Zahlen.)

Τι παρατηρείτε;

(β) Μελετήστε και υλοποιήστε τον έλεγχο Miller-Rabin που αποτελεί βελτίωση του ελέγχου του Fermat και δίνει σωστή απάντηση με πιθανότητα τουλάχιστον 1/2 για κάθε φυσικό αριθμό (οπότε με 30 επαναλήψεις έχουμε αμελητέα πιθανότητα λάθους για κάθε αριθμό εισόδου). Δοκιμάστε τον με διάφορους αριθμούς Carmichael. Τι παρατηρείτε;

(γ)* Γράψτε πρόγραμμα που να βρίσκει όλους τους πρώτους αριθμούς Mersenne, δηλαδή της μορφής n = 2x − 1 με 100 < x < 3000 (σημειώστε ότι αν το x δεν είναι πρώτος, ούτε το 2 x − 1 είναι πρώτος – μπορείτε να το αποδείξετε;). Αντιπαραβάλετε με όσα αναφέρονται στην ιστοσελίδα https: //www.mersenne.org/primes/.

ΛΥΣΗ

(α) Για να μπορέσω να γράψω ένα κώδικα για το συγκεκριμένο θεώρημα (αν n πρώτος(prime), τότε για κάθε α τ.ω 1<α<n-1, ισχύει α^{n-1} mod n =1),πρέπει να βρω κατάλληλο τρόπο υπολογισμού του α^{n-1} mod n, καθώς για μεγάλα α και n θα έχω πάρα πολύ μεγάλα νούμερα που θα είναι δύσκολο να υπολογιστούν.

Από τις ιδιότητες του mod γνωρίζουμε το εξής:

Παρατηρούμε ότι μια καλή λύση του αλγορίθμου μας είναι να χτίσουμε σιγά σιγά την δύναμη, δηλαδή να υπολογίσουμε τις δυνάμεις μέσω πολλαπλασιασμού, ώστε να έχω πολυπλοκότητα O(n), αντί κατευθείαν. Άρα συμπεραίνουμε ότι θα βασίζεται στον αλγόριθμο του επαναλαμβανόμενου τετραγωνισμού.(βιβλίο σελ.49-51).

Ακολουθεί ο κώδικας :

```
#include <iostream>
#include <cstdlib>
Using namespace std;
Int fastmodpower(int a, int n, int m)
{
        int res=1;
        while(n>0)
        {
                If(n%2==1)
                {
                        res=(res*a)%m;
                        n=n/2;
                        a=(a*a)%m;
                }
                Else
                {
                        n=n/2;
                        a=(a*a)%m;
                }
```

```
        }
return res;
}
Int main()
{
        Int a,n;
        bool prime=true;
        cin>>n;
        if(n==1 || n==2|| n==3)
        {
                cout<<n<<"is prime"<<endl;
        }
        Else if(n==4)
        {
                cout<<n<<"is composite"<<endl;
        }
        Else
        {
                for(int i=0;i<30;i++)
                {
                        a=2+rand()%(n-3);
                        if (fastmodpower(a,n-1,n)==1)
                        {
                                prime=true;
                                continue;
                        }
                        else
                        {
```

```
                    prime=false;
                    break;
                }
                cout<<a<<endl;
            }
            if(prime)
            {
                cout<<n<<" is prime"<<endl;
            }
            else
            {
                cout<<n<<" is composite"<<endl;
            }
        }
        return 0;
}
```

(β) Σε αυτό το ερώτημα βελτιώνουμε υο παραπάνω πρόβλημα και θα εφαρμόσουμε τον αλγόριθμο του Miller-Rabin που είναι βελτίωση του τεστ του Fermat.

Ακολουθεί ο κώδικας :

```
#include <iostream>
#include <cstdlib>
using namespace std;
int fastmodpower(int x, int y, int p)
{
    int res=1;
    x=x%p;
    while(y>0)
    {
```

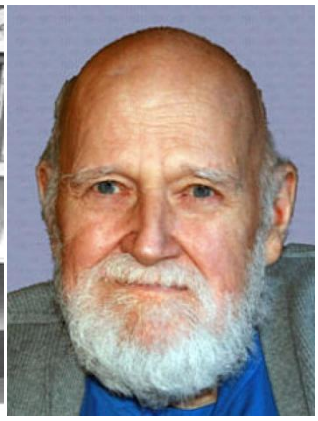
```
                if(y%2==1)
                        res = (res*x)%p;
                y=y/2;
                x=(x*x)%p;
        }
        return res;
}
bool millerTest(int d, int n)
{
        int a= 2 + rand() % (n-4);
        int x = fastmodpower(a,d,n);
        if(x==1 || x==n-1)
        {
                return true;
        }
        while (d!=n-1)
        {
                x=(x*x)%n;
                d=d*2;
                if(x==1
                        return false;
                if(x==n-1)
                        return true;
        }
        Return false;
}
```

```
Bool isPrime(int n, int k)
{
        If(n<=1 || n==4)
                Return false;
        If(n<=3)
                Return true;
        Int d=n-1;
        While(d%2==0)
                d=d/2;
        for(int i=0;i<k;i++)
                if(!millerTest(d,n))
                        return false;
                return true;
}
Int main()
{
        int k=30;
        Int n;
        cout<<"Give me a number"<<endl;
        cin>>n;
        if (isPrime(n,k))
                cout<<"The number pass the MillerRabin's test"<<endl;
        else
                cout<<"The number is not prime"<<endl;
        return 0;
}
```

3^{Η} ΑΣΚΗΣΗ

(Αλγόριθμοι Γράφων: Λιγότερα Διόδια)





Dijkstra **Bellman** **Ford**

Θεωρήστε το εξής πρόβλημα σε οδικά δίκτυα: κάθε κόμβος έχει διόδια (μια θετική ακέραια τιμή) και θέλουμε να βρεθούν οι διαδρομές με το ελάχιστο κόστος έναν αρχικό κόμβο s προς κάθε άλλο κόμβο.

Θεωρήστε ότι στον αρχικό κόμβο δεν πληρώνουμε διόδια (ενώ στον τελικό πληρώνουμε, όπως και σε κάθε ενδιάμεσο) και ότι το δίκτυο παριστάνεται σαν κατευθυνόμενος γράφος.

Περιγράψτε όσο το δυνατόν πιο αποδοτικούς αλγόριθμους για το πρόβλημα αυτό στις εξής περιπτώσεις:

(α) Το δίκτυο δεν έχει κύκλους (κατευθυνόμενους).

(β) Το δίκτυο μπορεί να έχει κύκλους.

(γ) Σε κάποιους κόμβους δίνονται προσφορές που μπορεί να είναι μεγαλύτερες από το κόστος διέλευσης (αλλά δεν υπάρχει κύκλος όπου συνολικά οι προσφορές να υπερβαίνουν το κόστος).

ΛΥΣΗ

(α) Για τον κατευθυνόμενο γράφο που παριστάνει το οδικό δίκτυο θα ισχύει ότι όλες οι ακμές που οδηγούν σε έναν κόμβο θα έχουν το ίδιο βάρος (κόστος διέλευσης από τα συγκεκριμένα διόδια). Εφόσον δεν υπάρχουν κατευθυνόμενοι κύκλοι μπορούμε να ταξινομήσουμε τοπολογικά τον γράφο. Έπειτα για να βρούμε την πιο οικονομική διαδρομή από τον αρχικό κόμβο s προς κάθε άλλο κόμβο, ξεκινώντας από την αρχή του ταξινομημένου γράφου(η οποία θεωρώντας ότι ο γράφος θα είναι έγκυρος θα είναι πάντα ο κόμβος s, αφού δε θα έχει κόστος 0), για κάθε κόμβο ενημερώνουμε το κόστος της διαδρομής προς αυτόν συγκρίνοντας για κάθε ακμή που οδηγεί στον κόμβο το κόστος της ακμής συν το ελάχιστο κόστος του κόμβου από τον οποίο προέρχεται η ακμή και κρατάμε την ακμή που δίνει το μικρότερο άθροισμα. Για κάθε κόμβο που ενημερώνουμε αποθηκεύουμε ποιος είναι ο γονέας του και με αυτόν τον τρόπο αφού περάσουμε από κάθε κόμβο έχουμε βρει την πιο φθηνή διαδρομη προς αυτόν.

(β) Αν το δίκτυο έχει κύκλους (δηλαδή δεν είναι DAG) μπορούμε να χρησιμοποιήσουμε τον αλγόριθμο του Dijkstra. Θα έχουμε ένα διάνυσμα d στο οποίο θα αποθηκεύουμε την ελάχιστη απόσταση κάθε κόμβου από την αφετηρία (αρχικά θα έχει τιμή 0 στη θέση που αντιστοιχεί στον κόμβο s και ∞ σε όλες τις υπόλοιπες), ένα σύνολο S στο οποίο θα μπαίνουν οι κόμβοι για τους οποίους έχει υπολογιστεί η ελάχιστη διαδρομή (αρχικά κενό), ουρά προτεραιότητας Q στην οποία αρχικά εισάγονται όλοι οι κόμβοι και ένα διάνυσμα στο οποίο αποθηκεύεται ποιος είναι ο γονέας κάθε κόμβου (αρχικά έχει παντού τιμή 0). Αρχικά εξάγουμε από τον Q τον πρώτο κόμβο s και τον εισάγουμε στο S. Έπειτα για κάθε γείτονά του, ενημερώνουμε το ελάχιστο κόστος (θα ταυτίζεται με το κόστος της ακμής αφού το κόστος για να φτάσουμε στον s είναι 0) και ορίζουμε ως γονέα του (prev) τον s. Ύστερα εξάγουμε από την Q τον κόμβο με το ελάχιστο διάνυσμα d(έστω x), τον εισάγουμε στο S και για κάθε γρίτονα του y που δεν ανληκει στο S αν d[y]>d[x] +[κόστος ακμής από το x στο y] τότε ενημερώνουμε το d[y] και του καταχωρούμε την τιμή d[x]+[κόστος ακμής από το x στο y] και ενημερώνουμε την τιμή prev[y] ώστε να γίνει x. Επαναλαμβάνουμε εώς ότου όλοι οι κόμοι να ανήκουν στο S και έχουμε υπολογίσει τις διαδρομές με το ελάχιστο κόστος από τον αρχικό κόμβο s προς κάθε άλλο κόμβο. Χρονική πολυπλοκότητα $O(|V^2|)$.

(γ) Εφόσον υπάρχουν αρνητικά βάρη θα προτιμήσουμε την εφαρμογή του Bellman-Ford αντί του Dijksra αλγορίθμου. Έχουμε έναν πίνακα d και έναν πίνακ prev με τις ίδιες αρχικές τιμές όπως και στο ερώτημα β. Επναλαμβάνουμε |V|-1 φορές για κάθε ακμή από έναν κόμβο u σε έναν κόμβο v αν d[u]+[κόστος ακμής από τον u στον v] και ενημερώνουμε την τιμή prev[v] ώστε να γίνει u. Με αυτό τον τρόπο έχουμε υπολογίσει τις διαδρομές με το ελάχιστο κόστος από τον αρχικό κόμβο s προς κάθε άλλο κόμβο. Χρονική πολυπλοκότητα O(|V||E|).

4^{Η} ΑΣΚΗΣΗ

(Προγραμματισμός Χριστουγεννιάτικων Διακοπών)

Μπορούμε να αναπαραστήσουμε το οδικό δίκτυο μιας χώρας ως ένα συνεκτικό μη κατευθυνόμενο γράφημα G(V, E, w) με n κορυφές και m ακμές.

Κάθε πόλη αντιστοιχεί σε μια κορυφή του γραφήματος και κάθε οδική αρτηρία σε μία ακμή.

Κάθε οδική αρτηρία e ∈ E συνδέει δύο πόλεις και έχει μήκος w(e) χιλιόμετρα.

Η ιδιαιτερότητα της συγκεκριμένης χώρας είναι ότι έχουμε σταθμούς ανεφοδιασμού σε καύσιμα μόνο στις πόλεις / κορυφές του γραφήματος, όχι στις οδικές αρτηρίες / ακμές.

Θέλουμε να ταξιδέψουμε από την πρωτεύουσα s σε ένα ορεινό θέρετρο t για τις Χριστουγεννιάτικες διακοπές μας.

Θα χρησιμοποιήσουμε το αυτοκίνητό μας που διαθέτει αυτονομία καυσίμου για L χιλιόμετρα.

(α) Να διατυπώσετε έναν αλγόριθμο, με όσο το δυνατόν μικρότερη χρονική πολυπλοκότητα, που υπολογίζει αν κάτι τέτοιο είναι εφικτό. Ποια είναι η χρονική πολυπλοκότητα του αλγορίθμου σας στη χειρότερη περίπτωση;

(β) Να διατυπώσετε αλγόριθμο με χρονική πολυπλοκότητα (m log m) που υπολογίζει την ελάχιστη αυτονομία καυσίμου (σε χιλιόμετρα) που απαιτείται για το ταξίδι από την πόλη s στην πόλη t.

(γ*) Να διατυπώσετε αλγόριθμο με γραμμική χρονική πολυπλοκότητα που υπολογίζει την ελάχιστη αυτονομία καυσίμου για το ταξίδι από την πόλη s στην πόλη t. Υπόδειξη: Εδώ μπορεί να σας φανεί χρήσιμο ότι (με λογική αντίστοιχη με αυτή της Quicksort) μπορούμε να υπολογίσουμε τον median ενός μη ταξινομημένου πίνακα σε γραμμικό χρόνο.

ΛΥΣΗ

(α) Με m επαναλήψεις ελέγχουμε για κάθε ακμή αν το w(e) είανι μεγααλύτερο του L. Εάν είναι, τότε το αυτοκίνητό μας δεν μπορεί να διασχίσει αυτή την ακμή και συνεπώς την αφαιρούμε. Έπειτα εφαρμόζοντας DFS στο γράφο που προκύπτει, με αρχικό κόμβο την πρωτεύουσα s ελέγχουμε αν υπάρχει μονοπάτι που συνδέει την s με το ορεινό θέρετρο t (αν κατά τη διάρκεια της DFS διάσχισης εμφανιστεί ο κόμβος t τότε υπάρχει, αλλιώς δεν υπάρχει). Για να ελέγξουμε όλες τις ακμές (για να διαπιστώσουμε αν w(e)>L) απαιτείται χρόνος O(m) ενώ για την διάσχιση απαιτείται χρόνος O(n+m). Άρα η χρονική πολυπλοκότητα του αλγορίθμου στην χειρότερη περίπτωση θα είναι O(n+m).

(β) Για να υπολογίσουμε την ελάχιστη αυτονομία καυσίμου που απαιτείται για το ταξίδι μπορούμε να εφαρμόσουμε μία τροποποιημένη μορφή του Dijkstra στον αρχικό γράφο. Έχοντας τον Dijkstra όπως αυτός περιγράφηκε στο ερώτημα 3β, αν όταν εντοπίζει για τους γείτονες y κάποιου κόμβου x αντί να ελέγχει αν d[y]>d[x]+[κόστος ακμής από το x στο y], θα ελέγχει αν d[y]>max(d[x], [κόστος ακμής από το x στο y]) και θα ενημερώνει την τιμη d[y] και θα την κάνει max(d[x],[κόστος ακμής από το x στο y]). Αυτό μας βολεύει εφόσον σε κάθε πόλη το αυτοκίνητο ανεφοδιάζεται. Αν η ουρά του Dijkstra υλοποιηθεί με δυαδικό σωρό τότε ο αλγόριθμος θα έχει πολυπλοκότητα O(mlogn)≈O(mlogm) αφού $m<=n^2$.

5Η ΑΣΚΗΣΗ

(Αγορά Εισιτηρίων)

Εκτός από τις Χριστουγεννιάτικες διακοπές σας, έχετε προγραμματίσει προσεκτικά και τις σπουδές σας.

Συγκεκριμένα, έχετε σημειώσει ποιες από τις επόμενες T ημέρες του εξαμήνου θα έρθετε στο ΕΜΠ για να παρακολουθήσετε μαθήματα και εργαστήρια.

Το πρόγραμμά σας έχει τη μορφή ενός πίνακα S με T θέσεις, όπου για κάθε ημέρα t = 1, . . . , T, S[t] = 1, αν θα έρθετε στο ΕΜΠ, και S[t] = 0, διαφορετικά. Το πρόγραμμά σας δεν έχει κανονικότητα και επηρεάζεται από διάφορες υποχρεώσεις και γεγονότα.

Για κάθε μέρα που θα έρθετε στο ΕΜΠ, πρέπει να αγοράσετε εισιτήριο που να επιτρέπει τη μετακίνησή σας με τις αστικές συγκοινωνίες. Υπάρχουν συνολικά k διαφορετικοί τύποι εισιτηρίων (π.χ., ημερήσιο, τριών ημερών, εβδομαδιαίο, μηναίο, εξαμηνιαίο, ετήσιο).

Ο τύπος εισιτηρίου i σας επιτρέπει να μετακινηθείτε για ci διαδοχικές ημέρες (μπορεί βέβαια κάποιες από αυτές να μην χρειάζεται να έρθετε στο ΕΜΠ) και

κοστίζει pi ευρώ. Για τους τύπους των εισιτηρίων, ισχύει ότι 1 = c1 < c2 < · · · < ck ≤ T, p1 < p2 < · · · < pk, και p1/c1 > p2/c2 > · · · > pk/ck (δηλαδή, η τιμή αυξάνεται με τη διάρκεια του εισιτηρίου, αλλά η τιμή ανά ημέρα μειώνεται).

Να διατυπώσετε αλγόριθμο που υπολογίζει τον συνδυασμό τύπων εισιτηρίων με ελάχιστο συνολικό κόστος που καλύπτουν όλες τις ημέρες που θα έρθετε στο ΕΜΠ.

Ποια είναι η χρονική πολυπλοκότητα του αλγορίθμου σας στη χειρότερη περίπτωση;

<u>ΛΥΣΗ</u>

Αρχικά υποθέτουμε ότι έχουμε βρεί το βέλτιστο κόστος εισιτηρίων για κάθε μέρα εώς τη μέρα t(συμβ.T[t]). Το κόστος για κάθε επόμενη μέρα x με S[x] = 0, εώς 'οτου να συναντήσω κάποια μέρα y με S[y] = 1, ισούνται με T[x] = T[t]. Το T[y] θα ισούται με το ελάχιστο μεταξύ k επιλογών. Οι επιλογές αυτές είναι οι εξής:

1) Να αγοράσουμε εισιτήρια έως τη μέρα y-1 και να επιλέξουμε εισιτήριο μίας ημέρας.

$T[y] = T[y-1] + p_1$

2) Να αγοράσουμε εισιτήρια έως τη μέρα $y - c_2$ και να επιλέξουμε εισιτήριο c ημερών.

$T[y] = T[y-c_2] + p_2$

Και συνεχίζω ομοίως έως την k-οστή επιλογή:

k) Να αγοράσουμε εισιτήρια έως τη μέρα $y - c_k$ και να επιλέξουμε εισιτήριο c_k ημερών.

$T[y] = T[y-c_k] + p_k$

Σε περίπτωση που $y - c_k < 0$ για κάποιο y και c_k τότε δεν λαμβάνω υπόψιν αυτή την επιλογή.

Αν y = 1 ή y = 0 τότε T[y] = y.

Χρησιμοποιώντας T δομές container, μία για κάθε ημέρα t, όταν υπολογίσουμε το T[t], αποθηκεύουμε τον τύπο εισιτηρίων που επιλέξαμε μέχρι την ημέρα j, συν το εισιτήριο $c_k = n - j$ ημερών που επιτυγχάνει το ελάχιστο δυνατό κόστος T(t). Το output του προβλήματος θα αποτελούν τα περιεχόμενα του container που αποθηκεύει τα εισητήρια της τελευταίας ημέρας του εξαμήνου.

Η πολυπλοκότητα του αλγορίθμου είναι της τάξης O(T*K) όπου k ο αριθμός των εισιτηρίων και T το πλήθος των ημερών του εξαμήνου.

6Η ΑΣΚΗΣΗ

(Δυναμικός Προγραμματισμός: Αντοχή Ποτηριών)

Σε ένα εργοστάσιο υαλικών θέλουν να μετρήσουν την αντοχή των κρυστάλλινων ποτηριών που κατασκευάζουν. Συγκεκριμένα, θέλουν να βρουν ποιο ακριβώς (με ακρίβεια εκατοστού) είναι το μέγιστο ύψος από το οποίο μπορούν να πέσουν τα ποτήρια τους χωρίς να σπάσουν. Η διεύθυνση του εργοστασίου μπορεί να διαθέσει έως ένα πλήθος ποτηριών *k* για τις δοκιμές (δηλ. δέχεται να καταστραφούν *k* ποτήρια) και θέλει το μέγιστο πλήθος δοκιμών που θα χρειαστούν να είναι όσο το δυνατόν μικρότερο. Επιπλέον, είναι γνωστό ότι τα ποτήρια σίγουρα σπάνε από ένα δεδομένο ύψος *n* εκατοστών και πάνω και ότι όλα τα ποτήρια είναι της ίδιας ακριβώς αντοχής. Με βάση τα παραπάνω δεδομένα σχεδιάστε αλγόριθμο που να βρίσκει την βέλτιστη σειρά δοκιμών, με είσοδο τα *n* και *k* έτσι ώστε να ελαχιστοποιείται το πλήθος των δοκιμών στη χειρότερη περίπτωση. Ειδικότερα: (α) Βρείτε την βέλτιστη σειρά δοκιμών για *n* = 100 και *k* = 1, καθώς και για *n* = 100 και *k* = 2. Ποιό είναι το μέγιστο πλήθος δοκιμών που μπορεί να χρειαστεί η λύση σας; Γενικεύστε για οποιοδήποτε *n* και *k* = 2. (β) Γενικεύστε για οποιαδήποτε *n* και *k* (*υπόδειξη*: προσπαθήστε αρχικά να εκφράσετε τη βέλτιστη λύση της περίπτωσης οποιουδήποτε *n* και *k* = 2 αναδρομικά, χρησιμοποιώντας λύσεις της περίπτωσης *k* = 1, και της περίπτωσης *k* = 2 για μικρότερα *n*).Ποια είναι η πολυπλοκότητα του αλγορίθμου σας;

ΛΥΣΗ

(α) Θα εργαστούμε αρχικά με την περίπτωση για n=100 εκατοστά, k=1 ποτήρι.

Με μια τεχνική όπως η δυαδική αναζήτηση(binary search) που είναι μια λογική σκέψη, μπορεί το ποτήρι να σπάσει πριν βρούμε το ζητούμενο μέγιστο δυνατό ύψος (δηλαδή πιθανότατα χρειαζόμαστε και άλλα ποτήρια).

Η μόνη λύση με το 1 ποτήρι είναι να ξεκινήσουμε από το χαμηλότερο ύψος και ανά 1 εκατοστό να δοκιμάζουμε αν το ποτήρι θα σπάσει. Άρα, θα έχουμε δοκιμή στο 1ο εκατοστό, στο 2ο κ.ο.κ. μέχρι το ποτήρι τελικά να σπάσει. Έτσι θα έχουμε τη σωστή απάντηση χρησιμοποιώντας το ποτήρι που μας δίνεται, όμως μπορεί να χρειαστεί να δοκιμάσουμε n φορές με πολυπλοκότητα, επομένως, O(n).

Τώρα, θα εργαστούμε με την περίπτωση για n=100 εκατοστά, k=2 ποτήρια.

Αφού θα έχουμε πλέον δύο(2) ποτήρια αντί για ένα(1) όπως πριν, θα πρέπει να χειριστούμε το πρόβλημα πιο αποδοτικά.

Θα πρέπει να περιγράψουμε μια στρατηγική για εύρεση του μεγίστου δυνατού ύψους που απαιτεί το πολύ f(n) δοκιμές, για μια συνάρτηση f(n) που αυξάνει πιο αργά από τη γραμμική (:πολυπλοκότητα της αμέσως προηγούμενης περίπτωσης, δηλαδή k=1). Άρα, θα πρέπει

$$\lim_{n\to\infty} \frac{f(n)}{n} = 0.$$

Μια καλή στρατηγική θα είναι να ρίξουμε το πρώτο ποτήρι από ύψος που είναι πολλαπλάσιο του $\sqrt{n}$. Αν φτάσουμε την κορυφή (n) και το ποτήρι δε σπάσει, τότε θα έχουμε τελειώσει με 10=$\sqrt{100}$= $\sqrt{n}$ προσπάθειες ($\sqrt{n}$ γιατί το n=100 στην περίπτωση αυτή είναι τέλειο τετράγωνο). Αν το ποτήρι σπάσει σε ρίψη από ύψος λ• $\sqrt{n}$, τότε ξέρουμε ότι το ζητούμενο μέγιστο δυνατό ύψος θα βρίσκεται μεταξύ των (λ-1) • $\sqrt{n}$ και λ• $\sqrt{n}$. Στο σημείο αυτό ξεκινάμε να ρίχνουμε το δεύτερο ποτήρι από το ύψος (λ-1) • $\sqrt{n}$ + 1 εκατοστών, ανεβαίνοντας 1 εκατοστό σε κάθε δοκιμή. Αυτό θα πάρει το πολύ $\sqrt{n} - 1$ βήματα. Συνεπώς, στη χειρότερη περίπτωση, ο αλγόριθμος θα βρει το ζητούμενο μέγιστο ύψος σε το πολύ 2 $\sqrt{n}$ -1 ρίψεις, με πολυπλοκότητα, επομένως O($\sqrt{n}$). Είναι πράγματι $\lim_{n\to\infty} \frac{\sqrt{n}}{n} = 0.$
(Σημείωση: αν το n δεν ήταν τέλειο τετράγωνο, τότε χρησιμοποιώντας την ίδια στρατηγική διατηρούμε πάλι αλγόριθμο που τρέχει σε O($\sqrt{n}$))

(β) Τώρα θα εργαστούμε για την περίπτωση οποιωνδήποτε n,k.

- Εάν k=1 ή k=2, εκτελούμε τη στρατηγική του α ερωτήματος.
- Εάν k>2, ξεκινάμε να ρίχνουμε το πρώτο ποτήρι από ύψη που είναι πολλαπλάσια του $n^{\frac{k-1}{k}}$. Παρατηρούμε ότι θα ρίξουμε το πρώτο ποτήρι το πολύ n/ $n^{\frac{k-1}{k}}$=$n^{\frac{1}{k}}$ φορές πριν σπάσει. Αν το ποτήρι σπάσει σε ύψος λ* $n^{\frac{k-1}{k}}$ ξέρουμε ότι το ζητούμενο μέγιστο δυνατό ύψος θα βρίσκεται μεταξύ των (λ-1)* $n^{\frac{k-1}{k}}$ και λ* $n^{\frac{k-1}{k}}$. Επαναλαμβάνουμε την ίδια τεχνική για τα εναπομείναντα (κ-1) ποτήρια στο διάστημα [(λ-1)* $n^{\frac{k-1}{k}}$,λ* $n^{\frac{k-1}{k}}$].

Θα αποδείξουμε με χρήση της μαθηματικής επαγωγής ότι $f_k(n) \le k * n^{\frac{1}{k}}$.

Βάση επαγωγής: Εάν k=2 αποδείξαμε στο α ερώτημα ότι $f_2(n) \le 2 * n^{\frac{1}{2}}$

Επαγωγική υπόθεση: Υποθέτουμε ότι $f_{n-1}(n) \le (k-1) * n^{\frac{1}{\kappa-1}}$.

Όπως παρατηρήσαμε πριν, ξέρουμε ότι θα ρίξουμε το πρώτο ποτήρι το

πολύ $n^{\frac{1}{k}}$ φορές. Τώρα χρειάζεται να τρέξουμε τον αλγόριθμο για τα εναπομείναντα k-1 ποτήρια. Αυτό θα χρειαστεί $f_{k-1}(n^{\frac{k-1}{k}})$, αφού υπάρχουν $n^{\frac{k-1}{k}}$ εκατοστά στο διάστημα [(λ-1)* $n^{\frac{k-1}{k}}$, λ*$n^{\frac{k-1}{k}}$]. Επομένως, από επαγωγική υπόθεση, θα χρειαστεί λιγότερο από (k-1)* $\left(n^{\frac{k-1}{k}}\right)^{\frac{1}{k-1}}$=(k-1)* $n^{\frac{1}{k}}$.

Άρα, ο συνολικός αλγόριθμος θα χρειαστεί λιγότερο από $n^{\frac{1}{k}}$ + (k-1)* $n^{\frac{1}{k}} = k* n^{\frac{1}{k}}$ ρίψεις.

Επομένως, $f_k(n) \leq k* n^{\frac{1}{k}}$.

Συνεπώς, ο αλγόριθμος θα τρέχει με πολυπλοκότητα $O(n^{\frac{1}{k}})$.