

Lucrare avizată ca accesoriu curricular
de Ministerul Educației și Cercetării cu nr. 38905/29.10.2002

Emanuela Cerchez – absolventă a Facultății de Matematică, secția Informatică (1990), și a Seminarului Pedagogic Postuniversitar (1997); profesor de informatică (grad didactic I) la Liceul de Informatică „Grigore C. Moisil”, Iași. Membru în Comisia Națională de Informatică. Participă la elaborarea subiectelor și evaluarea elevilor la competițiile locale și naționale de informatică. Desfășoară activități de pregătire de performanță a elevilor la Centrul de Excelență pentru Tineri Capabili de Performanță, Iași, și în taberele naționale de pregătire și selecție ale lotului național de informatică. De același autor, la editura Polirom: *Informatică. Manual pentru clasa a X-a*, Iași, 2000; *Internet. Manual optional pentru liceu*, Iași, 2000; *PC pas cu pas*, Iași, 2001; *Informatica pentru gimnaziu*, Iași, 2002.

Emanuela Cerchez

INFORMATICA

Culegere de probleme pentru liceu

© 2002 by Editura POLIROM

www.polirom.ro

Editura POLIROM
Iași, B-dul Copou nr. 4; P.O. BOX 266, 6600
București, B-dul I.C. Brătianu nr. 6, et. 7, ap. 33; O.P. 37, P.O. BOX 1-728, 70700

Descrierea CIP a Bibliotecii Naționale a României:

CERCHEZ, EMANUELA

Informatica. Culegere de probleme pentru liceu / Emanuela Cerchez – Iași: Polirom, 2002
240 p., 24 cm

ISBN: 973-681-088-7

004(075.35)(076)

Printed in ROMANIA

Polirom
2002

Cuprins

<i>Cuvânt înainte</i>	7
Algoritmi elementari	9
<i>Probleme propuse</i>	21
Structuri elementare de date	26
<i>Probleme propuse</i>	54
Subprograme. Recursivitate	65
<i>Probleme propuse</i>	91
Metoda backtracking	100
<i>Prezentare generală</i>	100
<i>Probleme propuse</i>	112
Metoda programării dinamice	121
<i>Prezentare generală</i>	121
<i>Probleme propuse</i>	148
Algoritmi pe grafuri	164
<i>Probleme propuse</i>	198
Structuri arborescente	207
<i>Probleme propuse</i>	230
Bibliografie	237

Cuvânt înainte

Informatica este știința care are ca obiect de studiu reprezentarea și organizarea informației și algoritmii de prelucrare a informației.

Calitatea de *informatician* presupune creativitate, autonomie în gândire, responsabilitate.

Această carte își propune ca, într-o anumită măsură, să contribuie la formarea viitorilor informaticieni. Pentru că cea mai bună metodă de a forma gândirea algoritmică este proiectarea și valorificarea strategiilor de rezolvare a problemelor.

Este un material bun? Toate problemele sunt lucrate cu elevii, la orele de clasă, la pregătirea pentru concursuri sau în cadrul Centrului de Excelență pentru Tineri Capabili de Performanță, Iași. Răspunsul la această întrebare ar trebui să se deducă din rezultatele lor, atât cele școlare, dar mai ales cele de informaticieni profesioniști.

Culegerea cuprinde 95 de probleme rezolvate și 115 probleme propuse spre rezolvare, grupate în 7 capitulo tematice:

- „Algoritmi elementari”, în care se realizează prelucrări simple asupra datelor, fără o organizare a acestora în structuri de date;
- „Structuri elementare de date”, în care se utilizează structurile de date statice fundamentale (tabloul, șirul de caractere, articolul);
- „Subprograme și recursivitate”, în care accentul este plasat pe dezvoltarea modulară a aplicațiilor;
- „Metoda *backtracking*”, în care sunt rezolvate și propuse probleme care necesită aplicarea acestei metode;
- „Metoda programării dinamice”, în care este realizată o prezentare a metodei și aplicarea acesteia pentru rezolvarea a numeroase probleme;
- „Algoritmi pe grafuri”, în care sunt rezolvate și propuse spre rezolvare probleme care vizează algoritmii fundamentali pentru grafuri orientate și neorientate (parcursarea grafurilor, conexitate, probleme de drum optim, probleme de ordonanțare temporală etc.);
- „Structuri arborescente”, în care sunt prezentate câteva aplicații ale acestei categorii speciale de grafuri.

Implementarea soluțiilor este realizată în limbajul C++.

În cadrul fiecărui capitol, problemele sunt introduse gradat, în funcție de dificultate. Unele sunt probleme „clasice”, dar cele mai numeroase sunt probleme deosebite, care au fost utilizate la diferite concursuri și olimpiade: concursuri ale palatelor copiilor (PACO și Lugoj), concursuri regionale, Olimpiada Județeană (OJ), Olimpiada Națională (ONI), Olimpiada Națională pentru Gimnaziu (ONIG), Balcaniada de Informatică (BOI), Concursul de Informatică al Europei Centrale și de Est (CEOI), Olimpiada Internațională (IOI) etc. În dreptul fiecărei probleme sunt precizate concursul și clasa la care a fost prezentată, atât din fair play, cât și cu scopul de a ascuții spiritul de competiție al rezolvatorilor. Multe dintre probleme au fost propuse de autor, iar altele au fost propuse de profesori de prestigiu sau studenți olimpici: Rodica Pintea, Dana Lica, Maria și Adrian Niță, Emil Onea, Dan Grigoriu, Doru Popescu Anastasiu, Moț Nistor, Stelian Ciurea,

Ovidiu Domşa, Mugurel Andreica, Mihai Pătraşcu, Cătălin Frâncu, Mihai Stroe şi, bineînţeles, Marinel Şerban.

Acest material se datorează elevilor, care constituie o permanentă „provocare”. Dar, în aceeaşi măsură, se datorează acestei colaborări vitale, care îmi creionează orizontul performanţei.

Algoritmi elementari

1. Paşi de broască

O broscuţă se deplasează, efectuând câte o săritură de lungime p centimetri la fiecare secundă. După fiecare n secunde broscuţă devine mai obosită, iar lungimea săriturii pe care o face se înjumătăşează.

Scrieţi un program care să citească de la tastatură lungimea iniţială a săriturii (p), numărul de secunde după care broscuţă îşi înjumătăşează saltul (n), precum şi durata totală a deplasării broscuţei T (exprimată în număr de secunde) şi care să determine distanţa totală pe care a parcurs-o broscuţă. Distanţa totală determinată va fi afişată pe ecran, cu două zecimale.

(ONIG, Gălăciuc, 2001, clasa a V-a)

Restricţii

- p, n, T sunt numere naturale; $p, n, T < 30000$;
- $T/n < 16$.

Exemplu

Pentru $n=10$, $p=20$ şi $T=33$, distanţa totală pe care se deplasează broscuţă este 35750 cm.

Soluţie

O primă soluţie este de a simula mişcarea broscuţei, adunând $n \cdot p$ cm la distanţa parcursă de broscuţă la fiecare n secunde şi apoi înjumătăţind lungimea saltului p .

```
#include <iostream.h>
void main()
{ int n, T;
  float D=0, p;
  cout<<"n, p, T ="; cin>>n>>p>>T;
  while (T>=n)
    { D+=n*p; p/=2; T-=n; }
  D+=T*p;
  cout<<"Distanţa totală este: "<<D; }
```

O altă soluţie, mai scurtă, este de a utiliza o formulă de calcul. În primul rând, determinăm de câte ori îşi înjumătăşează broscuţă lungimea saltului: $n_j = T/n$. În primele n secunde, broscuţă parcurge $n \cdot p$ cm. În următoarele n secunde, parcurge $n \cdot p/2$ cm, apoi $n \cdot p/2^2$ cm ş.a.m.d. Cum de obicei T nu este divizibil cu n , la sfârşit broscuţă se mai deplasează timp de $\text{rest} = T \bmod n$ (T%n) secunde făcând salturi de lungime $p/2^{n_j}$. Prin urmare, distanţa totală parcursă de broscuţă este:

$$n \cdot p + n \cdot p/2 + n \cdot p/2^2 + \dots + n \cdot p/2^{n_j-1} + \text{rest} \cdot p/2^{n_j}$$

Scoțând în factor $n \cdot p$ și determinând suma progresiei geometrice $1 + 1/2 + 1/2^2 + \dots + 1/2^{n_j-1}$, obținem: $n \cdot p \cdot (2^{n_j} - 1) / 2^{n_j-1} + \text{rest} \cdot p / 2^{n_j}$

Datorită restricției $T/n < 16$, deducem că putem calcula 2^{n_j} utilizând operatorii de deplasare.

```
#include <iostream.h>
void main()
{ int n, p, T, nj, rest, dnj;
cout<<"n, p, T="; cin>>n>>p>>T;
nj=T/n; rest=T%n; dnj=1<<nj;
cout<<"Distanța totală este:";
cout<<p*(n*2*(float)(dnj-1)/dnj+(float)rest/dnj); }
```

2. Sir

Să considerăm următorul sir de numere naturale:

7, 17, 37, 47, 67, 97, 107, 127, 137, 157, 167, ...

Deducreți regula după care sunt generați termenii sirului și scrieți un program care să citească de la tastatură un număr natural N și să afișeze pe ecran al N-lea termen din sirul de mai sus.

(Concurs Lugoj, 2001, clasele a V-a – a VI-a)

Restricții

- $0 < N \leq 2000$

Soluție

Sirul este format din numerele prime care au ultima cifră egală cu 7, în ordine crescătoare. Prin urmare, trebuie să determinăm cel de-al N-lea număr prim cu ultima cifră egală cu 7. Pentru aceasta, vom parcurge mulțimea numerelor naturale, începând cu 7, din 10 în 10 (acestea sunt numerele care au ultima cifră 7). Pentru fiecare astfel de număr vom testa dacă este sau nu număr prim. Algoritmul se termină după ce am depistat al N-lea număr prim.

```
#include<iostream.h>
#include<math.h>
void main()
{ long N, d, x, nr=7, rad, rang;
cout<<"N= "; cin>>N;
for (x=7, rang=1; rang<=N; x+=10)
{ for (d=2, rad=sqrt(x); d<=rad && x%d; d++);
if (x%d) rang++;
if (x==nr) cout<<nr<<endl; }
```

3. Virus

La laboratorul de cercetări genetice din Lugoj, în urma unor experiente nereusite, un virus a suferit mutații. Ca urmare, există trei tipuri de virusi: virusul inițial (îl vom numi

virus de tip A) și două tipuri de virusi mutanți (îl vom numi virusi de tip B și virusi de tip C). Pentru a studia comportamentul virusilor, cercetătorii au izolat într-un mediu steril un virus de tip A, un virus de tip B și un virus de tip C și au observat că la fiecare secundă se produc următoarele transformări:

- Orice virus de tip A se divide și se obțin un nou virus de tip A, un nou virus de tip B și un nou virus de tip C.
- dintr-un virus de tip B și un virus de tip C se obțin trei noi virusi (unul de tip C și doi de tip B).

De asemenea, cercetătorii au observat că virusii nu mor.

Scrieți un program care să determine numărul de virusi de fiecare tip, existenți după n secunde.

(Concurs Lugoj, 2001, clasele a V-a – a VI-a)

Restricții

- $n \leq 25$

Exemple

Pentru $n=1$, pe ecran se va afișa 2A 4B 3C

Pentru $n=2$, pe ecran se va afișa 4A 12B 8C

Soluție

Vom simula evoluția populației de virusi timp de n secunde, determinând la fiecare pas numărul de virusi existenți din fiecare tip:

```
#include <iostream.h>
void main()
{ int n, i;
long nra=1, nrb=1, nrc=1;
cout<<"n= "; cin>>n;
for (i=1; i<=n; nra*=2, i++)
{ nrb=nrb+2*nrc+nra;
nrc=2*nrc+nra;}
cout<<nra<<"A "<<nrb<<"B "<<nrc<<"C"; }
```

4. Cercetași

În zona Vrancea există două grupuri de cercetași, cu sediul în două regiuni (Gălăciuc și Soveja). Cercetașii din Gălăciuc sunt organizați în D1 deașamente a căte N1 cercetași fiecare. Cercetașii din Soveja sunt organizați în D2 deașamente, a căte N2 cercetași fiecare. În urma unui ordin de la Organizația Europeană a Cercetașilor, trebuie ca toate deașamentele din zona Vrancea să fie formate din același număr de cercetași. Pentru îndeplinirea ordinului, trebuie reorganizate deașamentele din fiecare regiune, fără a deplasa cercetași dintr-o regiune în alta.

Scrieți un program care să citească de la tastatură D1, N1, D2 și N2 și care să determine și să afișeze pe ecran:

- numărul de cercetași din fiecare deașament după reorganizare, cât mai mare posibil;

- câte detașamente se formează în zona Gălăciuc și câte detașamente se formează în zona Soveja.

(ONIG, Gălăciuc, 2001, clasa a V-a)

Restrictii

- D1, N1, D2 și N2 sunt numere naturale; D1, N1, D2 și N2 < 1000;

Exemplu

Pentru D1=3, N1=15, D2=2 și N2=18 veți afișa pe ecran:

Nr. de cercetași din fiecare detașament: 9

Nr. de detașamente din Galaciuc: 5

Nr. de detașamente din Soveja: 4,

Soluție

Determinăm numărul total de cercetași din zona Gălăciuc ($T_1=D_1 \cdot N_1$), respectiv numărul total de cercetași din zona Soveja ($T_2=D_2 \cdot N_2$). Numărul de cercetași din fiecare detașament după reorganizare trebuie să fie un divizor atât pentru T_1 , cât și pentru T_2 și în plus să fie cât mai mare posibil. Prin urmare, acesta este c.m.m.d.c. (T_1, T_2).

```
#include <iostream.h>
void main()
{
    long int N1, D1, N2, D2, T1, T2, rest;
    cout<<"N1, D1, N2, D2 = "; cin>>N1>>D1>>N2>>D2;
    T1=N1*D1; T2=N2*D2;
    while (T2)
        { rest=T1%T2; T1=T2; T2=rest; }
    cout<<"Nr. de cercetași din fiecare detașament:"<<T1<<endl;
    cout<<"Nr. de detașamente din Galaciuc: "<<N1*D1/T1<<endl;
    cout<<"Nr. de detașamente din Soveja: "<<N2*D2/T1; }
```

5. Inscriptibilitate

Să considerăm un cerc de rază R. În acest cerc putem înscrie un pătrat, apoi în pătrat putem înscrie un cerc și.a.m.d.

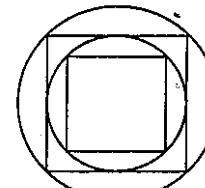
Scriți un program care să citească de la tastatură un număr real pozitiv R (raza cercului inițial) și un număr natural nenul n ($n < 100$), care reprezintă numărul de perechi cerc-pătrat care se construiesc și care afișează pe ecran n perechi de valori ce reprezintă, în ordine, raza unui cerc și latura pătratului înscris în el.

Exemplu

Pentru R=100 și n=2 (ca în figură) programul va afișa pe ecran:

R= 100 L= 141.42

R= 70.71 L= 100



Soluție

Să considerăm un cerc de rază R și un pătrat înscris în el. Să notăm cu L lungimea laturii pătratului înscris în cerc. Observăm că diagonală pătratului este egală cu diametrul cercului. Prin urmare $L\sqrt{2}=2R \Rightarrow L=R\sqrt{2}$.

În continuare, observăm că lungimea diametrului cercului înscris în pătrat este egală cu lungimea laturii pătratului. Prin urmare, pentru pasul următor $R=L/2$.

```
#include <iostream.h>
#include <iomanip.h>
#include <math.h>
void main()
{
    float R, L, rad=sqrt(2);
    int i, n;
    cout<<"R, n= "; cin>>R>>n;
    for (i=0; i<n; i++)
        { L=R*rad;
        cout<<setprecision(2)<<"R= "<<R<<" L= "<<L<<endl;
        R=L/2; }
```

6. Cod corect

Serviciul de pază al unei firme are nevoie de un program care să verifice corectitudinea codului de pe cartelele de identificare a angajaților. Codul este un număr întreg de maxim 9 cifre, care conține cel puțin o cifră pară și una impară și pentru care suma tuturor cifrelor impare și produsul tuturor cifrelor pare trebuie obligatoriu să producă același rest la împărțirea cu prima cifră a codului (în ordinea de la stânga la dreapta). Scrieți un program care să citească de la tastatură un cod și să afișeze pe ecran mesajul CORECT sau INCORRECT, în funcție de situație.

Exemplu

Pentru n=253271 programul va afișa mesajul CORECT (produsul cifrelor pare este 4, suma cifrelor impare este 16, deci restul împărțirii lui 4, respectiv 16 la 2, prima cifră a numărului, este 0).

Soluție

Pentru a valida un cod, trebuie să „spargem” în cifre numărul întreg respectiv. Mai exact, vom extrage succesiv cifrele numărului, începând cu cifra unităților, calculând pe parcurs suma cifrelor impare și produsul cifrelor pare. Pentru a determina ultima cifră din număr (cifra unităților) vom calcula restul împărțirii numărului la 10. După prelucrarea ultimei cifre, o vom elimina din număr (împărțind numărul la 10) și vom relua procedeul pentru cifrele rămase, până la epuizarea tuturor cifrelor.

```
#include <iostream.h>
void main()
{
    long n, s=0, p=1;
    int cifra;
    cout<<"n= "; cin>>n;
```

```

do
{cifra=n%10;           //determin ultima cifra
 if (cifra%2) s+=cifra; //cifra este impara
 else      p*=cifra;   //cifra este para
 n/=10; }               //elimin ultima cifra
while (n);
if (s%cifra == p%cifra) cout<<"CORECT";
else cout<<"INCORECT"; }

```

7. Progresie aritmetică

Se citește de la tastatură o succesiune de cel puțin două numere naturale, până la întâlnirea valorii -1. Să se verifice dacă numerele naturale citite constituie, în ordine, termeni consecutivi ai unei progresii aritmetice.

Observație

Numim *progresie aritmetică* o succesiune de numere a_1, a_2, \dots, a_n astfel încât diferența dintre oricare doi termeni consecutivi este aceeași (această diferență se numește *rația progresiei*).

Soluție

Vom citi mai întâi primii doi termeni și vom calcula diferența lor, pentru a stabili rația progresiei aritmetice. Vom citi apoi succesiv câte un număr natural, până la întâlnirea valorii -1, determinând diferența dintre numărul curent și cel precedent. Dacă diferența nu este egală cu rația progresiei, deducem că succesiunea de numere citită nu constituie o progresie aritmetică.

```

#include <iostream.h>
void main()
{ int a1, a2, r, e_progresie=1;
cout<<"Introduceti o succesiune de numere naturale:";
cin>>a1>>a2; r=a2-a1;
do
{ a1=a2; cin>>a2;
 if (a2 != -1 && a2-a1 != r) e_progresie=0; }
while (a2!= -1);
if (e_progresie) cout<<"Este progresie aritmetica";
else cout<<"Nu este progresie aritmetica"; }

```

8. Excursie

Drumul de la Gălăciuc la Soveja este marcat prin $n+1$ borne consecutive, borne numerotate de la 0, la n . Borna 0 (Gălăciuc) este la altitudinea 0. Pe fiecare dintre următoarele n borne sunt scrise câte două numere naturale, primul reprezentând altitudinea (înălțimea) locului și al doilea reprezentând distanța față de borna precedentă. Se consideră că dacă o bornă este la altitudinea x , iar borna următoare la altitudinea y (cu $x < y$), atunci drumul între cele două borne urcă (nu există zone plate sau de coborâre). Dacă o bornă este la altitudinea x , iar borna următoare este la altitudinea y (cu $x > y$), atunci drumul între cele

două borne coboară (nu există zone plate sau de urcuș). Dacă o bornă este la altitudinea x , iar borna următoare tot la altitudinea x , atunci drumul între cele două borne este tot timpul plat (nu există zone de urcuș sau de coborâș).

Se citește de la tastatură valoarea n și apoi n perechi de numere naturale reprezentând valorile înscrise pe cele n borne în ordinea: borna 1, borna 2, ..., borna n .

Deoarece turistul Hăstemele Softgimescu care pleacă de la Gălăciuc spre Soveja găfăie ori de câte ori urcă, vi se cere să afișați pe ecran lungimea celei mai lungi porțiuni continue pe care Softgimescu o parurge fără să găfăie. Dacă există numai porțiuni de urcare, se va afișa valoarea 0.

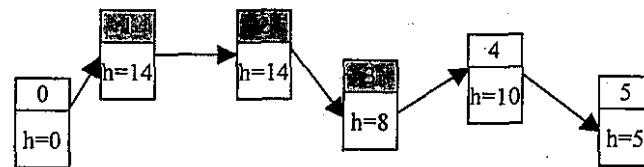
(ONIG, Gălăciuc, 2001, clasa a VI-a)

Restricții

- Toate numerele din problemă sunt numere naturale nenule de cel mult două cifre fiecare.

Exemplu

Pentru $n=5$, $h_1=14$, $d_1=3$, $h_2=14$, $d_2=19$, $h_3=8$, $d_3=4$, $h_4=10$, $d_4=10$, $h_5=5$, $d_5=17$ se afișează valoarea 23.



Soluție

Vom citi succesiv înălțimea și distanța până la borna precedentă pentru fiecare din bornele 1, 2, ..., n . Pe parcursul citirii vom determina lungimea porțiunii de plat și/sau coborâș curente (în variabila D), iar în variabila DMax vom reține lungimea celei mai lungi porțiuni continue de plat sau/și coborâș. La urcuș variabila D va reține valoarea 0. La fiecare pas verificăm dacă este sau nu urcuș (adică dacă înălțimea bornei curente este mai mare decât înălțimea bornei precedente). În caz afirmativ, porțiunea curentă de plat sau/și coborâș s-a terminat, comparăm lungimea ei cu DMax și actualizăm DMax dacă este necesar. Dacă nu, deducem că porțiunea de plat sau/și coborâș continuă, prin urmare adăugăm la D distanța dintre borna curentă și cea precedentă.

```

#include <iostream.h>
void main()
{ int n, i, h1=0, h2, d1=0, d2, DMax=0, D=0;
cout<<"n= "; cin>>n;
cout<<"Introduceti "<<n<<" perechi de numere naturale:";
for (i=1; i<=n; i++)
{ cin>>h2>>d2;
 if (h1<h2)      //urcuș
 { if (D>DMax) DMax=D;
 D=0; }
}

```

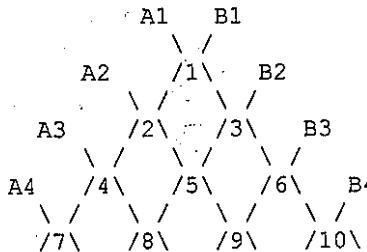
```

        else D+=d2;
        h1=h2; d1=d2; }
//e posibil ca drumul sa nu se termine cu urcuse
if (D>DMax) DMax=D;
cout<<DMax;
    }

```

9. Coordonate

Numerele naturale se aşază într-un triunghi, ca în figură:



Fiecare număr este așezat în vârful unui romb și este unic determinat de cele două diagonale (A și B). Diagonalele sunt numerotate de sus în jos. În acest fel:

- numărul 1 este așezat pe diagonala A-1 și diagonala B-1;
- numărul 2 este așezat pe diagonala A-2 și diagonala B-1;
- numărul 3 este așezat pe diagonala A-1 și diagonala B-2 și.a.m.d.

Fiind dat de la tastatură un număr întreg pozitiv n, să se scrie coordonatele celor două diagonale pe care se găsește n.

(Concurs Sinaia, 1998, clasa a VII-a)

Restricții

- $0 < N < 32768$

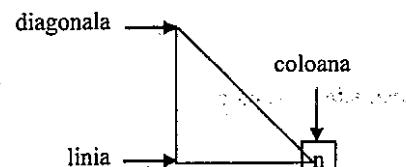
Exemple

Pentru $n=12$ se va afișa pe ecran: A-4 B-2. Pentru $n=6$ se va afișa pe ecran: A-1 B-3.

Soluție

Pentru a determina cu ușurință numărul diagonalelor corespunzătoare numărului n, „deformăm” triunghiul, transformându-l într-un triunghi dreptunghic isoscel:

1
2 3
4 5 6
7 8 9 10



Observăm că numărul diagonalei B pe care se află n coincide cu numărul coloanei pe care se află n în triunghiul deformat. Numărul diagonalei A îl putem deduce din numărul liniei pe care se află n. Mai exact, deoarece într-un triunghi dreptunghic isoscel catetele au aceeași lungime,

deducem că diferența dintre numărul liniei și numărul diagonalei A + 1 este egală cu numărul coloanei pe care se află n. Deci problema se reduce la a determina numărul liniei și numărul coloanei pe care se găsește n în triunghiul deformat. Pentru aceasta observăm că pe oricare linie lin se află exact lin numere. Prin urmare, pentru a afla numărul liniei pe care se află n, vom scădea succesiv din n, atât timp cât este posibil (adică $lin < n$), valoarea lin (unde lin este numărul liniei curente) și incrementând apoi numărul liniei. După scădere, în n va rămâne numărul coloanei.

```

#include <iostream.h>
void main()
{int n, lin=1;
cout<<"n = "; cin>>n;
while (lin < n)           //determin numărul liniei
    (n-=lin, lin++);
cout<<"A-<<lin - n + 1<< B-<<n; }

```

10. Becuri și etichete

Pentru iluminatul public, pe strada Info se găsesc un număr par de stâlpi, pe fiecare stâlp fiind plasat un singur bec, de culoare albă sau galbenă. Pentru a se putea ști și ziua, cândbecurile sunt stinse, dacă un bec este alb sau galben, elevii de la o școală au scris pe fiecare stâlp un număr. Numărul de pe stâlpii cu număr de ordine impar reprezintă numărul debecuri albe ce sunt în partea stângă a stâlpului (exclusiv stâlpul respectiv). Numărul de pe stâlpii cu număr de ordine par reprezintă numărul de becuri galbene ce sunt în partea stângă a stâlpului (exclusiv stâlpul respectiv). Un elev mai zburdalnic a modificat numerele de pe stâlpi, astfel încât numărul scris sub fiecare bec galben este eronat, iar numărul scris sub fiecare bec alb este corect. Elevii care învață informatică au spus că, în aceste condiții, ei pot să determine cu exactitate culoarea tuturor becurilor.

scrieți un program care să citească de la tastură numărul de stâlpi N, precum și numerele scrise pe cei N stâlpi și care să afișeze pe ecran culoarea celor N becuri.

(OJL, Timiș, 2000, clasa a IX-a)

Exemplu

Pentru $N=6$ și numerele 0 0 1 1 4 2 se va afișa pe ecran:

Becul 1: alb
Becul 2: alb
Becul 3: galben
Becul 4: alb
Becul 5: galben
Becul 6: alb

Soluție

Vom citi succesiv numerele scrise pe cei N stâlpi, contorizând în același timp numărul de becuri albe și numărul de becuri albe depistate până la momentul respectiv. Dacă numărul curent corespunde unui stâlp cu număr de ordine impar și numărul de becuri albe este corect, atunci becul curent este alb, altfel becul curent este galben. Dacă numărul

current corespunde unui stâlp cu număr de ordine par și numărul de becuri galbene este corect, atunci becul current este alb, altfel este galben. În orice caz, becul current va fi contorizat, în funcție de culoarea lui.

```
#include <iostream.h>
void main()
{int n, i, impar, par, nra=0, nrg=0;
cout<<"n= "; cin>>n;
for (i=1; i<=n; i+=2)
    {cin>>impar;
     if (impar==nra) {cout<<"Becul "<<i<<" alb\n"; nra++;}
     else {cout<<"Becul "<<i<<" galben\n"; nrg++;}
    cin>>par;
     if (par==nrg) {cout<<"Becul "<<i+1<<" alb\n"; nra++;}
     else {cout<<"Becul "<<i+1<<" galben\n"; nrg++;}
    }
}
```

11. Fracții

O proprietate interesantă a fracțiilor ireductibile este că orice fracție se poate obține după următoarele reguli:

- pe primul nivel se află fracția $1/1$;
- pe al doilea nivel, în stânga fracției $1/1$ de pe primul nivel, plasăm fracția $1/2$, iar în dreapta ei fracția $2/1$;
- nivelul 1: $1/1$
- nivelul 2: $1/2 \quad 2/1$
- pe fiecare nivel k se plasează sub fiecare fracție i/j de pe nivelul de deasupra, fracția $i/(i+j)$ în stânga, iar fracția $(i+j)/j$ în dreapta.
- nivelul 1: $1/1$
- nivelul 2: $1/2 \quad 2/1$
- nivelul 3: $1/3 \quad 3/2 \quad 2/3 \quad 3/1$

Dându-se o fracție oarecare prin numărătorul M și numitorul său N , determinați numărul nivelului pe care se află fracția sau o fracție echivalentă (având aceeași valoare) cu aceasta.

(ONI, Bacău, 2001, clasa a IX-a)

Restriții

- $1 < N, M \leq 2000000000$ (două miliarde)
- Timp maxim de execuție: 1 secundă/test.

Exemple

Pentru $M=13$ și $N=8$, numărul nivelului este 6. Pentru $M=12$ și $N=8$, numărul nivelului este 3.

Soluție

În primul rând, determinăm c.m.m.d.c. (M, N) și simplificăm fracția, obținând astfel o fracție ireductibilă. Pentru a determina nivelul, am putea să efectuăm scăderi repetitive (simulând operația inversă construcției fracțiilor pe niveluri), până când numitorul sau numărătorul devin 1. Pentru a mări viteza de execuție a programului (care ar fi extrem

de mică în anumite situații, de exemplu $M=2, N=1999999999$), vom utiliza împărțiri în locul scăderilor repetitive, având grijă să contorizăm corect nivelurile.

```
#include <iostream.h>
void main()
{long m, n, D, I, r, niv=0;
cout<<"n= "; cin>>n;
D=m; I=n; //determin cmmmdc(m,n)
while (I) {r=D%I; D=I; I=r;} //simplific fractia
I=n/D; D=m/D; //determin nivelul
while (I!=1 & D!=1) //determin nivelul
    if (D>I) {niv+=D/I; D%D=0;}
    else {niv+=I/D; I%D=0;}
cout<<niv+D+I-1; }
```

12. Sumă de numere consecutive

Fie n un număr natural, citit de la tastatură. Scrieți un program care să determine toate reprezentările numărului natural n ca sumă de cel puțin două numere naturale consecutive.

Exemplu

Pentru $n=15$, programul va afișa:

Solutia nr. 1 $7+8$
Solutia nr. 2 $4+5+6$
Solutia nr. 3 $1+2+3+4+5$

Soluție

Să considerăm că n este scris ca sumă de x termeni consecutivi, începând cu k :

$$n=k+(k+1)+\dots+(k+x-1)=k \cdot x + (1+2+\dots+x-1)=k \cdot x + x \cdot (x-1)/2.$$

Înmulțind ambii membri ai acestei egalități cu 2, obținem:

$$2 \cdot n = 2k \cdot x + x \cdot (x-1) \Rightarrow 2 \cdot n = x \cdot (2k+x-1)$$

Deducem că x trebuie să fie un divizor al lui $2 \cdot n$, prin urmare este suficient să îl căutăm pe x printre divizorii lui $2 \cdot n$ și să îl determinăm pe k în funcție de n și x ($k=(n/x-(x-1))/2$).

```
#include <iostream.h>
void main()
{long n, n2, x, k, nr=0, i;
cout<<"n= "; cin>>n;
for (n2=2*n, x=2; x<=n2; x++)
    if (n2%x ==0) //x divide 2n, determin k
        if ((n2/x-x+1)%2==0) //k este numar intreg
            k=(n2/x-x+1)/2;
            if (k>0)
                {cout<<"Solutia nr. "<<++nr<<' ';
```

```

    for (i=0; i<x-1 ; i++) cout<<k+i<<'+';
    cout<<k+x-1<<endl;
}

```

13. Tablou

Scrieti un program care sa genereze un tablou bidimensional cu proprietatile:

- contine N linii si N coloane;
- elementele sale sunt numere naturale nenule;
- suma elementelor este egală cu numărul natural nenul S;
- pe nici o linie și pe nici o coloană nu există două elemente identice;
- diferența dintre cel mai mare și cel mai mic element ale tabloului este minimă.

Programul va citi de la tastatură numerele naturale nenul N și S și va afișa pe ecran tabloul generat.

(ONI, Bacău, 2001, clasa a IX-a)

Restricții

- $1 < N \leq 100$
- $0 < S \leq 2^{31}$
- Dacă problema nu are soluție, în fișierul de ieșire se va scrie cifra 0.
- Dacă problema are mai multe soluții, se va scrie una singură.

Exemplu

Pentru $N=3$ și $S=51$ programul poate afișa:

```

4 6 7
7 4 5
5 7 6

```

Soluție

Deoarece se dorește ca „diferența absolută” (diferența dintre cea mai mică și cea mai mare valoare din tablou) să fie minimă, vom plasa un număr maxim de elemente cu aceeași valoare în tablou, astfel încât să nu se repete pe aceeași linie sau coloană. Pentru aceasta, vom realiza o distribuție pe diagonale, plasând pe orice diagonală elemente cu aceeași valoare (considerăm că toate diagonalele sunt formate din n elemente, construindu-le prin completarea tabloului cu o copie a primelor $n-1$ linii). Din același motiv, între valorile de pe diagonale diferite diferența trebuie să fie cât mai mică. De aceea vom alege valori consecutive. Suma tuturor elementelor în acest caz este:

$$\begin{aligned}
S' &= n \cdot a + n \cdot (a+1) + n \cdot (a+2) + \dots + n \cdot (a+n-1) \\
&= n \cdot [a + (a+1) + (a+2) + \dots + (a+n-1)] = n^2 [a + (n-1)/2]
\end{aligned}$$

Deoarece $S' \leq S$, cea mai mare valoare posibilă pentru variabila a este:

$$a = [S/n^2 - (n-1)/2]$$

Pentru exemplul nostru, se obține $a=4$ și o distribuție de tipul:

4	5	6
6	4	5
5	6	4

Diferența $S-S'$ se obține adăugând căte o unitate elementelor tabloului, pornind de la cea mai mare valoare din progresie $(a+n-1)$, pe toată diagonala sa, apoi se trece la valoarea următoare $(a+n-2)$ pe toată diagonala sa, procesul încheindu-se când s-au epuizat cele $S-S'$ unități de distribuit. Datorită faptului că valorile plasate pe o diagonală se măresc toate cu o unitate înainte de a trece la o altă diagonală (cu valori mai mici), nu se vor realiza situații în care pe două diagonale diferite să se afle aceeași valoare. Evident, dacă diferența $S-S'$ nu se divide cu n , va exista o diagonală care va avea doar o parte din valori mărite, celelalte rămânând cu vechea valoare.

Deoarece valoarea maximă a tabloului se mărește o singură dată cu o singură unitate, vom obține un tablou cu diferență absolută n .

```

#include <iostream.h>
void main()
{
    long n, S, a, Sp, i, j, cd, rd, d;
    cout<<"n, S= "; cin>>n>>S;
    a=(S/n-n*(n-1)/2)/n;
    if (a<=0) cout<<0;
    else
        { Sp=n*(n*a+n*(n-1)/2);
          cd=(S-Sp)/n; //nr. de diagonale actualizate integral
          rd=(S-Sp)%n;
          //nr.de elemente de actualizat de pe diagonala cd+1
          for (i=1; i<=n; i++)
              { for (j=i; j<=n; j++)
                  { d=(n+j-i)%n+1; //diagonala elementului i,j
                      if ((d>n-cd)||((d=n-cd)&&(j<=rd)))
                          cout<<a+d<<' ';
                      else cout<<a+d-1<<' ';
                  }
              cout<<endl;
        }
}

```

Probleme propuse

1. Triunghi

Se citesc de la tastatură 3 numere reale pozitive, care reprezintă lungimile laturilor unui triunghi. Scrieți un program care să determine:

- aria triunghiului;
- lungimile înălțimilor triunghiului;
- natura triunghiului (echilateral, isoscel, dreptunghic, dreptunghic isoscel, oarecare etc.).

2. Balaur

A fost odată un balaur cu 6 capete. Într-o zi Făt-Frumos s-a supărat și i-a tăiat un cap. Peste noapte i-au crescut alte 6 capete în loc. Pe același gât! A doua zi, Făt-Frumos iar i-a tăiat un cap, dar peste noapte balaurului i-au crescut în loc alte 6 capete... și tot aşa timp de n zile. În cea de $(n+1)$ -a zi, Făt-Frumos s-a plăcuit și a plecat acasă!

Scrieți un program care citește de la tastatură n , numărul de zile, și care afișează pe ecran câte capete avea balaurul după n zile.

(ONIG, 2002, clasa a V-a)

3. Gigel

Gigel este un tip ciudat. Lui îi place să își impresioneze colegii exprimând duratele numai în secunde. De exemplu, dacă îl vei întreba cât e ceasul el își va răspunde câte secunde s-au scurs de la ora 0.00 din ziua respectivă. Dacă ai să-l întrebi ce vârstă are, el își va răspunde câte secunde au trecut de când s-a născut.

Colegii lui Gigel au hotărât că nu e cazul să se lase impresionați; ca urmare au nevoie de un program care să citească de la tastatură un număr natural N ($N \leq 2000000000$) care reprezintă vârsta lui Gigel exprimată în secunde și care va afișa pe ecran câți ani, câte luni și câte zile are Gigel (orele și minutele rămase sunt considerate nesemnificative). Scrieți acest program pentru colegii lui Gigel!

Nu uită că anii bisecți sunt cei divizibili cu 4, dar nedivizibili cu 100 sau divizibili cu 400. De exemplu, 1992 și 2000 au fost ani bisecți. Dar anul 1900 nu a fost bisect. Anii bisecți au 366 de zile, spre deosebire de ceilalți care au doar 365. Considerăm că ne aflăm în ultima zi de școală (15 iunie 2002).

Exemplu

Pentru $N=69206400$ programul va afișa: Gigel are 2 ani, 2 luni și 10 zile.

(ONIG, Gălăciuc; 2002, clasa a V-a)

4. Ciupercute fermecate

Un vrăjitor bătrân vrea să prepare o licoare specială. Pentru o doză de licoare el are nevoie de M ciuperci fermecate. O ciupercă este fermecată dacă numărul bulinelor de pe pălăria ei este prim. Ucenicul vrăjitorului a cules N ciuperci dintre care unele sunt fermecate, altele nu. Vrăjitorul vrea să afle câte doze de licoare poate prepara din ciupercile culese, câte ciuperci fermecate îi rămân și câte ciuperci nu sunt bune de nimic. Scrieți un program care-l ajută!

Date de intrare

Se citesc de la tastatură:

M – numărul de ciuperci necesare pentru o doză

N – numărul de ciuperci culese

Numărul de buline de pe pălăria fiecărei dintre cele N ciuperci

Date de ieșire

Se vor afișa pe ecran, pe linii diferite:

– numărul dozelor care se pot obține;

- numărul ciupercilor fermecate care rămân;
- numărul ciupercilor care nu sunt fermecate.

Exemplu

Pentru $M=3$, $N=8$ și numerele bulinelor de pe fiecare ciupercă:

2 11 5 15 7 3 13 23

se vor afișa:

doze: 2

ciuperci fermecate ramase: 1

ciuperci care nu sunt fermecate: 1

Observații

Fiecare ciupercă are cel puțin două buline.

(Lugoj, 2002, clasele a V-a – a VI-a)

5. Divizibilitate

Se citește de la tastatură un număr natural nenul n și apoi o succesiune de n numere întregi a_1, a_2, \dots, a_n . Scrieți un program care să verifice dacă numerele citite se divid succesiv (adică $a_1 | a_2, a_2 | a_3, \dots, a_{n-1} | a_n$).

6. Paranteze**Definiții**

Un sir format numai din paranteze ‘[’ și ‘]’ este *corect* numai dacă :

- numărul de paranteze ‘[’ este egal cu numărul de paranteze ‘]’;
- orice paranteză ‘]’ din sir are înaintea ei (pe pozițiile precedente) mai multe paranteze ‘[’ decât paranteze ‘]’.

De exemplu, sirul [[]] [] este corect.

Sirul [[]] [] [] nu este corect.

Nici sirul [] [] [] [] nu este corect.

Orice paranteză ‘[’ urmată imediat în sir de o paranteză ‘]’ formează un *mugur*.

De exemplu, sirul [[]] [[[]]] este corect și conține 2 muguri.

Cerință

Se citesc de la tastatură două numere naturale p și m (cu cel mult 2 cifre fiecare).

Să se afișeze pe ecran 3 siruri diferite de paranteze, fiecare fiind un sir corect format din exact p paranteze și conținând exact m muguri.

Dacă nu există 3 astfel de siruri se va afișa mesajul “Nu se poate”.

Exemplu

Pentru $p=8$, $m=2$, trei siruri corecte cu 8 paranteze și 2 muguri sunt:

[[]] [[]]

[] [[[]]]

[[[]]] []

(Lugoj, 2002, clasele a V-a – a VI-a)

7. Câte sunt?

Fie mulțimea $A=\{1, 2, \dots, n\}$. Scrieți un program care să citească de la tastatură numărul natural n și care afișează pe ecran câte numere rationale distincte de forma p/q cu p și q din A există?

(Concurs Lugoj, 2000, clasele a VII-a – a VIII-a)

Restricții

- $1 < n < 10000$

Exemplu

Pentru $n=3$ se afișează 7.

8. Numere naturale

Fie M și N două numere naturale nenule date. Scrieți un program care să determine mulțimea numerelor naturale K cu proprietatea că numărul $((M+N)*K) / (K*K+M*N)$ este natural.

9. Factorial

Fie N un număr natural ($1000 \leq N \leq 2000000.000$) despre care se știe că reprezintă factorialul unui număr k ($N=1 \cdot 2 \cdot 3 \cdots k$). Scrieți un program care să citească pe N și să determine numărul k .

(OJI, Satu Mare, clasa a IX-a)

10. Număr rotund

Fie N un număr natural cu cel mult 9 cifre. O permutare circulară la stânga a lui N se realizează prin deplasarea fiecărei cifre a numărului cu o poziție spre stânga, prima cifră a numărului devenind ultima. De exemplu, printr-o permutare circulară la stânga a numărului 1234, vom obține 2341. Dacă notăm cu Lg numărul de cifre din N , observăm că după Lg permutări circulare, revenim la numărul inițial. Vom spune că numărul N este rotund dacă el se poate obține prin x permutări circulare, cu $x < Lg$. De exemplu, $n=3232$ este un număr rotund.

Scrieți un program care să citească de la tastatură un număr natural și să verifice dacă el este sau nu rotund.

11. Paginare

Profesorul de informatică afirmă că pentru numerotarea paginilor manualului de informatică au fost utilizate n cifre. Scrieți un program care să verifice dacă este posibil acest lucru și dacă da, să determine câte pagini are manualul de informatică. Numărul natural n va fi citit de la tastatură ($n < 10^9$).

12. Sumă minimă

Se citește de la tastatură un număr natural n ($2 \leq n \leq 65000$). Numărul n se reprezintă în bazele de numerație de la 2 la 10. Să se determine bazele în care suma cifrelor reprezentării numărului n este minimă.

(Concurs Suceava, 1999, clasa a VII-a)

Exemplu

Pentru $n=13$ bazele în care suma cifrelor reprezentării lui n este minimă sunt:

- 2, 3, 6

13. Propoziție

Se citește caracter cu caracter o propoziție terminată cu '...'. Scrieți un program care să determine lungimea celui mai lung cuvânt din propoziție, știind că oricare două cuvinte consecutive sunt separate prin unul sau mai multe spații.

14. Pitagora

Se consideră un număr natural din intervalul [1,32457]. Să se găsească toate perechile de numere naturale b și c având proprietatea $a < b < c$ pentru care există un triunghi dreptunghic având laturile de lungimi a , b , c .

15. Sferă

Câte puncte cu coordonate întregi sunt conținute într-o sferă de rază R cu centrul în originea sistemului de coordonate? Se consideră că R este un număr natural, $R \leq 30$. Amintim că distanța d dintre un punct cu coordonatele (x, y, z) și originea sistemului de coordonate se determină după formula

$$d = \sqrt{x^2 + y^2 + z^2}$$

Exemplu

Pentru $R=4$, programul va afișa 257.

(Olimpiada Republicană de Informatică, Republica Moldova, 2001, clasele a VI-a – a IX-a)

16. Numere

Elaborați un program care descompune numărul natural n în sumă de numere naturale $n = n_1 + n_2 + \dots + n_k$, astfel încât produsul lor $p = n_1 \cdot n_2 \cdot \dots \cdot n_k$ să fie maxim. Programul va citi numărul n ($n \leq 50$) și va afișa produsul p .

Exemplu

Pentru $n=11$, programul va afișa 54.

(Olimpiada Republicană de Informatică, Republica Moldova, 2001, clasele a VI-a – a IX-a)

Structuri elementare de date

1. Descompunere

Fie n un număr natural ($n \leq 1000$). Scrieți un program care să determine descompunerea în factori primi a lui $n!$.

(OJI, Iași, 2001, clasa a IX-a)

Exemplu

Pentru $n=6$, programul va afișa:

```
2 4
3 2
5 1
```

Soluție

Deoarece $n \leq 1000$, e dificil să calculăm pe $n!$. Cum factorii primi ai lui $n!$ sunt din intervalul $[2, 1000]$, în loc să calculăm pe $n!$ și apoi să îl descompunem în factori primi, vom determina descompunerea în factori primi pe parcurs. Mai exact, vom utiliza un vector m în care să reținem pentru fiecare număr prim din intervalul $[2, 1000]$ multiplicitatea sa în descompunerea în factori primi a lui $n!$. Înmulțirea cu x presupune numai actualizarea multiplicităților factorilor primi din descompunerea în factori primi a lui x .

```
#include <iostream.h>
int m[1001], d, mx, n, x;
void main ()
{
    cin>>n;
    for (int xx=2; xx<=n; xx++)
        //descompun pe x în factori primi
        x=xx;
        for (d=2; x>1; d++)
            if (x%d == 0) //d divide x
                { for (mx=0; x%d==0; mx++, x/=d);
                  m[d]+=mx;
                }
    for (d=2; d<=n; d++)
        if (m[d]) cout<<d<<' '<<m[d]<<endl;
}
```

2. Suma cuburilor

Gelu îndrăgește matematica. El are totdeauna la el un număr natural.

Într-o bună zi, Gelu se întâlnește cu zâna matematicii. Zâna are o baghetă fermecată: când atinge o dată un număr, acesta se transformă în alt număr natural, obținut prin însumarea cuburilor cifrelor numărului inițial. Astfel, dacă Gelu are la el numărul 314,

acesta devine 92 după o atingere de baghetă ($3^3+1^3+4^3=92$). La o nouă atingere, numărul obținut iar se transformă, noul număr fiind suma cuburilor cifrelor numărului anterior. Deci, la o nouă atingere, 92 devine 737 ($9^3+2^3=737$). Gelu o roagă pe zână să atingă cu bagheta magică numărul pe care îl are la el. El are o memorie bună și ține minte toate numerele obținute prin atingerile succesive ale baghetei. După un număr de atingeri, timp în care numărul se tot transformă, Gelu o oprește pe zână spunându-i: „Numărul acesta a mai apărut. De acum nu mai are rost să te obosești”.

Scrieți un program care să citească de la tastatură numărul natural n pe care îl are Gelu la el și care să afișeze pe un rând numerele obținute după fiecare atingere și, pe rând nou, numărul de atingeri după care băiețelul o oprește pe zână.

(Concurs Lugoj, 2000, clasele a V-a – a VI-a)

Restricții

- $1 \leq n \leq 999$

Exemplu

Pentru $n=314$, programul va afișa pe ecran:

```
92 737 713 371 371
5
```

Soluție

Vom calcula succesiv suma cuburilor cifrelor numărului n curent, până când obținem un număr pe care l-am mai obținut. Pentru a reține mulțimea numerelor pe care le-am obținut, vom utiliza o reprezentare prin vector caracteristic. Mai exact, vom utiliza un vector uz , cu următoarea semnificație $uz[i]=1$ dacă numărul i a fost deja obținut și 0, altfel.

```
#include <iostream.h>
int uz[2500];
void main()
{
    unsigned n, nr=0, c, sum;
    cout<<"n = "; cin>>n;
    do
    { sum=0; uz[n]=1;
      do //determin suma cuburilor cifrelor lui n
          (c=n%10; sum+=c*c*c; n/=10; )
      while (n);
      n=sum; cout<<n<<' ';
      nr++; }
    while (!uz[n]);
    cout<<endl<<nr;
}
```

3. Ordonare

În urma unor lucrări experimentale de laborator, micii fizicieni au obținut prin măsurători o succesiune de valori, numere naturale, care nu pot depăși valoarea 25000.

Pentru a prelucra aceste valori în scopul interpretării lor din punct de vedere fizic, valorile trebuie să fie ordonate. Pentru a ușura procedeul de ordonare, micii fizicieni au observat că nici una dintre valorile obținute nu se repetă de mai mult de 60000 de ori. Scrieți un program care să ordoneze descrescător valorile obținute.

Date de intrare

Fișierul de intrare VALORI.IN conține, câte una pe linie, valorile:

V₁
V₂
...

Date de ieșire

Fișierul de ieșire VALORI.OUT va conține, câte una pe linie, valorile ordonate descrescător:

V_{O1}
V_{O2}
...

Restricții

- 1 ≤ V_i ≤ 25000

Timp maxim de execuție: 1 secundă/test.

Soluție

Problema impune sortarea unor valori. Datorită condițiilor particulare ale problemei (faptul că domeniul valorilor posibile este restrâns, între 1 și 25000), putem realiza sortarea în timp liniar: vom citi valorile din fișierul de intrare, calculând pe parcursul citirii, frecvența de apariție a fiecarei valori din domeniul specificat. Pentru a afișa valorile în ordine descrescătoare, vom parcurge domeniul de valori și vom afișa fiecare valoare, de câte ori apare ea în fișierul de intrare.

```
#include <fstream.h>
#define VMax 25000
unsigned Cat[i[VMax];
void main()
{ unsigned i, j, Numar;
  ifstream f("valori.in");
  ofstream g("valori.out");
  while (!f.eof()) { f>>Numar; Cat[Numar]++; }
  f.close();
  for (i=VMax; i>0; i--)
    for (j=1; j<=Cat[i]; j++) g<<i<<endl;
  g.close();}
```

4. Cifra 4

Fie n un număr natural nenul. Scrieți un program care să genereze numărul n pornind de la cifra 4, fiind permise numai următoarele operații:

- împărțirea la 2;
- adăugarea la dreapta a cifrei 0;
- adăugarea la dreapta a cifrei 4.

(Tabăra de Informatică, Focșani, 1993).

Soluție

Vom încerca să identificăm operațiile care generează numărul n, plecând înapoi, de la n către 4. Mai exact: dacă ultima cifră a lui n este 0 sau 4, o eliminăm (ceea ce corespunde operației de adăugare la sfârșit a unui 0 sau 4), altfel înmulțim numărul cu 2. Procedeul se repetă până când obținem rezultatul 4. Rezultatele parțiale obținute pe parcurs, le vom reține într-un vector. Deoarece noi am utilizat operațiile inverse, vom afișa componentele vectorului de rezultate în ordine inversă. Rămâne ca exercițiul de demonstrat că acest algoritm se termină (pentru aceasta este suficient să demonstrezi că, pentru orice n, prin aceste operații inverse se poate ajunge la 1, utilizând scrierea în baza 2 a lui n).

```
#include <iostream.h>
void main()
{ int n, i=1, rez[100];
  cout<<"n= "; cin>>n;
  rez[0]=n;
  while (n!=4)
    { if (n%10 == 0 || n%10 == 4) n/=10;
      else n*=2;
      rez[i++]=n;
    }
  for (n=i-1; n>=0; n--) cout<<rez[n]<<";" }
```

5. Formă 3-prime

Fie P, Q și R trei numere naturale prime astfel încât P<Q<R. Scrieți un program care să genereze în ordine crescătoare toate numerele naturale mai mici decât o valoare dată VMax, care au forma PⁱQ^jR^k (i, j, k numere naturale).

Restricții

- VMax<30000

Exemplu

Pentru P=2, Q=3, R=5 și VMax=20, programul va afișa:

1 2 3 4 5 6 8 9 10 12 15 16 18 20

Soluție

Să notăm cu H mulțimea numerelor naturale de forma $P^i Q^j R^k$. Observăm în primul rând că $1 \in H$ și că $P \cdot z, Q \cdot z$ și $R \cdot z \in H$, pentru orice $z \in H$. Pentru a genera în ordine crescătoare elementele mulțimii H , vom utiliza 3 variabile (xp , xq și xr) în care memorăm elementele care urmează să fie introduse în H și care se obțin prin înmulțirea cu P , Q , respectiv R , a unor elemente deja introduse în H (referite cu ajutorul indicilor ip , iq , respectiv ir). La fiecare pas va fi introdus $\min\{xp, xq, xr\}$, fiind actualizată numai variabila corespunzătoare elementului introdus.

```
#include <iostream.h>
void main()
{ int VMax, p, q, r, xp, xq, xr, x, ip, iq, ir, i, H[300];
cout<<"VMax, p, q, r = "; cin>>VMax>>p>>q>>r;
x=H[0]=1; xp=p; xq=q; xr=r; ip=iq=ir=i=0;
while (x <= VMax)
{ x=(xp<xq?(xp<xr?xp:xr):(xq<xr?xq:xr));
  if (x<=VMax) H[++i]=x;
  if (xp==x) xp=p*H[++ip];
  if (xq==x) xq=q*H[++iq];
  if (xr==x) xr=r*H[++ir];
}
for (ip=0; ip<=i; ip++) cout<<H[ip]<<' ';
```

6. Secvență palindromică

Să considerăm un sir de caractere, care pot fi doar litere mici ale alfabetului englez. Numim *secvență palindromică* o succesiune de litere din sir care are proprietatea palindromică (fie că o parcugem de la stânga la dreapta, fie că o parcugem de la dreapta la stânga, secvența este aceeași). De exemplu, succesiunea de litere *cojoc* are proprietatea palindromică.

Scriți un program care să determine cea mai lungă secvență palindromică dintr-un sir dat.
(Concurs Lugoj, 2001, clasele a VII-a – a VIII-a)

Date de intrare

Fișierul de intrare SP.IN conține două linii:

N – numărul de litere din sirul de intrare
 $s_1 s_2 \dots s_N$ – sirul de N litere

Date de ieșire

Fișierul de ieșire SP.OUT conține:

poz – poziția de început a celei mai lungi secvențe palindromice
 lg – lungimea celei mai lungi secvențe palindromice

Restricții

- $0 < N \leq 20000$

Timp maxim de execuție: 1 secundă/test.

Exemplu

SP.IN	SP.OUT
22	9
anaareuncojocasafrumos	5

Observații

- Dacă există mai multe soluții, programul va obține una singură.
- Pozиїile elementelor din sirul de intrare sunt numerotate începând cu 1.

Soluție

O primă idee ar fi de a lăua în considerare toate secvențele posibile, în ordinea descrescătoare a lungimii lor și de a verifica dacă sunt sau nu secvențe palindromice. În cazul cel mai defavorabil, acesta este un algoritm de ordinul lui n^3 . Pentru a obține un algoritm practic, vom analiza sirul de caractere dat, fixând mijlocul secvenței palindromice curente în toate modurile posibile (în cazul cel mai defavorabil n variante posibile). Mijlocul fiind fixat, pentru a determina secvența palindromică având mijlocul respectiv, ne vom deplasa de la mijloc spre stânga și spre dreapta, comparând elementele simetrice (maximum $n/2$ comparații). Evident, vom avea grija să considerăm ambele situații: mijlocul secvenței palindromice este formaț dintr-un singur element (lungimea ei este impară) sau din două elemente (lungimea ei este pară). Pe parcursul acestei analize, vom compara permanent lungimea secvenței palindromice curente cu lungimea secvenței palindromice maximale, actualizând secvența maximală, dacă este cazul.

```
#include <fstream.h>
#include <string.h>
#define NMax 20050

char s[NMax];
int n, stfix, max=1;

void main()
{int st, dr, m, mijloc;
ifstream f("sp.in");
f>>n; f.get(); f.getline(s, NMax);
f.close();
for (mijloc=0; 2*(long)(n-mijloc)>max; mijloc++)
{for(st=mijloc, dr=mijloc+1;
st>=0&&dr<n&&s[st]==s[dr]; st--, dr++);
m=dr-st-1;
if (max<m) {max=m; stfix=st+1;}
for (st=dr=mijloc; st>=0 && dr<n && s[st]==s[dr]; )
st--, dr++;
m=dr-st-1;
if (max<m) {max=m; stfix=st+1;}
}
ofstream f("sp.out");
f<<stfix+1<<endl<<max<<endl;
f.close(); } //afisare
```

7. Elevi

Fișierul de intrare ELEVI.IN conține pe fiecare linie următoarele informații despre câte un elev de la liceul de informatică, separate prin spații:

nume_elev clasa medie_generală absențe_nemotivate

Știind că sunt maximum 500 de elevi, scrieți un program care să determine:

- elevii cu număr maxim de absențe nemotivate;
- mediile generale pe clase.

Soluție

Vom memora informațiile despre elevi într-un vector cu înregistrări având următoarea structură:

```
struct elev
{
    char nume[20];
    char clasa[3];
    float mg;
    int abs;
} e[500];
```

Vom citi informațiile din fișier linie cu linie, extrăgând de pe fiecare linie câmpurile înregistrării, utilizând funcția strtok():

```
ifstream f("elevi.in");
char s[100], *p;
int n=0;
do
{
    f.getline(s,100);
    if (f.good())
    {
        p=s;
        strcpy(e[n].nume, p=strtok(p, " "));
        strcpy(e[n].clasa, p=strtok(NULL, " "));
        e[n].mg=atof(p=strtok(NULL, " "));
        e[n].abs=atoi(p=strtok(NULL," "));
        n++;
    }
} while (!f.eof());
```

Printr-o parcurgere a vectorului vom determina numărul maxim de absențe:

```
for (int maxabs=-1, i=0; i<n; i++)
    if (e[i].abs>maxabs) maxabs=e[i].abs;
```

Vom efectua o a doua parcurgere a vectorului, pentru a identifica elevii cu număr maxim de absențe:

```
for (i=0; i<n; i++)
    if (e[i].abs==maxabs) cout<<e[i].nume<<endl;
```

Pentru a calcula mediile generale pe clase, vom sorta mai întâi elevii, după clasă:

```
int schimb;
elev aux;
do
{
    for (schimb=i=0; i<n-1; i++)
        if (strcmp(e[i].clasa, e[i+1].clasa)>0)
            {schimb=1;
             aux=e[i]; e[i]=e[i+1]; e[i+1]=aux;}
    }
while (schimb);
```

Vom parcurge acum fiecare clasă, determinând media pe clasă:

```
int nrelevi=1;           //nr. de elevi dintr-o clasă
float smg=e[0].mg;      //suma mediilor generale dintr-o clasă
for (i=1; i<n; i++)
{
    if (strcmp(e[i].clasa, e[i-1].clasa)==0)
        {smg+=e[i].mg; nrelevi++;}           //aceeași clasă
    else
        {cout<<"Media cls. "<<e[i-1].clasa<<":
             "<<smg/nrelevi<<endl;
         nrelevi=1; smg=e[i].mg; }
cout<<"Media clasei "<<e[n-1].clasa<<": "<<smg/nrelevi;
```

8. Înmulțirea a două matrice

Fie A o matrice cu n linii și m coloane și B o matrice cu m linii și p coloane ($1 \leq n, m, p \leq 100$). Să se calculeze produsul celor două matrice.

Soluție

Așa cum știm de la matematică, produsul $A \times B$ va fi o matrice C cu n linii și p coloane, definită astfel:

$$C[i][j] = \sum_{k=1, m} A[i][k] \cdot B[k][j], \forall i \in \{1, 2, \dots, n\}, j \in \{1, 2, \dots, p\}$$

Următoarea secvență de program calculează elementele matricei C:

```
for (i=0; i<n; i++)
    for (j=0; j<p; j++)
        for (k=C[i][j]=0; k<m; k++)
            C[i][j]+=A[i][k]*B[k][j];
```

Observați că pentru a calcula produsul $A \times B$ au fost necesare $n \cdot m \cdot p$ înmulțiri elementare.

9. Fracții

Se consideră primele N litere mari ale alfabetului englez ($1 \leq N \leq 26$) care reprezintă nume de variabile. Aceste litere se scriu succesiv în ordine alfabetică, delimitate prin operatorul de împărțire ':', formând astfel o expresie algebraică. Prin eventuala adăugare la această expresie a unor perechi de paranteze rotunde, se obține o fracție etajată. Fracția poate fi scrisă, în urma eliminării etajelor, sub forma unei fracții simple (care are la numărător, respectiv numitor, câte un produs de variabile). Cunoscând setul de caractere care formează produsul la numărător, se cere să se indice o modalitate de a așeza parantezele.

(ONI, Oradea, 1998, clasa a IX-a)

Date de intrare

Fișierul de intrare FRACTIE.IN conține două linii. Pe prima linie se află N (numărul de litere folosite), iar pe cea de a doua linie se află un sir de litere mari distințe, ordonate alfabetic, fără spații, reprezentând variabilele de la numărător.

Date de ieșire

Dacă există soluție, fișierul de ieșire FRACTIE.OUT conține pe prima linie expresia cu paranteze, fără nici un spațiu. Dacă nu există soluție, în fișierul de ieșire se va scrie mesajul NU.

Observații

1. Expresia trebuie să fie corect parantezată.
2. Dacă există mai multe soluții, se va afișa una singură.

Exemplu

FRACTIE.IN
5
ACE

FRACTIE.OUT
A: (B:(C:D):E)

Soluție

Observăm că, plasând perechi de paranteze rotunde, o variabilă poate ajunge fie la numărător, fie la numitor, cu excepția primei variabile (A) care nu poate fi decât la numărător și la celei de a doua variabile (B) care nu poate fi decât la numitor. Vom utiliza doi vectori (sus și current), de câte 26 de componente, câte una pentru fiecare literă a alfabetului englez. Vectorul sus reține dacă o literă trebuie să fie plasată „sus”, adică la numărător (sus[i]=1 dacă cea de a i literă din alfabet este la numărător și 0, altfel). Vectorul current reține amplasarea curentă a literelor, în același mod.

Parcurgem în ordine literele alfabetului, până la ultima literă citită ca fiind la numărător și testăm dacă în amplasarea curentă (reprezentată de vectorul current) litera este plasată corect. Dacă da, afișăm literă precedentă aşa cum este, dacă nu, afișăm o paranteză deschisă și apoi afișăm literă precedentă (în acest caz toate literele care urmează își schimbă poziția curentă). La sfârșit, închidem toate parantezele rotunde deschise și afișăm apoi eventualele litere neafișate (cele care urmează de la ultima literă citită la numărător, până la cea de a n-a literă din alfabetul englez).

```
#include <fstream.h>
int sus[26], current[26];
int n, i, nrdeschise, nrsus;
void main()
{ char c, cc, ultima;
  ifstream fin("fractie.in");
  fin>>n;
  while (!fin.eof())
    { fin>>c; nrsus++; sus[c-'A']=1;}
    ultima=c; //ultima literă care apare la numărator
  fin.close();
  ofstream fout("fractie.out");
  if (ultima=='A')
    if (!sus[0]) fout<<'NU';
    else fout<<'A';
  else
    if (!sus[0] || sus[1]) fout<<'NU';
    else
      { current[0]=1; fout<<"A:";
        for (c='C'; c<=ultima; c++)
          { if (sus[c-'A']!=current[c-'A'])
              { fout<< '(';
                for (cc=c; cc<=ultima; cc++)
                  current[cc-'A']=!current[cc-'A'];
                nrdeschise++;
              }
              fout<<(char)(c-1)<< ':';}
            fout<<ultima;
            for (i=1; i<=nrdeschise; i++) fout<< ')';
          }
        if (ultima<'A'+n-1)
          for (c=ultima+1; c<='A'+n-1; c++) fout<< ':'<<c;
        fout.close();}
```

10. Reconstituire

Într-un fișier text, denumit SURSA.OUT, se află cel mult $n \leq 10000$ caractere tipăribile (cu codurile ASCII cuprinse între 32 și 255) scrise pe aceeași linie. Din acest fișier se obține un alt fișier text, denumit DESTIN.IN, înlocuind fiecare caracter cu răsturnatul codului ASCII corespunzător (de exemplu caracterul 'A' care are codul ASCII 65 va fi înlocuit cu secvența '56', iar caracterul 'f' care are codul ASCII 102 cu secvența '201'). Primind fișierul DESTIN.IN se cere să se reconstituie fișierul SURSA.OUT.

(ONI, Suceava, 1996, clasa a XI-a)

Exemplu

Pentru intrarea:
566676201301
avem ieșirea:
ABCfg

Soluție

Observăm că în fișierul destinație există două tipuri de coduri: coduri formate din două caractere (cuprinse între 32 și 99) și coduri din 3 caractere (cuprinse între 100 și 255). Observăm că un cod de 3 caractere începe cu cifra 1 sau 2, în timp ce un cod de două caractere începe cu o cifră strict mai mare decât 2. Codurile fiind inversate, vom analiza fișierul destinație de la sfârșit spre început și pentru a decodifica un caracter vom extrage mai întâi prima cifră a codului (în ordinea dreapta – stânga). În funcție de prima cifră (mai mare decât 2 sau nu) vom extrage restul codului, deci încă unul sau două caractere.

```
#include <fstream.h>
#include <string.h>
char d[10000], s[5000];
int n, nr, i;
void main()
{
    ifstream f("destin.in");
    f.getline(d, 10000); n=strlen(d);
    f.close();
    for (i=n-1; i>=0;)           //decodific sirul
        if (d[i]>'2')
            {s[nr++]=(d[i]-'0')*10+d[i-1]-'0'; i-=2;}
        else
            {s[nr++]=(d[i]-'0')*100+(d[i-1]-'0')*10+d[i-2]-'0';
             i-=3;}
    ofstream g("sursa.out");
    g<<strrev(s);               //afisez sirul inversat
    g.close(); }
```

11. Permutarea următoare

Fie n ($n \leq 1000$) un număr natural nenul și $p = (p_1, p_2, \dots, p_n)$ o permutare de ordin n (succesiune de valori distincte din mulțimea $\{1, 2, \dots, n\}$). Determinați permutarea imediat următoare din punct de vedere lexicografic.

Observație

Spunem că sirul $x_1x_2\dots x_p$ precedă lexicografic sirul $y_1y_2\dots y_k$ dacă există un indice j , astfel încât $x_i=y_j$ pentru toți indicii $i < j$ și $x_j < y_j$ sau $p < k$ și $x_i=y_i$ pentru toți indicii $i \leq p$. De exemplu, 3 7 2 5 precedă lexicografic 3 7 4 1 6 2; 1 2 3 precedă lexicografic 2.

Soluție

Pentru a determina permutarea care succede permutarea p în ordine lexicografică, în primul rând trebuie să identificăm cel mai din dreapta element al permutării care poate fi mărit. Mai exact, trebuie să identificăm un indice i (cât mai mare posibil) astfel încât $p_i < p_{i+1} > p_{i+2} > \dots > p_n$. Vom interschimba p_i cu cel mai mic dintre elementele $p_{i+1}, p_{i+2}, \dots, p_n$ care este mai mare decât p_i . Dar aceasta nu este permutarea care succede imediat permutarea p în ordine lexicografică. Trebuie să construim cea mai mică secvență

(din punct de vedere lexicografic) constituită din elementele $p_{i+1}, p_{i+2}, \dots, p_n$. Pentru aceasta este suficient să considerăm aceste elemente în ordine crescătoare, adică să inversăm secvența $p_{i+1}, p_{i+2}, \dots, p_n$.

```
for (i=n-1; i && p[i]>p[i+1]; i--);
if (!i) cout<<"Nu există succesor!";
else
    //p[i] trebuie marit
    //determin cel mai mic elem mai mare decat p[i]
    for (j=n; p[j]<p[i]; j--);
    //p[j] se va interschimba cu p[i]
    aux=p[j]; p[j]=p[i]; p[i]=aux;
    //inversez elem p[i+1]\dots p[n]
    for (j=i+1, k=n; j<k; j++, k--)
        {aux=p[j]; p[j]=p[k]; p[k]=aux; }
```

12. Grup

Administratorul rețelei cu N calculatoare de la SRI împarte, din motive strategice, aceste calculatoare în mai multe grupuri. De fapt, important este doar numărul de grupuri și numărul de calculatoare din fiecare grup, așa că împărțirea este descrisă prin sirul numerelor de calculatoare din fiecare grup, ordonat crescător. Periodic, el procedează la o nouă împărțire pe grupe a calculatoarelor. Dintre toate împărțirile posibile ale calculatoarelor în grupuri putem alege ca următoare împărțire doar aceea a cărei descriere precedă sau succedă lexicografic imediat împărțirii curente.

Dându-se o împărțire în grupe a celor N calculatoare, determinați cele două variante candidate pentru împărțirea următoare.

(ONI, Bacău, 2001, clasa a IX-a)

Date de intrare

Din fișierul de intrare GRUP.IN se citesc de pe prima linie N (numărul total de calculatoare din rețea) și k (numărul de grupe), iar de pe cea de-a doua linie numărul de calculatoare din fiecare grupă:

N k

g_1 g_2 ... g_k

Date de ieșire

În fișierul de ieșire GRUP.OUT se vor afișa:

p	– numărul de grupe din împărțirea care precedă lexicografic imediat împărțirea dată;
h_1 h_2 ... h_p	– numărul de calculatoare din cele p grupe ale împărțirii precedente;
u	– numărul de grupe din împărțirea care succede lexicografic imediat împărțirea dată;
t_1 t_2 ... t_u	– numărul de calculatoare din cele u grupe ale împărțirii următoare.

Restricții

- $2 \leq N \leq 1000$

- $g_1+g_2+\dots+g_k = h_1+h_2+\dots+h_p = t_1+t_2+\dots+t_u = N$
- $1 \leq g_1 \leq g_2 \leq \dots \leq g_k; 1 \leq h_1 \leq h_2 \leq \dots \leq h_p; 1 \leq t_1 \leq t_2 \leq \dots \leq t_u;$
- $1 < k < N$
- $1 \leq p, u \leq N$

Timp de execuție: 1 secundă/test.

Exemplu

GRUP.IN

14 3
2 6 6

GRUP.OUT
3
2 5 7
2
2 12

Soluție

Numim partiție a unui număr natural n o succesiune de numere naturale nenule $g_1 \leq g_2 \leq \dots \leq g_k$ astfel încât $g_1+g_2+\dots+g_k=n$.

Dată fiind o partiție a unui număr natural n , problema cere să se determine partiția imediat următoare, respectiv partiția imediat precedentă în ordine lexicografică.

Vom citi partiția dată într-un vector g :

```
#define NMax 1001
int n, k, g[NMax];
ifstream fin("grup.in");
fin>>n>>k;
for (int i=1; i<=k; i++) fin>>g[i];
```

Să analizăm modul de obținere a partiției imediat următoare. Pentru aceasta, vom parcurge elementele partiției de la dreapta la stânga, determinând primul element care poate fi mărit. Evident, ultimul element $g[k]$ nu poate fi mărit. Prin urmare, vom încerca să îl mărim pe $g[k-1]$ căt mărim pe $g[k-1]$. Pentru a obține partiția următoare, ar trebui să îl mărim pe $g[k-1]$ cu 1 mai puțin posibil. Pentru a mări pe $g[k-1]$ cu 1 trebuie să putem plasa după $g[k-1]$ mai multe elemente cea mai mică partiție în ordine lexicografică a lui $g[k]-1$ (adică $g[k-1], g[k-1], \dots, g[k-1], g[k-1]+(g[k]-1) \dots g[k-1]$). Dacă nu este posibil să mărim cu 1 pe $g[k-1]$, următoarea partiție din punct de vedere lexicografic se obține înlocuind pe $g[k-1]$ cu $g[k-1]+g[k]$.

```
int s=g[k], j, p; //p reprezinta lungimea noii partitii
i=k-1; //generare cod urmator
if (g[i]+1 < s)
{ g[i]++; s--;
  for (j=i+1; s-g[i]>=g[i]; j++)
    { g[j]=g[i]; s-=g[i]; }
  if (s) g[j]=s;
```

```
else j--;
p=j; }
else
{ p=k-1; g[i]+=g[k]; }
```

Partiția precedentă din punct de vedere lexicografic se obține prin procedeul invers. Analizăm ultimul element $g[k]$ și verificăm dacă se poate împărți în două, adică $g[k]/2$ este cel puțin egal cu $g[k-1]$. În acest caz, partiția precedentă se obține înlocuind pe $g[k]$ cu $g[k]/2$, urmat $g[k]-g[k]/2$. Dacă nu, căutăm un element, începând de la dreapta către stânga, care să poată fi micșorat cu 1 unitate (adică pentru care $g[i]-1$ este cel puțin egal cu elementul din stânga sa $g[i-1]$). Când am depistat un astfel de element, partiția imediat precedentă din punct de vedere lexicografic se obține, decrementând elementul respectiv și înlocuind toate elementele care urmează după el cu un singur element egal cu suma lor +1.

```
if (g[k]/2>=g[k-1]) //generare cod precedent
{ g[k+1]=g[k]-g[k]/2;
  g[k]/=2; p=k+1; }
else
{ int s=g[k];
  for (int i=k-1; g[i]-1<g[i-1]; i--) s+=g[i];
  g[i]--;
  g[i+1]=s+1; p=i+1; }
```

13. Element majoritar

Fie $A = a_1, a_2, \dots, a_n$ o secvență de numere întregi. Fiind dat un element x , vom numi multiplicitatea a lui x în A numărul de apariții ale lui x în A . Un element se numește **majoritar** dacă multiplicitatea sa este mai mare decât $n/2$. Să se determine elementul majoritar dintr-un sir, dacă un astfel de element există.

Soluție

O primă idee ar fi de a calcula multiplicitatea pentru fiecare element din A (ceea ce ar necesita n^2 operații). Pentru a obține un algoritm liniar, parcugem sirul A de două ori: o dată pentru a determina un „candidat” la majoritate (singurul element care ar avea şanse să fie element majoritar), a doua oară pentru a „valida” candidatul (numărăm aparițiile candidatului în sir și comparăm cu $n/2$). Inițial presupunem că primul element este „candidat la majoritate”. La pasul i , variabila **Candidat** va conține candidatul la majoritate până la momentul i , iar variabila **Nr**, numărul de apariții necontracarurate ale variabilei **Candidat** în sirul A , din momentul în care a fost declarat candidat până la pasul i (fiecare apariție a unui element în sir diferit de **Candidat** va anula o apariție a candidatului). La compararea candidatului elementului $a[i]$ pot apărea trei situații:

- $a[i] = \text{Candidat}$, caz în care numărul de apariții ale candidatului crește;
- $a[i] \neq \text{Candidat}$, dar $\text{Nr} > 0$, caz în care anulăm o apariție a candidatului;
- $a[i] \neq \text{Candidat}$, iar $\text{Nr} = 0$, caz în care candidatul curent nu are şanse să fie majoritar, propunem drept candidat la majoritate elementul $a[i]$.

```
Candidat=a[0]; Nr=1; //determin un "candidat" la majoritate
//Nr este nr. curent de "voturi" pentru Candidat
for (i=1; i<n; i++)
    if (Candidat==a[i]) Nr++;
    else // "vot pentru"
        if (Nr) Nr--;
        else //Candidat nu are sanse sa fie majoritar
            //a[i] devine noul candidat la majoritate
            Candidat=a[i]; Nr=1;
```

Verificăm acum dacă acest candidat este majoritar:

```
for (Nr=i=0; i<n; i++)
    if (a[i]==Candidat) Nr++;
if (Nr>n/2) cout<<"Elementul majoritar este "<<Candidat;
else cout<<"Nu exista element majoritar";
```

14. Palindroame în baza b

Fie n și b două numere naturale ($n \leq 40, b < 10$). Să se genereze toate palindroamele de n cifre scrise în baza b .

Soluție

Prin adunare cu 1 în baza b , vom genera toate numerele scrise în baza b , care au $(n+1)/2$ cifre. Aceste numere reprezintă jumătăți de palindrom, pe care le vom afișa așa cum sunt, dar și „oglindite”, pentru a obține întreg palindromul.

```
#include <iostream.h>
void main()
{ int n, b, p[20], i;
  cout<<"n, b= "; cin>>n>>b;
  ofstream fout("palindr.out");
  int mijloc=(n+1)/2, gata=0;
  p[0]=1; //cel mai mic numar este 10...0
  for (i=1; i<mijloc; i++) p[i]=0;
  while (!gata)
    { //scriu palindromul curent
      for (i=0; i<mijloc; i++) fout<<p[i]; //direct
      for (i=mijloc-1-n%2; i>=0; i--) fout<<p[i]; //oglindit
      fout<<endl;
      //generez jumătatea urmatoare, adunand 1 în baza b
      for (i=mijloc-1; i>=0 && p[i]==b-1; i--) p[i]=0;
      if (i<0) gata=1; //numarul era b-1b-1...b-1
      else p[i]++;
    }
  fout.close();}
```

15. Premii

Organizatorii concursului au obținut de la sponsorii pentru premii N obiecte, de valori cunoscute. Ca premiu se pot oferi unul, două sau mai multe obiecte, valoarea premiului fiind egală cu suma valorilor obiectelor din care este constituit.

Determinați numărul de premii de valori diferite care se pot obține cu obiectele date.

Date de intrare

Fișierul de intrare PREMII.IN conține $N+1$ linii:

N – numărul de obiecte

v_1 – valoarea obiectului 1

v_2 – valoarea obiectului 2

...

v_N – valoarea obiectului N

Date de ieșire

În fișierul de ieșire PREMII.OUT se va afișa pe prima linie numărul de premii de valori diferite determinat.

Restricții

N număr natural, $0 < N < 200$

v_i numere naturale, $0 < v_i \leq 500$

Exemplu

PREMII.IN	PREMII.OUT
3	7
7	
20	
5	

Soluție

Observăm că sunt maxim 200 de obiecte, valoarea unui obiect fiind maxim 500. Prin urmare, cel mai valoros premiu care se poate obține are valoarea mai mică decât 100000. Vom construi mulțimea tuturor valorilor de premii posibile. Pentru a reprezenta o mulțime vom utiliza vectorul caracteristic: $s[i]=1$ dacă i aparține mulțimii și 0 altfel. Pentru a reține o informație binară (1/0) este suficient să utilizăm un singur bit. Prin urmare, pentru o mulțime cu 100000 de valori posibile sunt necesari $100000/8=12500$ octeți. Fiecare bit din această zonă de memorie va corespunde unei valori din domeniul [0, 100000]. Mai exact, bitul corespunzător valorii x se află în octetul $x/8$ și este bitul $x \% 8$ (considerând biți numerotați de la 0 la 7). Pentru a verifica dacă elementul x aparține mulțimii, trebuie să testăm dacă bitul corespunzător este 1 (verificăm dacă expresia $s[x/8] \& (1 << x \% 8)$ este diferită de 0). Pentru a adăuga un element x la mulțime, setăm bitul corespunzător ($s[x/8] |= 1 << (x \% 8)$).

Vom considera succesiv cele n valori. La pasul j vom reține în vectorul s mulțimea valorilor premiilor care se obțin adăugând valoarea $v[j]$ la valorile premiilor deja construite, precum și valoarea $v[j]$.

```

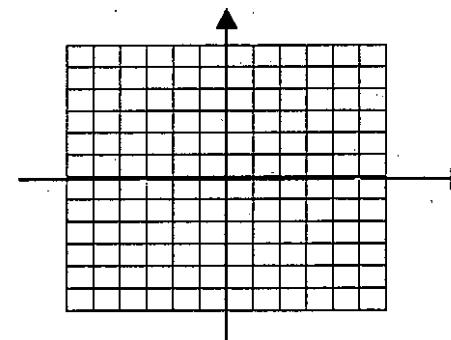
#include <fstream.h>
#include <string.h>
#define NMax 200
#define ValMax 500
#define DimMax 12501
char s[DimMax], temp[DimMax];
int n, i, j, k;
long nr, y, ultim;
int v[NMax];

void main()
{ifstream f("premii.in");
 f>>n; //citire
 for (i=0; i<n; i++) f>>v[i];
 f.close();
 ultim=-1;
 for (i=0; i<n; i++)
 /*creeaza premii noi adaugand valoarea v[i] la toate
 premiile existente; in temp obtin valorile noilor premii*/
 {for (j=0; j<=ultim; j++)
 if (s[j])
 for (k=0; k<8; k++)
 if (s[j]&(1<<k))
 //daca exista un pachet cu valoarea j*8+k
 {y=((long)j*8+k+v[i]);
 // adaug valoarea v[i]
 temp[y/8]|=1<<(y%8);
 if (y/8>ultim) ultim=y/8;
 temp[v[i]/8]|=1<<(v[i]%8);
 //formez un premiu numai cu v[i]
 if (v[i]/8>ultim) ultim=v[i]/8;
 for (j=0; j<=ultim; j++) s[j]=temp[j];
 }
 //scrie
 for (j=0; j<=ultim; j++)
 if (s[j])
 for (k=0; k<8; k++)
 if (s[j]&(1<<k)) nr++;
 ofstream fout("premii.out");
 fout<<nr;
 fout.close(); }

```

16. Păianjen

Să ne imaginăm o rețea planară, formată din noduri situate în punctele de coordonate întregi, fiecare nod fiind unit prin bare paralele cu axele de coordonate de cele 4 noduri vecine. Un păianjen este plasat inițial în originea sistemului de coordonate. La fiecare secundă, păianjenul se poate deplasa din nodul în care se află în unul din nodurile vecine.



Scriți un program care să determine în câte moduri se poate deplasa păianjenul din poziția inițială, într-o poziție finală dată prin coordonatele ei x și y , în timpul cel mai scurt.

(ONIG, 2001, clasele a VII-a – a VIII-a)

Exemple

Pentru $x=1$ și $y=2$, numărul de moduri determinat va fi 3. Pentru $x=2$ și $y=3$, numărul de moduri va fi 10.

Soluție

Cum păianjenul se deplacează numai de-a lungul barelor, el poate să ajungă (în timpul cel mai scurt) în poziția (x, y) numai din poziția $(x-1, y)$ sau din poziția $(x, y-1)$. Dacă notăm cu $nr(x, y)$ numărul de modalități prin care păianjenul poate ajunge în timp minim în poziția (x, y) , atunci deducem că:

$$\begin{aligned} nr(0, y) &= nr(x, 0) = 1; \\ nr(x, y) &= nr(x-1, y) + nr(x, y-1), \text{ pentru orice } x, y > 0. \end{aligned}$$

```

#include <iostream.h>
#define NMAX 21
unsigned long nr[NMAX][NMAX];
void main()
{ int x, y;
cout<<"x, y ="; cin>>x>>y;
for (int i=0; i<=x; i++) nr[i][0]=1;
for (int j=0; j<=y; j++) nr[0][j]=1;
for (i=1; i<=x; i++)
for (j=1; j<=y; j++)
nr[i][j]=nr[i-1][j]+nr[i][j-1];
cout<<"nr= "<<nr[x][y]<<endl;
cin.get(); }

```

Exercițiu

Rezolvați această problemă și pentru cazul în care dimensiunea maximă a rețelei este ≤ 100 . Veți observa că numărul de modalități crește foarte rapid și va fi necesar să implementați adunarea cu numere mari.

17. Baze

Există numere care au proprietatea că se scriu, în două baze diferite, prin trei cifre identice. De exemplu, numărul $273_{(10)}$ în baza 9 se scrie $333_{(9)}$, și în baza 16 se scrie $111_{(16)}$. Concepți un program care să determine toate numerele mai mici ca N care au această proprietate.

(Concurs Lugoj, 2001, clasele a VII-a - a VIII-a)

Date de intrare

Numărul N se citește de la tastatură.

Date de ieșire

În fișierul de ieșire BAZE.OUT se vor scrie numerele determinate, fiecare pe câte o linie. Pentru fiecare număr se vor scrie, separate prin câte un spațiu, numărul în baza 10 și cele două baze în care numărul respectiv are proprietatea din enunț.

Restricții

- $0 < N < 20000$

Exemplu

Pentru $N=300$ fișierul de ieșire va conține:

273 9 16

Soluție

Să considerăm $xxx_{(b)}$ scrierea cu 3 cifre identice a unui număr natural Z în baza b. Numărul natural scris în baza 10 este $Z=x \cdot b^2 + x \cdot b + x$. Cum $Z \leq N$ și $1 \leq x \leq b-1$, din inegalitatea $x \cdot b^2 + x \cdot b + x \leq N$ deducem baza maximă în care pot fi scrise numerele mai mici decât N cu 3 cifre identice: $x \cdot b^2 + x \cdot b + x \leq N \Rightarrow x \cdot (b^2 + b + 1) \leq N$. Cum $x \geq 1 \Rightarrow b^2 + b + 1 - N \leq 0$.

Rezolvând această inecuație de gradul II și ținând cont de faptul că baza trebuie să fie pozitivă, deducem că $b \leq (\sqrt{4 \cdot N - 3} - 1) / 2$. Să notăm cu BMax această limită.

Vom considera succesiv toate numerele de forma xxx , scrise în toate bazele mai mici decât BMax și le vom converti în baza 10 ($Z=x \cdot b^2 + x \cdot b + x$).

Pentru a verifica dacă un număr natural a putut fi scris cu 3 cifre identice în cel puțin două baze diferite, vom utiliza o matrice M cu N linii și 2 coloane.

```
char M[20000][2];
```

$M[Z][0]$ poate fi:

- 0, dacă Z nu poate fi scris cu 3 cifre identice în nici o bază;
- b_0 , dacă Z poate fi scris cu 3 cifre identice în baza b_0 ;
- BMax+1, dacă Z se scrie cu 3 cifre identice în mai mult de două baze.

$M[Z][1]$ poate fi:

- 0, dacă Z nu poate fi scris cu 3 cifre identice în două baze;
- b_1 , dacă Z poate fi scris cu 3 cifre identice în baza b_1 și în baza $M[Z][0]$.

```
BMax=(sqrt(4*N-3)-1)/2;
for (b=2; b<=BMax; b++)
    for (x=1; x<b; x++)
        Z=x*b*b+x*x+b;
        if (Z<=N)
            if (!M[Z][0]) M[Z][0]=b;
            else
                if (!M[Z][1]) M[Z][1]=b;
                else
                    M[Z][0]=BMax+1; }
```

// Z este numarul in baza 10

La afișare vom parcurge matricea M:

```
ofstream f("baze.out");
for (Z=7; Z<=N; Z++)
    if (M[Z][0] && M[Z][1] && M[Z][0]<=BMax)
        f<<Z<<' '<<(int)M[Z][0]<<' '<<(int)M[Z][1]<<endl;
f.close();
```

18. Numere

Într-un sir dat de n numere naturale, orice număr natural din intervalul inchis $[0, 1000000000]$ fie nu apare niciodată, fie apare de un număr de ori care este multiplu de k (dat). Excepție face un singur număr care apare de un număr de ori ce NU este multiplu de k. Scrieți un program care determină numărul căutat.

Date de intrare

Fișierul de intrare NUMERE.IN conține:

NUMERE.IN

n k

s₁

s₂

...

s_n

Semnificație

- valorile naturale n și k separate printr-un spațiu;
- următoarele n linii conțin cele n valori ce formează sirul.

Date de ieșire

Pe ecran se va afișa valoarea cerută.

Restricții

- $1 \leq n \leq 100000$
- $k > 1$
- $0 \leq s_i \leq 1000000000$

Exemple

NUMERE.IN

```
9 2
4
2
1
1
4
2
1
2
1
```

NUMERE.IN

```
14 3
5
1000000000
5
0
1000000000
0
7
7
7
0
1000000000
5
5
5
```

Se afișează
2

Se afișează
5

Observații
Citirea datelor din fișier durează maxim 2 secunde la evaluare. Timp maxim de execuție: 4 secunde/test.

(Lugoj, 2002, clasa a IX-a)

Soluție

Numărul mare de valori din fișier nu ne permite memorarea tuturor acestor valori într-un vector. De asemenea, intervalul în care se găsesc valorile din sir este foarte mare, deci nu putem construi un vector în care să reținem frecvențele de apariție ale valorilor. Prin urmare, singura soluție este de a reține, pe măsură ce citim valorile din sir informațiile care ne sunt utile într-o structură de date convenabilă.

În acest scop vom utiliza o matrice Nr cu 10 linii și 10 coloane, având semnificația $Nr[i][j] =$ numărul de valori din sir care au cifra i ($i \in \{0, 1, \dots, 9\}$) pe poziția j ($j \in \{0, 1, \dots, 9\}$, pozițiile în numere fiind numerotate începând cu cifra unităților). Această matrice va fi construită la citirea succesivă a valorilor din sir:

```
ifstream fin("numere.in");
fin>>n>>k;
long x;
int Lg;
for (long i=0; i<n; i++)
    {fin>>x;
     //determinam cifrele lui x
     Lg=0;
     do {Lg++; Nr[x%10][Lg]++;
          x/=10;} while (x);
    }
fin.close();
```

Pe baza informațiilor memorate în matricea Nr vom determina succesiv cifrele numărului care apare în sir, dar numărul lui de apariții nu este multiplu de k . În acest scop vom construi un vector Cifra, cu 10 componente, având semnificația $Cifra[i] =$ cifra care apare pe poziția i și care nu se repetă de un multiplu de k ori (conform datelor problemei există o singură astfel de cifră). Numărul căutat se obține afișând componentele vectorului Cifra în ordine inversă:

```
int i, j;
for (i=0; i<10; i++)
    for (j=0; j<10; j++)
        if (Nr[i][j] % k) Cifra[i]=j;
for (i=9; i>=0; i--) cout<<Cifra[i];
```

19. Problema spectacolelor

Managerul artistic al unui festival trebuie să selecteze o mulțime cât mai amplă de spectacole ce pot fi jucate în singura sală pe care o are la dispoziție. Știind că i s-au propus $n \leq 100$ spectacole și pentru fiecare spectacol i -a fost anunțat intervalul în care se poate desfășura $[s_i, f_i]$ (s_i reprezintă ora și minutul de început, iar f_i ora și minutul de final al spectacolului i) scrieți un program care să permită spectatorilor vizionarea unui număr cât mai mare de spectacole. De exemplu, dacă vom citi $n=5$ și următorii timpi:

```
12 30 16 30
15 0 18 0
10 0 18 30
18 0 20 45
12 15 13 0
```

Spectacolele selectate sunt: 5 2 4.

Soluție

Ordonăm spectacolele crescător după ora de final. Selectăm inițial primul spectacol (deci cel care se termină cel mai devreme). La fiecare pas selectăm primul spectacol neselectat, care nu se suprapune cu cele deja selectate (deci care începe după ce se termină ultimul spectacol selectat).

```
#include<iostream.h>
struct spectacol
    {int inceput, sfarsit, nr;} s[100];

void main()
{spectacol aux;
 int n, i, h, m, schimb, ultim;
 cout << "n="; cin >> n; //citire
 cout << "Introduceti inceputul și sfarsitul spectacolelor";
 for (i=0; i<n; i++)
    {s[i].nr=i+1; //transform timpul in minute
     cin >> h >> m; s[i].inceput=h*60+m;
      cin >> h >> m; s[i].sfarsit=h*60+m;}
 //ordonez spectacolele crescator dupa ora de final
```

```

do
    {schimb=0;
     for (i=0; i<n-1; i++)
        if (s[i].sfarsit>s[i+1].sfarsit)
            { aux=s[i]; s[i]=s[i+1]; s[i+1]=aux; schimb=1; }
    }
while (schimb);
cout << "Spectacolele selectate sunt:\n" << s[0].nr << ' ';
for (ultim=0, i=1; i<n; i++)
    if (s[i].inceput >= s[ultim].sfarsit)
        {cout << s[i].nr << ' '; ultim=i;}
}

```

Pentru rezolvarea acestei probleme am utilizat o metodă de programare importantă, denumită Greedy. În general, metoda Greedy se aplică problemelor de optimizare. Specificul acestei metode constă în faptul că se construiește soluția optimă pas cu pas, la fiecare pas fiind selectat (sau „înghesit”) în soluție elementul care pare „cel mai bun” la momentul respectiv, în speranță că această alegere locală va conduce la optimul global.

Algoritmii Greedy sunt foarte eficienți, dar nu conduc în mod necesar la o soluție optimă. Și nici nu este posibilă formularea unui criteriu general conform căruia să putem stabili exact dacă metoda Greedy rezolvă sau nu o anumită problemă de optimizare. Din acest motiv, orice algoritm Greedy trebuie însoțit de o demonstrație a corectitudinii sale. Demonstrația faptului că o anumită problemă are proprietatea alegerii Greedy se face de obicei prin inducție matematică.

20. Planificarea optimală a lucrărilor

Patronul unei firme a contractat n lucrări. În contractul fiecărei lucrări i sunt specificate termenul de predare t_i și penalizarea p_i care se plătește în cazul în care lucrarea nu este finalizată la termen. Execuția oricărui lucru necesită o unitate de timp, iar la firmă nu poate executa două lucrări în același timp.

Scrieți un program care să determine o planificare optimală a celor n lucrări contractate. O planificare este considerată optimală dacă penalizarea totală plătită în urma executării tuturor lucrărilor este minimă.

Soluție

Vom spune că o lucrare este *restantă* dacă execuția ei se termină după termenul de predare; altfel o vom numi *în termen*. Observăm că orice planificare poate fi adusă la o formă echivalentă (din punctul de vedere al penalizării totale) în care toate lucrările *în termen* precedă lucrările *restante*. Mai mult, orice planificare poate fi adusă în formă canonică, astfel încât lucrările *în termen* să preceadă lucrările *restante*, iar lucrările *în termen* să fie în ordinea crescătoare a termenelor de predare (dacă într-o planificare oarecare L_i , lucrarea *în termen* L_i are termenul de predare ulterior termenului de predare a lucrării *în termen* L_j , $i < j$, putem inversa în planificare execuția lucrărilor L_i și L_j , acestea rămânând *în termen*). Deci problema determinării unei planificări optimale se reduce la a determina lucrările *în termen* în ordinea crescătoare a termenelor de predare.

Pentru rezolvarea problemei vom utiliza metoda Greedy. Vom ordona lucrările descrescător după penalizări și vom construi mulțimea L a lucrărilor în termen selectând la fiecare pas prima lucrare (deci cea a cărei întârziere ar produce penalizarea cea mai mare), care poate fi executată în termen până la momentul curent, înseamnă în planificare lucrarea astfel încât lucrările să fie în ordinea crescătoare a termenelor de predare. Utilizăm următoarele variabile:

```

int n;                                //nr. de lucrari
int nt;                               // nr. de lucrari "in termen"
int nr;                                //numarul de lucrari restante
int t[NMax];                          //termenele de predare a lucrarilor
double p[NMax];                      //penalizarile pentru restante
int o[NMax];                          //ordinea lucrarilor in functie de penalizari
int R[LNMax]                         //multimea lucrarilor restante
int L[NMax];                          //multimea lucrarilor "in termen"

```

După citirea șiordonarea lucrărilor descrescător după penalizări, determinăm mulțimea lucrărilor în termen astfel:

```

nt=1; L[0]=o[0];
for (i=1;i<n; i++)
    { //incerc sa inserez lucrarea o[i] in L
    for (j=nt-1;j>=0&&t[L[j]]>(j+1)&&t[L[j]]>t[o[i]]; j--);
    //Lucrarea L[j] nu mai poate fi decalata
    j++;
    /*verific daca lucrarea o[i] ar fi "in termen" la
    momentul j */
    if (t[o[i]] > j)
    {
        /*inserez lucrarea o[i] in L pe pozitia j, decaland
        celelalte lucrari */
        for (k=nt++;k>j;k--) L[k]=L[k-1];
        L[j]=o[i];
    }
    else
        { //lucrare restanta
        ptotal+=p[o[i]];
        R[nr++]=o[i];
    }
}

```

21. Depozit

Considerăm un depozit care are n ($n \leq 1000$) camere, care conțin respectiv cantitățile de marfă C_1, C_2, \dots, C_n , care sunt numere naturale distincte. Scrieți un program care să determine un grup de camere cu proprietatea că suma cantităților de marfă pe care le conțin se poate împărtăși exact la cele n camioane care o transportă.

(ONI, Iași, 1993)

Soluție

Problema este particulară: solicită determinarea unei submulțimi a unei mulțimi cu n elemente, cu suma divizibilă tot cu n . Vom construi sumele S_1, S_2, \dots, S_n și resturile pe care acestea le dă prin împărțire la n astfel:

$$\begin{aligned} S_1 &= C_1 & \Rightarrow R_1 = S_1 \% n \\ S_2 &= C_1 + C_2 & \Rightarrow R_2 = S_2 \% n \\ &\dots \\ S_i &= C_1 + C_2 + \dots + C_i & \Rightarrow R_i = S_i \% n \\ &\dots \\ S_n &= C_1 + C_2 + \dots + C_n & \Rightarrow R_n = S_n \% n \end{aligned}$$

Cazul I: Există un rest $R_i = 0$. În acest caz, suma S_i este divizibilă cu n , prin urmare camerele solicitate sunt $1, 2, \dots, i$.

Cazul II: Toate resturile sunt diferite de 0. Prin urmare, R_1, R_2, \dots, R_n sunt n resturi care iau valori în mulțimea $\{1, 2, \dots, n-1\}$. În mod obligatoriu există cel puțin două resturi egale: $R_i = R_j$ ($i < j$), adică S_i și S_j produc același rest la împărțirea cu $n \Rightarrow$ suma $S_j - S_i$ este divizibilă cu n , deci camerele solicitate sunt $i+1, i+2, \dots, j$.

Observații

1. Soluția nu este unică. Procedeul prezentat produce o soluție posibilă.
2. Rezolvarea se bazează pe *Principiul cutiei al lui Dirichlet*¹.

```
#include <iostream.h>

void main ()
{ int SR[100], n, i, j, k, c, gasit;
  cout << "n="; cin >> n;
  SR[0]=0; //calculez sumele de la citire
  for (i=1; i<=n; i++)
    {cout << "C" << i << "="; cin >> c;
     SR[i]=SR[i-1]+c;}
  for (i=1; i<=n; i++) SR[i]%=n; //calculez resturile
  cout << "Solutia este " << endl;
  for (gasit=0, i=1; !gasit && i<=n; i++) //caut un rest = 0
    if (!SR[i])
      {gasit=1;
       for (k=1; k<=i; k++) cout << k << ' ';}
  if (!gasit) //cazul II, caut două resturi egale
    for (i=1; !gasit && i<n; i++)
      for (j=i+1; !gasit && j<=n; j++)
        if (SR[i]==SR[j])
          {gasit=1;
           for (k=i+1; k<=j; k++) cout << k << ' ';}
}
```

1. Matematicianul Peter Gustav Lejeune Dirichlet a enunțat următorul principiu: „Se consideră n obiecte care trebuie plasate în p cutii, unde $n > p \cdot k$, $k \in \mathbb{N}^*$. Oricum am plasa obiectele, există o cutie care va conține $k+1$ obiecte”.

22. Drumul regelui

Într-o partida de săh, jucată pe o tablă cu n linii și n coloane, regele a plecat din colțul din stânga-sus, notat $(1, 1)$, a parcurs toată tabla de săh ocupând fiecare câmp o singură dată și a revenit prin ultima mutare la poziția inițială. Unind centrele câmpurilor ocupate succesiiv de rege, s-a observat că se obține o linie poligonala fără autointersectii.

Reamintim că în săh la o mutare regele se poate deplasa doar în una din cele 8 poziții învecinate poziției curente, fără a ieși de pe tablă.

Vom considera că la o mutare, deplasarea regelui pe tablă de săh pe orizontală sau verticală este de lungime 1, iar deplasarea în celulele de pe diagonale este de lungime 2. Lungimea drumului parcurs de rege se obține prin însumarea tuturor deplasărilor efectuate.

Scrieti un program care să determine:

- lungimea maximă a unui drum pe care poate să îl parcurgă un rege pe tabla de săh, în condițiile de mai sus;
- un drum de lungime maximă.

Date de intrare

Dimensiunea tablei de săh n se citește de la tastatură.

Date de ieșire

Fisierul de ieșire REGE.OUT va conține, pe prima linie, despărțite printr-un singur spațiu, valorile a și b , cu semnificația „lungimea drumului este $a + b\sqrt{2}$ ”, iar pe următoarele n linii tabla de săh conținând valori de la 1 la n^2 indicând drumul regelui, sub forma:

a	b
1	P ₁₁ P ₁₂ ... P _{1n}
P ₂₁	P ₂₂ ... P _{2n}
...	
P _{n1}	P _{n2} ... P _{nn}

Restriții

- $1 \leq n \leq 100$
- a, b sunt numere naturale
- P_{ij} sunt numere naturale distincte din mulțimea $\{1, 2, \dots, n^2\}$

Exemplu

Pentru $n=3$ fișierul de ieșire REGE.OUT ar putea conține:

8	1
1	2 3
9	7 4
8	6 5

Soluție

Este evident că drumul cel mai lung se obține atunci când regele face numărul maxim de mutări pe diagonală. Vom numerota câmpurile *marginale* astfel:

- numerotăm câmpul $(1, 1)$ cu 1;
- deoarece regele pleacă din $(1, 1)$ și se va întoarce aici, cel puțin un segment din cele două este pe orizontală (sau verticală); numerotăm acest câmp cu 2;
- procedând la fel în continuare, vom obține o orientare a drumului parcurs de rege și o numerotare a câmpurilor *marginale* de la 1 la $4 \cdot (n-1)$.

Pentru $n=4$ se obține:

1	2	3	4
12		5	
11		6	
10	9	8	7

Demonstrăm că acesta este numărul *minim* de mișcări pe orizontală (sau verticală) – deci de lungime 1. Pentru aceasta arătăm mai întâi că primul câmp marginal care se întâlnește pornind din câmpul marginal k este câmpul marginal $k+1$ (pentru câmpul $4 \cdot (n-1)$ acesta este câmpul 1). Pentru $k=1$ afirmația este adevărată datorită alegerii câmpului 2. Fie h primul câmp marginal întâlnit după plecarea din câmpul k ($k>1$). Dacă $h=k-1$, drumul de la k la $k-1$, împreună cu cel de la $k-1$ la k (de la pasul anterior), și care nu conține alte câmpuri marginale, formează un drum închis care nu trece prin $k+1$; deci nu este corect. Dacă $h \neq k-1$ și $h \neq k+1$, drumul între câmpurile k și h împarte tabla în două zone; trecând din h în una din zone, regele nu mai are acces în cealaltă zonă fără a intersecta drumul deja parcurs, din nou incorrect. Rezultă că singura posibilitate este $h=k+1$.

Având în vedere cele de mai sus, întregul drum se împarte în $4 \cdot (n-1)$ sectoare de drum, de la câmpul k la câmpul $k+1$ ($k=1, 2, \dots, 4 \cdot (n-1)$). Dar trecerea de la câmpul k la câmpul $k+1$ se face cu *schimbarea culorii*, care este posibilă doar printr-o mișcare orizontală (sau verticală). Deci fiecare din cele $4 \cdot (n-1)$ sectoare de drum trebuie să conțină *minim* o mișcare de lungime 1. Ceea ce înseamnă că numărul *minim* de mișcări de lungime 1 este $4 \cdot (n-1)$. Rezultă simplu că numărul *maxim* de mișcări pe diagonală este $n^2 - 4 \cdot (n-1) = (n-2)^2$. Deci $a=4 \cdot (n-1)$, $b=(n-2)^2$ și drumul de lungime *maximă* va fi $4 \cdot (n-1) + \sqrt{2} \cdot (n-2)^2$.

```
#include <iostream.h>
#include <iomanip.h>
int Pas, n, l, c;
long a[101][101];

void main()
{
    ifstream f("REGE.IN");
    f>>n; f.close();
    ofstream g("REGE.OUT");
    g<<4*(n-1)<<' ' <<(n-2)*(n-2)<<endl;
    for (c=1; c<=n; c++) a[1][c]=++Pas;
    for (l=2; l<=n; l++) a[l][n]=++Pas;
    c=n-1; a[n][c]=++Pas; l=n;
}
```

```
while (c>1)
    {c--; a[l][c]=++Pas;
     while (c<n-1){c++; l--; a[l][c]=++Pas;}
     if (l>2)
        {l--; a[l][c]=++Pas;
         while (l<n) {l++; c--; a[l][c]=++Pas;}
     }
     else
     if (n%2)
        {c--; a[l][c]=++Pas;
         while (l<n-1){l++; c--; a[l][c]=++Pas;}
     }
}
while (l>2)
{l--; a[l][c]=++Pas;
 while (l>2) {c++; l--; a[l][c]=++Pas;}
if (c>=2)
    {c--; a[l][c]=++Pas;
     while (c>1) {l++; c--; a[l][c]=++Pas;}
    }
}
for (l=1; l<=n; l++)
{for (c=1; c<=n; c++) g<<setw(4)<<a[l][c];
 g<<endl;}
g.close();}
```

23. Colorare

Fie n segmente închise situate pe o același dreaptă. Scrieți un program care să determine numărul minim de culori necesare pentru a colora cele n segmente, astfel încât oricare două segmente care se intersectează să fie colorate diferit și, de asemenea, să determine o astfel de colorare.

Soluție

Ordonăm extremitățile segmentelor după coordonată. Parcurgem punctele în ordine, reținând la fiecare moment lista segmentelor „deschise” (acestea se suprapun). Dacă punctul curent este extremitate inițială a unui segment, inserăm segmentul respectiv în listă, și-l colorăm cu prima culoare disponibilă. Dacă punctul este extremitate finală a unui segment, eliminăm segmentul respectiv din lista segmentelor „deschise” și disponibilizăm culoarea acestuia.

```
#include <iostream.h>
#define NMax 1001
int n, nrc, už[NMax], c[NMax], s[NMax], ns;
struct Punct
{
    float x; //coordonata
    int i;//numarul segmentului caruia ii apartine punctul
} o[2*NMax], aux;
```

```

void main()
{cin>>n;
 for (int i=0; i<n; i++)
 {cin>>o[2*i].x; o[2*i].i=i+1; cin>>o[2*i+1].x;
 o[2*i+1].i=-i-1;}
 int nr=2*n, ok, j, k;
 do {ok=1;           //sortare puncte dupa coordonata
      for (i=0; i<nr-1; i++)
       if (o[i].x>o[i+1].x)
        {aux=o[i]; o[i]=o[i+1]; o[i+1]=aux; ok=0;}
     }
 while (!ok);
 for (i=0; i<nr; i++)          //determinare culori
 if (o[i].i>0)
 {s[nst++]=o[i].i;             //inserez in s
  if (ns>nrc) nrc=ns;
  //colorez segmentul o[i].i
  for (k=1; uz[k]; k++); //caut o culoare disponibila
  c[o[i].i]=k; uz[k]=1; }
 else                         //extrag din s
 {uz[c[-o[i].i]]=0; //culoarea devine disponibila
  for (j=0; s[j]!=-o[i].i; j++);
  for (k=j+1; k<ns; k++) s[k-1]=s[k];
  ns--;
 cout<<nrc<<endl;
 for (i=1; i<=n; i++) cout<<c[i]<<endl;}

```

Probleme propuse

1. Cât mai mulți

Fie n un număr natural nenul și un sir de n numere naturale $a=a_1, a_2, \dots, a_n$. Să se determine numerele din sirul a care au un număr maxim de divizori primi.

2. Medii aritmetice

Fie n un număr natural și o succesiune de n numere naturale $a=a_1, a_2, \dots, a_n$. Să se înlocuască fiecare element al sirului a cu media aritmetică a celorlalte $n-1$ elemente din sir.

3. Număr

Se citește de la tastatură un număr natural scris în baza 10, având maxim 80 de cifre. Scrieți un program care să afișeze cel mai mare număr natural care se poate construi din cifrele distincte ale numărului citit.

4. Antene

În legendarul ținut al Vrancei, au fost instalate s antene ale unei companii de telefonia mobilă. Fiecare antenă este situată într-un punct de coordonate întregi și emite pe o arie

circulară de rază dată. Orice antenă are o perioadă de funcționare proprie, urmată de o perioadă egală de revizie. Se consideră un moment $t=0$ la care toate antenele își încep funcționarea. Deținătorii de telefoane mobile din tabăra Gălăciuc sunt interesați să afle care dintre antene asigură semnal în tabără și dacă acoperirea este permanentă.

(ONIG, Gălăciuc, 2001, clasa a VIII-a)

Se citesc de la tastatură:

s – numărul de antene

$x_1 \ y_1 \ r_1 \ p_1$ – coordonatele (abscisa, respectiv ordonata), raza și

$x_2 \ y_2 \ r_2 \ p_2$ – perioada proprie ale fiecărei antene,

\dots

$x_s \ y_s \ r_s \ p_s$

$x_G \ y_G$ – coordonatele taberei Gălăciuc

Se vor afișa pe ecran:

- numerele de ordine ale antenelor care asigură acoperirea taberei Gălăciuc;
- mesajul DA în cazul în care tabăra are acoperire permanentă sau mesajul NU urmat de primul moment, t_{\min} , la care nu mai există semnal de la nici una dintre antene, în caz contrar.

Restricții

- $0 < s \leq 20$
- $0 \leq x_i, y_i, r_i \leq 100$
- $0 \leq x_G, y_G \leq 100$
- $0 \leq p_i \leq 1000000$

Exemplu

Pentru $s=4$ și antenele:

$x_1=2$	$y_1=2$	$r_1=3$	$p_1=3$
$x_2=3$	$y_2=3$	$r_2=4$	$p_2=5$
$x_3=0$	$y_3=2$	$r_3=5$	$p_3=4$
$x_4=2$	$y_4=0$	$r_4=3$	$p_4=2$
$x_G=0$	$y_G=0$		

se afișează:

1 3 4

NU 15

5. Salariul unui miner

În anul 1990 salariul unui miner era de forma 22..2 (număr format cu n cifre de 2, n număr natural dat). După un an salariul se mărește cu 22..2 (format cu $n-1$ cifre de 2), după 2 ani se mărește cu 22..2 (format cu $n-2$ cifre de 2) și așa mai departe.

- Să se afișeze salariul minerului în 1990;
- Să se afișeze salariul minerului după k ani ($k \leq n$), k număr natural dat;
- Să se determine numărul cel mai mare format numai din cifre impare ale salariului unui miner din anul 1990+x, cu x număr natural citit de la tastatură.

(Concurs PACO, București, 1997, clasa a V-a)

Exemplu

Pentru $n=6$, $k=3$, $x=5$ se va afișa pe ecran:

- a) 222222
- b) 246888
- c) 91

6. Panglica

Gigel are o panglică alcătuită din benzii de 1 cm lățime, colorate în diverse culori. Panglica are N benzi colorate cu C culori, culori pe care le vom numera de la 1 la C . Gigel vrea ca la ambele capete ale panglicii să aibă aceeași culoare, dar cum nu poate schimba culorile benzilor, singura posibilitate rămâne să tăiere unor bucăți de la capete. Scrieți un program care să determine modul de sătire a panglicii astfel încât la cele două capete să fie benzi de aceeași culoare, iar lungimea panglicii obținute să fie maximă.

Date de intrare

Fisierul de intrare PANGLICA.IN conține:

- pe prima linie numerele naturale N și C separate printr-un spațiu;
- pe următoarele N linii descrierea panglicii: pe fiecare linie un număr natural de la 1 la C , reprezentând în ordine culorile benzilor ce alcătuiesc panglica.

Date de ieșire

Fisierul de ieșire PANGLICA.OUT va conține următoarele 4 numere:

- pe prima linie numărul de benzi rămasă;
- pe linia a doua numărul culorii care se află la capete;
- pe linia a treia câte benzi trebuie sătate de la începutul panglicii inițiale;
- pe linia a patra câte benzi trebuie sătate de la sfârșitul panglicii inițiale.

Restrictions și precizări

- $2 \leq N \leq 10000$
- $1 \leq C \leq 200$
- Dacă există mai multe soluții alegeți pe cea în care se tăie cât mai puțin din partea de început a panglicii.

Exemplul 1

PANGLICA.IN	PANGLICA.OUT
6 3	4
1	2
2	1
1	1
3	2
2	2
3	

Exemplul 2

PANGLICA.IN	PANGLICA.OUT
5 2	4
1	2
2	1
1	1
2	0
2	

Timp maxim de execuție: 1 secundă/test.

(ONIG, 2002, clasa a VII-a)

7. Ferma animalelor

Deși primul din cele 7 precepte era „Orice merge pe două picioare și dușman”, porcii au hotărât că este timpul să exporte din producția de pertinax de la fermă. Au încheiat la oraș n ($1 \leq n \leq 5000$) contracte pentru cantitățile a_1, a_2, \dots, a_n . Tovărașul Squeler a desemnat pentru fiecare contract câte un porc, care urma să coordoneze tranzacția și să încaseze banii, și în măgarii care urmău să transporte pertinaxul la oraș. Transportul era totuși o problemă, pentru că măgarul este un animal cu un foarte dezvoltat simț al dreptății: nici un măgar nu ar fi vrut să transporte un bit mai mult decât ceilalți. S-a hotărât deci că se vor face mai multe transporturi, astfel încât la fiecare transport toți măgarii disponibili să transporte o aceeași cantitate. După fiecare transport, porcii ai căror pertinax a fost transportat integral urmău să rămână în oraș împreună cu măgarii lor, pentru a finaliza afacerea. Din acest motiv, cantitatea de pertinax aferentă unui contract nu poate fi transportată parțial.

Scrieți un program care să determine o modalitate de împărțire a pertinaxului pe transporturi.

Date de intrare

Datele de intrare se citesc din fisierul de intrare FERMA.IN. Pe prima linie se află n , numărul de contracte încheiate. Pe următoarele n linii se găsesc cantitățile contractate a_1, a_2, \dots, a_n ($a_i \in \{1, 2, \dots, 10000\}$, $\forall i \in \{1, 2, \dots, n\}$), către una pe linie.

Date de ieșire

Rezultatele vor fi afișate în fisierul FERMA.OUT. Fisierul de ieșire va conține mesajul NU EXISTA SOLUTIE sau va conține pe prima linie numărul de transporturi efectuate t ; pe următoarele linii sunt afișate cele t transporturi. Pentru fiecare transport este afișat pe prima linie n_t , numărul de contracte onorate la transportul respectiv, iar pe următoarele n_t linii cantitățile de pertinax contractate.

Exemplu

FERMA.IN	FERMA.OUT
7	3
13	3
2	2
4	4
29	29
17	2
6	6
10	10
	2
	13
	17

Observație

Cantitățile a_1, a_2, \dots, a_n sunt exprimate în MwKhTx. Un MwKhTx de pertinax este indivizibil.

Timp de execuție: 1 secundă/test.

8. Tablou

Să considerăm tablouri pătratice, cu n linii și n coloane ($n \leq 50$), construite ca în exemplile care urmează:

$n=1$	<table border="1"><tr><td>1</td></tr></table>	1
1		

$n=2$	<table border="1"><tr><td>1</td><td>1</td></tr><tr><td>1</td><td>2</td></tr></table>	1	1	1	2
1	1				
1	2				

$n=3$	<table border="1"><tr><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>1</td><td>3</td><td>6</td></tr></table>	1	1	1	1	2	3	1	3	6
1	1	1								
1	2	3								
1	3	6								

$n=4$	<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>1</td><td>3</td><td>6</td><td>10</td></tr><tr><td>1</td><td>4</td><td>10</td><td>20</td></tr></table>	1	1	1	1	1	2	3	4	1	3	6	10	1	4	10	20
1	1	1	1														
1	2	3	4														
1	3	6	10														
1	4	10	20														

$n=5$	<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr><tr><td>1</td><td>3</td><td>6</td><td>10</td><td>15</td></tr><tr><td>1</td><td>4</td><td>10</td><td>20</td><td>35</td></tr><tr><td>1</td><td>5</td><td>15</td><td>35</td><td>70</td></tr></table>	1	1	1	1	1	1	2	3	4	5	1	3	6	10	15	1	4	10	20	35	1	5	15	35	70
1	1	1	1	1																						
1	2	3	4	5																						
1	3	6	10	15																						
1	4	10	20	35																						
1	5	15	35	70																						

Deducreți regula după care sunt construite tablourile de mai sus și scrieți un program care să citească de la tastatură pe n și să afișeze pe ecran elementul de pe linia n și coloana n a tabloului cu n linii și n coloane, construit după regula determinată.

(ONIG, Gălăciuc, 2001, clasa a VII-a)

9. Eliminare cifră

Fie n un număr natural de maxim 100 de cifre. Scrieți un program care să determine cel mai mare număr natural care se poate obține din n prin eliminarea unei singure cifre.

10. Pătrate

Se consideră un dreptunghi de dimensiuni $m \times n$ împărțit în $m \cdot n$ pătrate de latură 1 ($2 \leq m, n \leq 100$). Unele dintre aceste pătrate au trasată diagonala NV-SE, adică $\boxed{\square}$, altele diagonala NE-SV, adică \blacksquare , altele nu au nici o diagonală.

Scrieți un program care să determine câte pătrate formează aceste diagonale.

Date de intrare

Fișierul de intrare PATRATE.IN conține pe prima linie numerele naturale m și n , iar pe următoarele m linii descrierea dreptunghiului: pe fiecare linie câte n numere, separate prin spații, având valoarea 0, 1 sau 2, 0 însemnând pătrat fără diagonală, 1 – pătrat cu diagonala NV-SE; 2 – pătrat cu diagonala NE-SV.

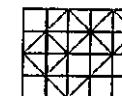
Date de ieșire

Fișierul PATRATE.OUT va conține pe prima linie numărul total de pătrate formate de diagonale, iar pe liniiile următoare numărul de pătrate de latură $\sqrt{2}$, $2\sqrt{2}$, $3\sqrt{2}$ etc. până la epuizarea tuturor păratelor. Dacă nu există pătrate de o anumită dimensiune scrieți pe linia respectivă 0.

Exemplu

PATRATE.IN
4 5
0 2 1 2 1
2 1 2 1 2
1 2 0 2 0
0 1 2 0 0

PATRATE.OUT
4
3
1

Figură

Timp maxim de execuție: 1 secundă/test.

(ONIG, Gălăciuc, 2002, clasa a VIII-a)

11. Cut & Paste

Să considerăm un fișier constituit din N linii, pe fiecare linie fiind câte un număr natural. Inițial, pe linia 1 se află valoarea 1, pe linia 2 valoarea 2 și.a.m.d., pe linia N se află valoarea N .

Fișierul inițial a fost modificat cu ajutorul unui editor de texte, prin efectuarea a M operații de tip *cut & paste*.

O operație *cut & paste* constă în selectarea unui grup de linii consecutive, eliminarea lor din poziția inițială și inserarea lor într-o altă poziție în document.

Scrieți un program care, pentru o secvență de operații *cut & paste* dată, determină conținutul primelor 10 linii din fișierul obținut după efectuarea operațiilor.

Date de intrare

Fișierul de intrare CP.IN conține

CP.IN
N M
l₁ l_{f1} unde₁
l₂ l_{f2} unde₂
...
l_M l_{fM} unde_M

Semnificație

numărul de linii din fișier și numărul de operații efectuate
operațiile efectuate: l_{i,k} și l_{f,k} – linia de început și linia de sfârșit ale celui de al k -lea grup de linii selectat
unde_k – linia după care se va insera grupul de linii specificat

Date de ieșire

Fișierul de ieșire CP.OUT va conține 10 linii reprezentând numerele scrise pe primele 10 linii din fișierul obținut după efectuarea operațiilor.

Restrictii

- $10 \leq N \leq 100000$
- $1 \leq M \leq 1000$
- $1 \leq l_{i,k} \leq l_{f,k} \leq N, 0 \leq u_{nde,k} \leq N - (l_{f,k} - l_{i,k} + 1)$ (pentru $\forall k \in \{1, 2, \dots, M\}$)
- Dacă $u_{nde,k} = 0$ deducem că inserarea se va face la începutul documentului.

Exemple

CP.IN	CP.OUT	CP.IN	CP.OUT	CP.IN	CP.OUT
15 1	1	13 3	6	1000 6	801
1 15 0	2	6 12 1	7	3 7 4	802
	3	2 9 0	8	1 100 57	803
	4	10 13 8	9	50 60 200	804
	5		10	63 70 500	101
	6		11	1 800 4	102
	7		12	7 77 98	36
	8		2		37
	9		3		38
10			4		39

Timp maxim de execuție: 1 secundă/test

(ONIG, Gălăciuc, 2002, clasa a VIII-a)

12. Submulțimea următoare

Fie n un număr natural nenul, k un alt număr natural, $k \leq n$ și $\{c_1, c_2, \dots, c_k\}$ o submulțime de k elemente a mulțimii $\{1, 2, \dots, n\}$ ($c_1 < c_2 < \dots < c_k$). Determinați submulțimea imediat următoare din punct de vedere lexicografic.

13. Parantezare

Fie n un număr natural par și $p_1 p_2 \dots p_n$ o succesiune de n paranteze rotunde care se închid corect. Determinați parantezarea imediat următoare din punct de vedere lexicografic, precum și numărul de ordine al acestei parantezări, în cadrul mulțimii tuturor succesiunilor de n paranteze rotunde care se închid corect, ordonate lexicografic.

14. Cod de identificare

Pentru a concura cu numărul de serie de la procesoarele Intel Pentium III, Advanced Micro Devices a stabilit un sistem de identificare pentru noile procesoare cu numele de cod Thunderbird. Fiecare firmă distribuitoare primește o mulțime de litere (de exemplu $\{a, m, x\}$) din care va trebui să-și formeze codurile proprii de identificare. Firmelor li se impune exact de câte ori trebuie să apară fiecare literă în aceste coduri. De exemplu, o firmă trebuie să formeze identificatori care să conțină exact 3 litere a , 2 litere m și 1 literă x . Scrieți un program care, cunoscând un anumit cod dat, determină următorul cod corect în ordine lexicografică, dacă există un astfel de cod următor.

(ONI, Constanța, 2000, clasa a IX-a)

Date de intrare

Fișierul de intrare COD.IN conține un cod. Codurile sunt formate din cel mult 100 de litere mici ale alfabetului latin.

Date de ieșire

Fișierul de ieșire COD.OUT va conține o singură linie pe care se va afla codul următor; dacă nu există un astfel de cod, atunci în fișier se va scrie 'Este ultimul cod.'

Exemple

COD.IN	COD.OUT
amaaxm	amamax
xmmaaa	Este ultimul cod.

15. Cel mai lung prefix

Anumite structuri biologice pot fi reprezentate prin secvențe de constituenți. Constituenții sunt notați cu litere mari. Biologii sunt interesați în descompunerea unei secvențe lungi în altele mai scurte, denumite primitive.

Spunem că o secvență S poate fi compusă dintr-o mulțime dată P de primitive, dacă există n primitive p_1, p_2, \dots, p_n în P astfel încât S să se obțină prin concatenarea lor. Aceeași primitivă poate apărea în concatenare de mai multe ori sau poate lipsi. De exemplu, secvența ABABACABAAB poate fi obținută din mulțimea de primitive $\{A, AB, BA, CA, BBC\}$.

Primele K caractere ale unei secvențe S constituie prefixul lui S de lungime K . Scrieți un program care acceptă la intrare un set P de primitive și o secvență T formată din constituenți. Programul trebuie să calculeze lungimea celui mai lung prefix al secvenței T care poate fi obținut cu primitive din P .

Date de intrare

Datele de intrare se citesc din două fișiere. Fișierul INPUT.TXT descrie mulțimea P de primitive, iar fișierul DATA.TXT conține secvența T de examinat.

Prima linie din INPUT.TXT conține numărul natural N , care reprezintă numărul de primitive din P ($1 \leq N \leq 100$). Fiecare dintre următoarele N linii conține o succesiune de maxim 20 litere mari, care reprezintă o primitivă. Cele N primitive sunt distincte.

Fișierul DATA.TXT conține o succesiune de maxim 500000 litere mari, care reprezintă secvența de examinat. Sfârșitul secvenței este marcat de apariția caracterului punct.

Date de ieșire

Scrieți pe ecran lungimea celui mai lung prefix al lui T care poate fi format din elemente ale mulțimii P .

Exemplu

INPUT.TXT	DATA.TXT	Rezultat
5 A AB BBC CA BA	ABABACABAABC.	11

16. Poarta

Se consideră harta universului ca fiind o matrice cu 250 de linii și 250 de coloane. În fiecare celulă se găsește o așa numită poartă stelară, iar în anumite celule se găsesc echipaje ale porții stelare. La o deplasare, un echipaj se poate deplasa din locul în care se află în oricare alt loc în care se găsește o sau două poarte, în cazul nostru în orice altă poziție din matrice. Nu se permite situaarea simultană a mai mult de un echipaj într-o celulă. La un moment dat un singur echipaj se poate deplasa de la o poartă stelară la alta.

Dându-se un număr p ($1 < p < 5000$) de echipaje, pentru fiecare echipaj fiind precizate poziția inițială și poziția finală, determinați numărul minim de deplasări necesare pentru ca toate echipajele să ajungă din poziția inițială în cea finală.

Datele de intrare

Se citesc din fișierul text POARTA.IN în următorul format:

- pe prima linie numărul natural p reprezentând numărul echipaje,
- pe următoarele p linii câte 4 numere naturale, primele două reprezentând coordonatele poziției inițiale a unui echipaj (linie coloană), următoarele două reprezentând coordonatele poziției finale a aceluiași echipaj (linie coloană).

Datele de ieșire

Pe prima linie a fișierului text POARTA.OUT se scrie un singur număr reprezentând numărul minim de deplasări necesar.

Exemplu

POARTA.IN	POARTA.OUT	Figură																																																																								
1 2 3 4 6 5 3 9 3 4 1 2	4	<table border="1"> <tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td></tr> <tr><td>1</td><td>e1</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td>2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td>3</td><td></td><td>e3</td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td>4</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td>5</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td>6</td><td></td><td></td><td>e2</td><td></td><td></td><td></td><td></td><td></td></tr> <tr><td>7</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> </table>	1	2	3	4	5	6	7	8	9	1	e1								2									3		e3							4									5									6			e2						7								
1	2	3	4	5	6	7	8	9																																																																		
1	e1																																																																									
2																																																																										
3		e3																																																																								
4																																																																										
5																																																																										
6			e2																																																																							
7																																																																										

Observații

- Coordonatele pozițiilor inițiale și finale ale echipajelor sunt numere naturale din intervalul $[1, 250]$.
- Pozițiile inițiale ale celor p echipaje sunt distincte două câte două.
- Pozițiile finale ale celor p echipaje sunt distincte două câte două.

Timp maxim de execuție: 1 secundă/test.

(OJ, 2002, clasa a IX-a)

17. Mouse

Un experiment urmărește comportarea unui șoricel pus într-o cutie dreptunghiulară, împărțită în $m \times n$ cămăruțe egale de formă pătrată. Fiecare cămăruță conține o anumită cantitate de hrana. Șoricelul trebuie să pornească din colțul $(1,1)$ al cutiei și să ajungă în colțul opus, mânând cât mai multă hrana. El poate trece dintr-o cameră în una alăturată (două camere sunt alăturate dacă au un perete comun), mânând toată hrana din cămăruță atunci când intră și nu intră niciodată într-o cameră fără hrana. Stabiliiți care este cantitatea

maximă de hrana pe care o poate mâncă și traseul pe care îl poate urma pentru a culege această cantitate maximă.

Datele de intrare

Fișierul de intrare MOUSE.IN conține pe prima linie două numere m și n reprezentând numărul de linii respectiv numărul de coloane ale cutiei, iar pe următoarele $m \times n$ linii cele $m \times n$ numere reprezentând cantitatea de hrana existentă în fiecare cămăruță, câte n numere pe fiecare linie, separate prin spații. Toate valorile din fișier sunt numere naturale între 1 și 100.

Datele de ieșire

În fișierul de ieșire MOUSE.OUT se vor scrie pe prima linie două numere separate printr-un spațiu: numărul de cămăruțe vizitate și cantitatea de hrana maximă culeasă. Pe următoarele linii se va scrie un traseu posibil pentru cantitatea dată, sub formă de perechi de numere (linie coloană) începând cu 1 1 și terminând cu $m \times n$.

Exemplu

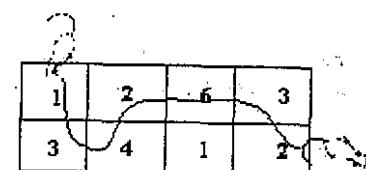
MOUSE.IN

2 4
1 2 6 3
3 4 1 2

MOUSE.OUT

7 21
1 1
2 1
2 2
1 2
1 3
1 4
2 4

Figură



Timp maxim de execuție: 1 secundă/test.

(OJ, 2002, clasa a IX-a)

18. Pătrate

Se dă un număr întreg n ($6 \leq n \leq 100$). Scrieți un program care să determine o modalitate de partionare a unui pătrat în n pătrate. Programul va citi de la tastatură numărul n și va afișa în fișierul PATRAT.OUT un tablou pătratic T ale cărui elemente sunt numere naturale din mulțimea $M=\{1, 2, \dots, n\}$. Fiecare element $x \in M$ este utilizat în T pentru reprezentarea unui pătrat din partitie.

De exemplu, pentru $n=10$ fișierul de ieșire poate să conțină:

1 1 1 1 4 4 7 8
1 1 1 1 4 4 9 10
1 1 1 1 5 5 6 6
1 1 1 1 5 5 6 6
2 2 2 2 3 3 3 3
2 2 2 2 3 3 3 3
2 2 2 2 3 3 3 3
2 2 2 2 3 3 3 3

(Tabăra de pregătire, Mediaș, 1999)

19. Perechi

Se consideră sirul numerelor naturale prime:

2, 3, 5, 7, 11, 13, 17, ...

Din el se construiește un al doilea sir astfel: fiecare număr prim de pe o poziție impară se concatenează cu numărul prim următor. Se obține sirul:

23, 57, 1113, ...

Din acest sir se formează un al treilea sir care conține doar numerele prime:

23, 3137, ...

Scrieți un program care să citească un număr natural n ($1 \leq n \leq 100$) și să determine al n-lea număr din acest al treilea sir.

(Baraj, Mediaș, 1999)

20. Spre culmi

Se dă un vector cu $N \leq 10000$ numere întregi, cuprinse între 1 și 50000. Să se particioneze acest vector în cât mai puține subșiruri strict crescătoare.

(Tabăra de pregătire, Mediaș, 1999)

21. Multiplu

Fie n un număr natural ($n \leq 1000$). Determinați un multiplu al lui n a căruia scriere în baza 10 conține numai cifrele 1 și 0.

Subprograme. Recursivitate**1. Bacalaureat**

Din cei n elevi de clasa a XII-a de la Liceul de Informatică, doar k vor susține examenul de bacalaureat în liceu, ceilalți fiind repartizați la alte centre. Pentru n și k două numere naturale date de la tastatură ($k \leq n < 50$):

- Scrieți o funcție nerecursivă care să calculeze numărul de posibilități de a selecta cei k elevi care urmează să susțină bacalaureatul în liceul lor!
- Scrieți o funcție recursivă care să rezolve aceeași problemă!
- Realizați o comparație între cele două funcții din punctul de vedere al eficienței!
- Scrieți un program (recursiv sau iterativ) care să genereze toate posibilitățile de a selecta cei k elevi care vor susține examenul de bacalaureat în liceul lor (considerați elevii numerotați de la 1 la n).

Soluție

a. Soluția nerecursivă se poate baza pe formula de recurență alăturată, care stă la baza algoritmului de calcul al combinărilor cunoscut sub denumirea „triunghiul lui Pascal”. De exemplu, triunghiul lui Pascal pentru $n=5$, este

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1

Observați că pe linia i (i variind de la 0 la n) în triunghiul lui Pascal se găsesc combinările de k elemente ale mulțimii {1, 2, ..., i} (k variind de la 0 la i).

Pentru a rezolva problema nu este necesar să reținem într-o matrice triunghiul lui Pascal în întregime, este suficient să reținem la fiecare pas i (i variind de la 1 la n) numai linia curentă și cea precedentă, pe care o vom utiliza pentru determinarea liniei curente.

```
unsigned long LNou[50], LVecchi[50];
unsigned long combinari(unsigned n, unsigned k)
{ LNou[0]=LVecchi[0]=1;
  for (int i = 1; i <= n; i++)
    for (int j=1;j<=i;j++) LNou[j]=LVecchi[j]+LVecchi[j-1];
    for (j=1; j<=i; j++) LVecchi[j]=LNou[j];
  return LNou[k];}
```

$$C_n^k = \frac{n!}{k!(n-k)!}$$

O altă soluție ar putea utiliza definiția:

Dar în acest caz, în urma calculării factorialelor, se vor obține erori datorate depășirilor valorilor maxime ce pot fi memorate într-un tip întreg (pentru $n > 16$ va fi depășită valoarea 2147483647, maximum pentru long int).

b. Varianta recursivă

Se va utiliza formula de recurență:

$$C_n^k = C_{n-1}^k + C_{n-1}^{k-1}$$

```
unsigned long comb(unsigned n, unsigned k)
{return n==k || !k || !n?1:comb(n-1, k)+comb(n-1, k-1);}
```

O altă soluție s-ar putea baza pe formula de recurență:

Dar în acest caz, datorită operațiilor de împărțire, se obține un rezultat real.

c. În varianta iterativă, pentru a calcula C_n^k se calculează C_i^j cu i variind de la 0 la n , și j de la 0 la i , dar o singură dată.

Varianta recursivă este mult mai dezavantajoasă, deoarece se recalculează de mai multe ori aceleși valori. De exemplu, pentru a calcula $\text{comb}(5, 3)$ se apelează funcția $\text{comb}(3, 2)$ de două ori, apoi $\text{comb}(2, 1)$ de 3 ori etc. În plus, la fiecare apel recursiv, se aloca memorie pe stivă pentru valorile parametrilor, rezultatul funcției, adresa de revenire, se copiază valorile parametrilor, se calculează valoarea funcției, revenirea din funcție fiind însotită de eliberarea memoriei alocate pe stivă pentru apelul respectiv. Toate aceste operații necesită un timp suplimentar.

d. Vom reprezenta o submulțime ca un vector S, de k elemente din mulțimea $\{1, 2, \dots, n\}$. Pentru a nu genera de două ori aceeași submulțime, considerăm elementele submulțimii în ordine crescătoare.

Vom descrie un algoritm de tip succesor: inițializăm vectorul S, în care memorăm succesiv submulțimile, cu $\{1, 2, \dots, k\}$, ceea ce „mai mică” submulțime în ordine lexicografică. La fiecare pas, afișăm submulțimea curentă și determinăm submulțimea care urmărează în ordine lexicografică. Pentru aceasta, parcurgem vectorul S, începând cu poziția k , determinând prima componentă a vectorului care poate fi mărită (valoarea maximă care poate fi plasată pe poziția i fiind $n-k+i$). Dacă nu am găsit o astfel de componentă, deducem că $S=\{n-k+1, n-k+2, \dots, n\}$, ultima submulțime în ordine lexicografică, deci generarea s-a terminat. Altfel, incrementăm componenta găsită, reinitializând pozițiile până la k cu cele mai mici valori posibile (pe fiecare poziție plasăm o valoare cu 1 mai mare decât valoarea de pe poziția precedentă).

```
#include <iostream.h>
unsigned n, k, s[50];
void afisare()
{ for (int i=1; i<=k; i++) cout << s[i] << ' ';
  cout << "\n"; }

void generare()
{ for (int i=1; i<=k; s[i]=i, i++);
  for (int gata = 0; !gata; )
    (afisare()); //afisez solutia curenta
```

```
for (i=k; i && s[i]==n-k+i; i--);
if (i)
  { // s[i] poate fi marit
    s[i]++;
    //reinitializam componentele de la i+1 la k
    for (int j=i+1; j <= k; s[j]=s[j-1]+1, j++);
    else gata=1; //am terminat generarea
  }
}

void main()
{cout << "n= "; cin >> n;
cout << "k= "; cin >> k;
generare(); }
```

2. Propoziții cu palindroame

În fișierul text TEXT.IN se găsesc propoziții terminate cu punct, în care cuvintele sunt separate prin spații și/sau marcaje de sfârșit de linie. O propoziție are cel mult 1000 de caractere. Să se afișeze pe ecran numai propozițiile care conțin cuvinte palindromice.

Un cuvânt se numește palindrom dacă citit de la stânga la dreapta sau citit de la dreapta la stânga este același (de exemplu, capac, radar etc.).

Soluție

Această problemă, relativ complexă, devine simplă dacă identificăm și rezolvăm subproblemele din care este constituită. O primă subproblemă pe care o identificăm este să testăm dacă un cuvânt este palindrom. Vom descrie o funcție care primește ca parametru un cuvânt și returnează valoarea 1, dacă acel cuvânt este palindrom și 0 altfel:

```
int palindrom (char *s)
{ int lg=strlen(s);
  for (int i=0, j=lg-1; i<j && s[i]==s[j]; i++, j--);
  return i>=j; } //lungimea cuvantului
```

Utilizând această funcție, putem verifica dacă o propoziție, pe care o vom transmite ca parametru funcției verificare(), conține sau nu palindroame. Pentru aceasta vom extrage succesiv cuvintele din propoziție, până la epuizarea tuturor cuvintelor sau până la întâlnirea unui palindrom. Pentru a extrage cuvintele vom utiliza funcția standard strtok(), cu sirul de separatori " \n" (spațiu și marcaj de sfârșit de linie):

```
int verificare (char *s)
{ s=strtok(s, " \n");
  while (s)
    { if (palindrom(s)) return 1;
      s=strtok(NULL, " \n"); }
  return 0; } //mai sunt cuvinte in propozitie
```

Vom citi succesiv propozițiile din fișierul de intrare, apelând pentru fiecare propoziție funcția verificare(). Datorită faptului că funcția strtok() distrugă șirul asupra căruia se aplică, am apelat funcția verificare() având ca parametru o copie a propoziției citite:

```
char prop[Max], s[Max];
void citire()
{
    ifstream fin("text.in");
    while (!fin.eof())
        if (fin.getline(prop, Max, '.')) //citesc o propozitie
            strcpy(s, prop); //copiez propozitia
        if (verificare(s)) cout << prop << endl;
    fin.close();
}
```

3. Operații cu polinoame

Descrieți funcții care să realizeze principalele operații cu polinoame cu o necunoscută și coeficienți reali, în formă algebrică (adunare, scădere, înmulțire, împărțire, evaluarea unui polinom într-un punct).

Soluție

Polinomul P de grad n are forma algebrică:

$P(X) = p_0 + p_1 \cdot X + p_2 \cdot X^2 + \dots + p_n \cdot X^n$. Pentru a reprezenta un astfel de polinom vom reține gradul și coeficienții, în ordinea crescătoare a gradelor:

```
#define Max 100
struct Polinom
{
    int n;
    float c[Max];
}
```

Evaluarea unui polinom într-un punct x presupune calcularea valorii

$$P(x) = p_0 + p_1 \cdot x + p_2 \cdot x^2 + \dots + p_n \cdot x^n$$

```
float eval (Polinom & P, float x)
{
    float val=0, px=1;
    for (int i=0; i<=P.n; i++, px*=x)
        val+=P.c[i]*px;
    return val;
}
```

Observați că, deși polinomul P nu se modifică în urma evaluării, am preferat să transmit o referință la polinom. Astfel, pe stivă se va copia doar adresa polinomului, nu întreaga structură.

Suma a două polinoame se efectuează adunând coeficienții corespunzători monoamelor de același grad:

```
void suma (Polinom& P, Polinom& Q, Polinom& S)
{
    //S=P+Q
    int gradmin=P.n<Q.n?P.n:Q.n;
    for (int i=0; i<=gradmin; i++)
        S.c[i]=P.c[i]+Q.c[i];
}
```

```
for ( ; i<=P.n; i++) //copiez eventualii coeficienți din P
    S.c[i]=P.c[i];
for ( ; i<=Q.n; i++) //copiez eventualii coeficienți din Q
    S.c[i]=Q.c[i];
//determin gradul polinomului suma
for (S.n=i-1; S.n>0 && !S.c[S.n]; S.n--); }
```

Pentru diferență, vom utiliza funcția suma(). Mai exact, vom înmulții coeficienții polinomului Q cu -1, apoi apelăm funcția suma(), apoi restaurăm coeficienții lui Q:

```
void diferența (Polinom& P, Polinom& Q, Polinom& S)
{
    //S=P-Q
    for (int i=0; i<=Q.n; i++) Q.c[i]=-Q.c[i];
    suma(P, Q, S);
    for (int i=0; i<=Q.n; i++) Q.c[i]=-Q.c[i];
}
```

Pentru a calcula produsul P·Q, vom înmulții fiecare monom din P cu fiecare monom din Q, apoi grupăm termenii după puterile lui X:

$$(p_0 + p_1 \cdot X + p_2 \cdot X^2 + \dots + p_n \cdot X^n) \cdot (q_0 + q_1 \cdot X + q_2 \cdot X^2 + \dots + q_m \cdot X^m) = p_0 \cdot q_0 + (p_1 \cdot q_0 + p_0 \cdot q_1) \cdot X + (p_2 \cdot q_0 + p_1 \cdot q_1 + p_0 \cdot q_2) \cdot X^2 + \dots$$

Prin urmare, coeficienții polinomului produs sunt sume de produse coeficienți unul din P, celălalt din Q. Mai exact, produsul $p_i \cdot q_j$ se va aduna la coeficientul lui X^{i+j} :

```
void produs (Polinom& P, Polinom& Q, Polinom& M)
{
    //M=P*Q
    M.n=P.n+Q.n;
    for (int i=0; i<=M.n; i++) M.c[i]=0;
    for (i=0; i<=P.n; i++)
        for (int j=0; j<=Q.n; j++)
            M.c[i+j] += P.c[i]*Q.c[j];
}
```

Împărțirea polinoamelor P și Q conduce la obținerea unui cât C și a unui rest R. Gradul câtului este egal cu diferența dintre gradul polinomului P și gradul polinomului Q. Coeficienții câtului se obțin împărțind coeficientul curent din P la coeficientul dominant al lui Q. Apoi, înmulțim coeficientul curent al câtului cu polinomul Q și scădem polinomul rezultat din P, începând cu poziția curentă. Coeficienții rămași la sfârșit în P reprezintă coeficienții restului.

```
void impartire(Polinom& P, Polinom& Q, Polinom& C, Polinom& R)
{
    //C este câtul, R este restul împărțirii lui P la Q
    if (P.n<Q.n)
        //câtul este 0, restul este Q
        R.n=Q.n;
        for (int i=0; i<=R.n; i++) R.c[i]=Q.c[i];
        C.n=0; C.c[i]=0;
    else
        C.n=P.n-Q.n; R.n=P.n; //copiez P în R
```

```

int i, ip;
for (i=0; i<=R.n; i++) R.c[i]=P.c[i];
for (i=C.n, ip=R.n; i>=0; i--, ip--)
    {C.c[i]=R.c[ip]/Q.c[Q.n];
     for (int j=Q.n; j>=0; j--)
        R.c[ip-Q.n+j]-=Q.c[j]*C.c[i];
    }
//determin gradul restului
for (; R.n>0 && !R.c[R.n]; R.n--); )
}

```

4. Expresie

Fie o expresie aritmetică fără paranteze, formată numai din caractere numerice (0..9), caracterele alfabetice a, b, c, d, e reprezentând variabile și operatorii aritmetici + și -. Semnificația caracterelor care apar în expresie este cea obișnuită într-o expresie algebrică. Într-o constantă și o variabilă, ca și între două variabile se consideră operația de înmulțire. Astfel, expresia 235aeae+c-12bb are semnificația algebrică $235 \cdot a \cdot e \cdot a \cdot e + c - 12 \cdot b \cdot b$.

De pe prima linie din fișierul de intrare EXPR.IN se citește expresia (maxim 200 caractere), apoi de pe fiecare din liniile care urmărează se citește căte o instrucțiune de atribuire de formă <variabilă>=<întreg>, prin care se atribuie valori pentru variabilele care apar în expresie. Scrieți un program care să afișeze valoarea expresiei date. Dacă informațiile nă sunt suficiente, programul va afișa mesajul NEDEFINITA.

(ONI, Mediaș, 1999, clasa a IX-a)

Exemple

EXPR.IN
235aeae+c-12bb
b=1
c=-7
a=10
e=2

EXPR.OUT
93981

EXPR.IN
abc
a=1
b=2

EXPR.OUT
NEDEFINITA

Soluție

Vom citi expresia dată într-un sir de caractere e. După citirea expresiei, vom reține într-un vector caracteristic uzexpr[5] variabilele care intervin în expresie (uzexpr[i]=1, dacă variabila 'a'+i intervine în expresie și 0 altfel). Vom citi apoi succesiv instrucțiunile de atribuire, reținând în vectorul val[5] valorile variabilelor citite, iar în vectorul caracteristic uzinstr[5] vom reține care variabile au intervenit în instrucțiuni de atribuire.

```

#define Max 201
int uzexp[5], uzinstr[5], val[5];
char e[Max];
void citire ()
{ ifstream f("expr.in");
  f.getline(e,Max);
}

```

```

for (i=0; i<strlen(e); i++)
    if (e[i]>='a' && e[i]<='e')      //variabila
        uzexp[e[i]-'a']=1;
    //citesc instructiunile de atribuire
    char sir[20];
    while (!f.eof())
    {
        f.getline(sir,20);
        val[sir[0]-'a']=atoi(sir+2);
        uzinstr[sir[0]-'a']=1;
    }
    f.close(); }

```

Evaluarea expresiei este posibilă dacă toate variabilele care intervin în expresie intervin și în instrucțiuni de atribuire. Funcția eposibil() testează acest lucru:

```

int eposibil()
{ for (int i=0; i<5; i++)
    if (uzexp[i] && !uzinstr[i]) return 0;
    return 1; }

```

În cazul în care evaluarea este posibilă, vom apela o funcție de evaluare a unei astfel de expresii evalexp(). Funcția va extrage succesiv termenii care intervin în expresie, adunând sau scăzând (după caz) valorile lor:

```

int i=0;

long evalexp ()
{ long valexp=0, t;
char op;
if (e[0]!='-')
    { char el[201]="+";
      strcat(el,e);
      strcpy(e,el);}
while (e[i]=='+' || e[i]=='-')
    { op=e[i]; i++;
      t=extragetermen();
      if (op=='+') valexp+=t;
      else valexp-=t;
    }
return valexp; }

```

Pentru a extrage un termen din expresie am apelat funcția extragetermen(), care returnează valoarea termenului extras. Un termen este un produs de factori:

```

long extragetermen()
{ long valt=1;
  while (e[i] && e[i]!='+' && e[i]!='-') //incepe un factor
    valt*=extragefactor();
  return valt; }

```

Un factor este fie o constantă (constituită dintr-o succesiune de cifre), fie o variabilă. Funcția `extragefactor()` returnează valoarea factorului curent:

```
long extragefactor()
{ long valf;
char s[10];
if (e[i]>='a' && e[i]<='e')      //variabila
    {valf=val[e[i]-'a']; i++;}
else                                //constantă
    { valf=atol(e+i);
      i+=strlen(ltoa(valf, s, 10)); }
return valf; }
```

Observați că funcțiile `extragefactor()` și `evalalex()` incrementează valoarea variabilei globale `i`, cu ajutorul căreia parcurgem sirul de caractere `e`, care reprezintă expresia dată.

În funcția `main()` vom apela funcția `citire()` și vom testa dacă este posibilă evaluarea expresiei, caz în care vom apela funcția de evaluare:

```
void main()
{ citire();
ofstream fout("expr.out");
if (eposibil()) fout<<evalalex();
else           fout<<"NEDEFINITA";
fout.close(); }
```

5. Arie poligon

Se dă un poligon, specificând în ordine coordonatele carteziene ale vârfurilor sale. Să se determine aria poligonului în ipoteza că poligonul este:

- convex;
- oarecare.

Soluție

Vom reprezenta un poligon ca un vector de puncte în plan, în care reținem în ordine coordonatele vârfurilor poligonului.

```
#define Max 202
struct Punct
{float x, y; };
typedef Punct Poligon[Max];
Poligon P;
int n;
```

Funcția `citire()` va citi `n`, numărul de vârfuri, precum și coordonatele vârfurilor. Vom utiliza două componente suplimentare (`P[0]` și `P[n+1]`), pe care le vom inițializa cu `P[n]`, respectiv `P[1]`, pentru a calcula ulterior cu ușurință aria poligonului:

```
void citire()
{ ifstream fin("poligon.in");
fin>>n;
for (int i=1; i<=n; i++)
    fin>>P[i].x>>P[i].y;
fin.close();
P[0]=P[n]; P[n+1]=P[1], }
```

Dacă poligonul este convex, aria se calculează cu ușurință, partionând poligonul în triunghiuri, calculând aria fiecărui triunghi și însumând ariile obținute:

```
float arie_convex (Poligon P, int n)
{ float arie=0;
for (int i=2; i<n; i++)
    arie+=triunghi(P[1], P[i], P[i+1]);
return arie; }
```

Observăm că pentru a determina aria unui poligon convex a fost necesară o funcție care să determine aria unui triunghi, specificat prin vârfurile sale. Pentru a calcula aria unui triunghi, utilizăm formula lui Heron:

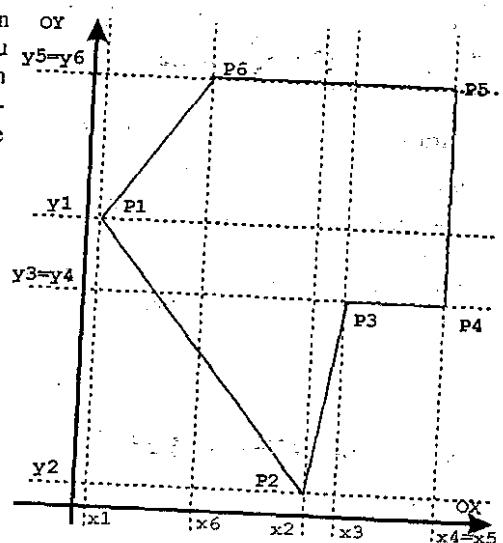
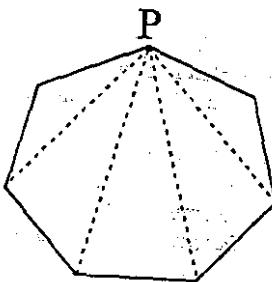
```
float triunghi (Punct a, Punct b, Punct c)
{float p, l1=dist(a,b), l2=dist(b,c), l3=dist(a,c);
p=(l1+l2+l3)/2;
return sqrt(p*(p-l1)*(p-l2)*(p-l3)); }
```

În formula lui Heron intervin lungimile laturilor triunghiului. Pentru a determina lungimea unei laturi, am utilizat funcția `dist()` care determină distanța euclidiană dintre cele două puncte specificate ca parametru:

```
float dist (Punct a,
Punct b)
{return sqrt((a.x-
b.x)*(a.x-b.x)+(a.y-
b.y)*(a.y-b.y));}
```

În cazul în care poligonul nu este în mod obligatoriu convex, calculul ariei se complică.

Pentru a deduce formula de calcul al ariei, vom trasa paralele la Oy prin fiecare vârf al poligonului. Oricare



două paralele consecutive împreună cu latura poligonului corespunzătoare celor două vârfuri prin care sunt trasate și cu axa OX formează un trapez dreptunghic. Calculăm aria trapezului dreptunghic respectiv ca aria cu semn:

$$A_i = h_i \cdot (b_i + B_i) / 2 = (x_{i+1} - x_i) \cdot (y_{i+1} + y_i) / 2$$

Calculând suma arajilor cu semn ale trapezelor dreptunghice, în valoare absolută, obținem aria poligonului convex:

$$A = \sum_{i=1, n} ((x_{i+1} - x_i) \cdot (y_{i+1} + y_i) / 2)$$

Reamintim că $P_{n+1} = P_1$ și $P_0 = P_n$.

Pentru a calcula mai ușor această valoare, vom regrupa termenii, scoțând în factor pe x_i :

$$A = \sum_{i=1, n} x_i \cdot (y_{i+1} - y_{i-1}) / 2$$

```
float arie_oarecare(Polygon P, int n)
{
    float arie=0;
    for (int i=1; i<=n; i++)
        arie+=P[i].x*(P[i+1].y-P[i-1].y);
    return abs(arie/2);
}
```

6. Nasturi

Presupunem că în fiecare pătrat al unei tabele de șah (având n pătrate pe orizontală și n pe verticală, n par, $3 < n < 31$) avem câte un nastur. Știind că urmează să luăm x nasturi de pe tablă, scrieți un program care să precizeze dacă putem face această operație astfel încât pe fiecare linie și pe fiecare coloană să rămână un număr par de nasturi.

(ONI, Mediaș, 1999, clasa a IX-a)

Date de intrare

Fișierul de intrare NASTURI.IN conține pe prima linie n și x .

Date de ieșire

Dacă problema are soluție, fișierul de ieșire NASTURI.OUT conține descrierea tablei. Mai exact, pe linia i ($i=1, 2, \dots, n$) a fișierului va fi descrisă linia i a tablei: pentru un pătrat cu nastur se va scrie litera o iar pentru pătrat liber, caracterul . (punct). Între aceste caractere nu se vor afla spații sau alte caractere. Dacă problema nu are soluție, pe prima linie a fișierului se va scrie textul **fara_solutie**.

Exemplu

NASTURI.IN
4 4

NASTURI.OUT
oo...
oooo
oo...
oooo

Soluție

Observăm că problema nu are soluție dacă $x \geq n^2$, $x=2$ (oricum luăm doi nasturi, rămân cel puțin două linii sau două coloane cu $n-1$, deci număr impar de nasturi) sau $x=n^2-2$ (rămân doar doi nasturi pe tablă).

Dacă $x=4k$ vom forma k grupe de câte 4 pătrate de dimensiune 2×2 (începând din colțul stânga-sus al tablei), de pe care eliminăm cei 4 nasturi. Cum n e par, eliminând în permanență 2 nasturi de pe o linie și 2 de pe o coloană, numărul de nasturi rămâne par.

Pentru $x=4k+2$, cum $x>2$, vom scrie pe x sub formă $x=4k+6$. Vom forma k grupe de dimensiune 2×2 la care procedăm ca în cazul $x=4k$, iar pentru cei 6 nasturi rămași, alegem colțul din dreapta-jos de dimensiune 3×3 din care eliminăm 6 nasturi conform schemei:

ooo		o..
ooo	->	.o.
ooo		..o

Care ne asigură că de pe fiecare linie și fiecare coloană am eliminat câte 2 nasturi, deci numărul de nasturi rămași pe tablă va fi par.

Funcția **citire()** citește datele de intrare și așază toți nasturii pe tablă:

```
#include <iostream.h>
#define Max 100
int n, x;
char tabla[Max][Max];
void citire()
{
    ifstream fin("nasturi.in");
    fin>>n>>x;
    fin.close();
    for (int i=1; i<=n; i++)
        for (int j=1; j<=n; j++)
            tabla[i][j] = 'o';
}
```

Funcția **există_solutie()** returnează 1 dacă problema are soluție sau scrie mesajul corespunzător și returnează 0 dacă problema nu are soluție:

```
ofstream fout("nasturi.out");
int există_solutie()
{
    if (n%2 || n<4)
        (fout<<"fara_solutie"; fout.close(); return 0);
    if (x%2 || x==2 || x>=n*n-2 || x<=2)
        (fout<<"fara_solutie"; fout.close(); return 0);
    return 1;
}
```

În funcția principală, vom citi datele de intrare, vom testa dacă există soluție și, în caz afirmativ, vom determina o soluție și vom afișa tabla:

```
void main()
{
    citire();
}
```

```

if (exista_solutie())
    if (!(x % 4))                                // x=4k
        for (int s=0, m=x/4, i=1; i<n && s<m; i+=2)
            for (int j=1; j<n && s<m; s++, j+=2)
                elimina(i,j);
    else
        { int m=(x-6)/4;                         // x=4k+6
        if (!m) elimina_margine();
        else
            for (int s=0, i=1; i<n && s<m; i+=2)
                {for (int j=1; j<n && s<m; j+=2, s++)
                    elimina(i,j);
                    elimina_margine();}
        }
afisare_tabla(); }

```

Funcția `elimină()` elimină nasturii din pătratul de latură 2, având colțul stânga-sus pe linia 1 și coloana c:

```

void elimina (int l, int c)
{tabla[l][c]='.'; tabla[l+1][c]='.';
 tabla[l+1][c+1]='.'; tabla[l][c+1]='.';}

```

Funcția `elimina_margine()` elimină cei 6 nasturi rămași în colțul dreapta-jos, conform schemei prezentate:

```

void elimina_margine()
{tabla[n][n-2]='.'; tabla[n][n-1]='.';
 tabla[n-1][n-2]='.'; tabla[n-1][n]='.';
 tabla[n-2][n]='.'; tabla[n-2][n-1]='.';}

```

7. Spion

Se cere să se scrie un program care să poată codifica sau decodifica un mesaj. Fiecare linie a mesajului de codificat are cel mult 80 de caractere, acestea fiind litere mari și mici ale alfabetului, caracterele , . : ; ? ! și caracterul spațiu, deci sunt recunoscute caracterele (cu codurile ASCII asociate):

"A"	65	"a"	97	" "	32
"B"	66	"b"	98	"!"	33
.	.	.	,	"	44
.	.	.	.	"."	46
.	.	.	:	" :"	58
"Z"	90	"z"	122	" ; "	59
				" ? "	63

Codificarea se face după următorul algoritm: fiecare linie a mesajului se scrie în ordine inversă, înlocuindu-se fiecare caracter cu sirul rezultat prin transformarea codului ASCII al caracterului respectiv în sir de caractere scris în ordine inversă. Astfel, dacă în mesaj apare caracterul cu codul 123, în mesajul codificat va apărea sirul 321. În mesajul codificat nu apar spații.

Numele fișierelor care conțin mesajul de codificat/decodificat, precum și al fișierului de ieșire care va contine mesajul codificat/decodificat se vor citi de la tastatură. Prin cercetarea mesajului de intrare se stabilește operația care trebuie făcută asupra acestuia (codificare/decodificare).

Exemplu

Dacă fișierul de intrare va conține mesajul (necodificat):

abc

Olimpiada Județeană de Informatică!
fișierul de ieșire va conține:

998979

3379995016117990141111120101137231010012379011791016111010017
114723790017950121190150180197

(pe aceeași linie) și invers.

(Lugoj, 2000, clasele a VII-a – a VIII-a)

Soluție

Vom citi primul caracter din fișierul de intrare, pentru a identifica dacă trebuie să codificăm sau să decodificăm mesajul. Caracterul citit va fi inserat la loc în `stream`-ul de intrare. Dacă am citit o cifră, apelăm funcția de decodificare, altfel apelăm funcția de codificare:

```

ifstream fin("spion.in");
ofstream fout("spion.out");
void main()
{ char c;
fin.get(c); fin.putback(c);
if ('0'<=c && c<='9')
    decodificare();
else
    codificare();
fin.close(); fout.close(); }

```

Pentru codificare, vom citi fișierul de intrare linie cu linie. Fiecare linie o vom inversa (folosind funcția standard `strev()`), apoi o vom parcurge caracter cu caracter. Pentru fiecare caracter de pe linie, vom reține codul său ASCII într-un sir, pe care îl vom inversa și îl vom scrie în fișierul de ieșire:

```

void codificare()
{ char s[81], sircod[4];
int cod;
while (!fin.eof())
    {fin.getline(s, 80);
    strrev(s);
    for (int i=0; s[i]; i++)
        {cod=(int)s[i];
        itoa(cod, sircod, 10);
        }
    }
}

```

```

    strrev(sircod);
    fout<<sircod;
    fout<<endl;
}

```

Pentru decodificare, vom citi de asemenea fișierul de intrare linie cu linie. De data aceasta, vom parcurge linia de la sfârșit către început, extrăgând succesiv codurile caracterelor. Dacă un cod începe cu 1, atunci el este format din trei cifre, altfel numai din două. Codurile inversate vor fi extrase în sirul sircod, care va fi convertit într-un număr întreg (folosind funcția standard atoi()). În fișierul de ieșire va fi afișat caracterul corespunzător codului inversat.

```

void decodificare()
{
    char s[241];
    char sircod[4];
    int cod;
    while (!fin.eof())
    {
        fin.getline(s, 240);
        for (int i=strlen(s)-1; i>0; )
            if (s[i]=='1')           //cod din trei caractere
                {sircod[0]=s[i]; sircod[1]=s[i-1];
                 sircod[2]=s[i-2]; sircod[3]=NULL; i-=3;}
            else                   //cod din două caractere
                {sircod[0]=s[i]; sircod[1]=s[i-1];
                 sircod[2]=NULL; i-=2;}
        cod=atoi(sircod);
        fout<<(char)cod;
        fout<<endl;
    }
}

```

8. Sumă termeni Fibonacci

Considerăm sirul lui *Fibonacci*: 1, 1, 2, 3, 5, 8, 13, 21,... Dat fiind un număr natural ($n \in \mathbb{N}^*$), scrieți acest număr sub formă de sumă de elemente neconsecutive din sirul *Fibonacci*, astfel încât fiecare element să apară cel mult o dată și numărul de termeni ai sumei să fie minim.

(ONI, Constanța, 2000, clasa a IX-a)

Date de intrare

Din fișierul de intrare FIB.IN se citește de pe prima linie numărul natural n . Acesta poate avea maxim 80 de cifre.

Date de ieșire

În fișierul de ieșire FIB.OUT se vor afișa termeni ai sirului *Fibonacci*, câte unul pe linie, a căror sumă este n .

Exemplu

FIB.IN	FIB.OUT
20	2
	5
	13

Soluție

O primă subproblemă pe care o identificăm sunt operațiile cu numere naturale mari. Vom reprezenta un număr natural mare ca un vector în care reținem în ordine cifrele sale, începând cu unitățile. Faptul că indicele din vector al cifrei coincide puterii bazei corespunzătoare cifrei va simplifica mult operațiile cu numere mari astfel reprezentate.

```

#define Lg 81
typedef char Numar[Lg];

```

Nu vom implementa toate operațiile cu numere naturale mari, ci doar pe acelea care sunt necesare. În primul rând avem nevoie de termenii sirului Fibonacci mai mici decât n . Deoarece orice termen din sirul Fibonacci, exceptând primii doi termeni, este egal cu suma celor doi termeni precedenți, pentru a genera termenii sirului Fibonacci avem nevoie de o funcție de adunare a două numere naturale mari:

```

void Adunare(Numar a, Numar b, Numar s)
{
    //s va fi a+b
    for (int t=0, i=0; a[i]&&b[i]; i++)
        { s[i]=(t+a[i]-'0'+b[i]-'0')%10+'0';
          t=(t+a[i]-'0'+b[i]-'0')/10; }
    for (; a[i]; i++) //nu s-au epuizat cifrele lui a
        { s[i]=(t+a[i]-'0')%10+'0';
          t=(t+a[i]-'0')/10; }
    for (; b[i]; i++) //nu s-au epuizat cifrele lui b
        { s[i]=(t+b[i]-'0')%10+'0';
          t=(t+b[i]-'0')/10; }
    if (t)           //retin eventuala cifra de transport
        { s[i]=t+'0'; i++; }
    s[i]=NULL;       //plasez marcajul de sfarsit de sir
}

```

O altă funcție necesară pentru generarea termenilor sirului Fibonacci mai mici sau egali decât n este o funcție de comparare a două numere mari:

```

int Compara(Numar a, Numar b)
/*compara două numere mari, returnând:
1, dacă a>b
0, dacă a==b
-1, dacă a<b */
{
    if (strlen(a) > strlen(b)) return 1;
    if (strlen(b) > strlen(a)) return -1;
    for (int i=strlen(a)-1; i>=0 && a[i]==b[i]; i--);
}

```

```

if (i==1) return 0;
if (a[i]>b[i]) return 1;
return -1; }

```

Vom genera termenii şirului Fibonacci mai mici sau egali decât n apelând funcția Fibonacci(). Termenii generați vor fi memorati în vectorul F, Nr fiind numărul lor:

```

Numar F[Max];
int Nr;
void Fibonacci()
{
    //genereaza termenii sirului Fibonacci mai mici decat n
    strcpy(F[0],"0"); strcpy(F[1],"1");
    for (int i = 1; Compara(F[i], nr) <= 0; i++)
        Adunare(F[i-1], F[i], F[i+1]);
    Nr=i;
}

```

Pentru a-l scrie pe n ca sumă de termeni neconsecutivi din şirul Fibonacci, vom determina cel mai mare termen din şir, mai mic sau egal cu n. Vom scrie acest termen în fișierul de ieșire, îl vom scădea din n, apoi vom relua procedeul pentru noua valoare a lui n, până când n devine 0.

Pentru aceasta ar fi necesară și o funcție de scădere a două numere mari:

```

void Scadere(Numar a, Numar b, Numar c)
{
    //c=a-b; unde a>=b
    for (int t=0, i=0; b[i]; i++)
        if (t+a[i]-b[i]>= 0)
            { c[i]=t+a[i]-b[i]+'0'; t=0; }
        else
            { c[i]=10+t+a[i]-b[i]+!0'; t=-1; }
    for (; a[i]; i++)
        if (t+a[i]-'0'>=0)
            { c[i]=t+a[i]; t=0; }
        else
            { c[i]=10+t+a[i]; t=-1; }
    while (i>1 && c[i-1]=='0') i--;
    c[i]=NULL; }

```

Funcția Determina() are rolul de a descompune pe n ca sumă de termeni Fibonacci neconsecutivi.

```

void Determina()
{
    ofstream fout("fib.out");
    int i=Nr;
    do
    {
        while (Compara(F[i], n)>0) i--;
        fout<<strrev(F[i])<<endl;
        Scadere(n, strrev(F[i]), rez);
        strcpy(n,rez); }

```

```

while (strcmp(n,"0"));
fout.close(); }

```

Rămâne să justificăm că termenii utilizati în sumă sunt neconsecutivi. Să considerăm că $F[i]$ este cel mai mare termen Fibonacci mai mic sau egal cu n . Noua valoare a lui n va fi diferența $n-F[i]$ care este strict mai mică decât $F[i-1]$ (în felic următorul termen din sumă ar fi $F[i-1]$, dar $F[i-1]+F[i]=F[i+1]$, în contradicție cu faptul că $F[i]$ este cel mai mare termen Fibonacci mai mic sau egal cu n). Prin urmare, dacă în sumă intervine termenul $F[i]$, $F[i-1]$ nu poate apărea.

9. Algoritm

Georgel scrie un algoritm care conține structuri de atribuire, alternative, de selecție, repetitive și compuse. După scrierea algoritmului vrea să-l testeze pentru toate cazurile posibile. Pentru aceasta trebuie să cunoască numărul minim de date de test necesare.

Pentru descrierea structurilor se utilizează următoarele cuvinte-cheie:

Instrucțunea	Format	Observații
Atribuire	ATRIB	
Alternativă	DACA ATUNCI ALTFEL	ALTFEL poate lipsi
Selecție	ALEGE CAZ CAZ_IMPLICIT	CAZ_IMPLICIT poate lipsi
Repetitivă condiționată anterior	CAT_TIMP	
Repetitivă condiționată posterior	REPETA PANA_CAND	
Repetitivă de tip for	PENTRU	
Compusă	INCEPUT SFARSIT	

Un algoritm începe cu cuvântul-cheie INCEPUT, se termină cu SFARSIT și nu conține structuri vide. De asemenea, o structură compusă începe cu cuvântul-cheie INCEPUT și se termină cu SFARSIT. După cuvintele-cheie ATUNCI, ALTFEL, CAZ, CAZ_IMPLICIT, CAT_TIMP, REPETA, PENTRU trebuie să existe obligatoriu o structură de atribuire, alternativă, de decizie, repetitivă sau compusă. Orice linie din algoritm începe cu un cuvânt-cheie și are maxim 200 de caractere. Nu se face diferență între literele mari și literele mici.

Să se calculeze numărul minim de date de test necesare verificarea algoritmului.

Date de intrare

Din fișierul text ALGOR.IN se citește algoritmul care conține pe o linie un singur cuvânt-cheie. Nu există linii vide. Algoritmul respectă principiile programării structurate și este scris corect.

Date de ieșire

În fișierul ALGOR.OUT se va scrie pe prima linie numărul minim de date de test necesare pentru verificarea algoritmului. Verificarea se bazează pe principiul „cutiei

"transparente", ceea ce înseamnă că testele se compun astfel încât să fie posibilă executarea algoritmului pe toate ramurile posibile. De exemplu, în cazul unei structuri repetitive CAT_TIMP care conține în corpul său un singur ATRIB, un test vizează o execuție fără să se intre în corpul structurii, altul pentru a trece cel puțin o dată și prin corpul acestuia. În mod similar se tratează și structura PENTRU.

Exemplul 1

ALGOR.IN	ALGOR.OUT
INCEPUT	2
atrib	
DACA	
ATUNCI	
ATRIB	
SFARSIT	

Exemplul 2

ALGOR.IN	ALGOR.OUT	OBSERVAȚIE
INCEPUT	1	REPETA se execută cel puțin o dată
ATRIB		
REPETA		
inceput		
atrib		
atrib		
SFARSIT		
SPARSIT		

Exemplul 5

ALGOR.IN	ALGOR.OUT	OBSERVAȚIE
INCEPUT	4	- se execută ATUNCI și PENTRU - ATUNCI se execută, PENTRU, nu - se execută ALTFEL și PENTRU - ALTFEL se execută, PENTRU, nu în total 4 teste.
atrib		
DACA		
ATUNCI		
ATRIB		
ALTFEL		
ATRIB		
pentru		
atrib		
SFARSIT		

Soluție

Vom descrie o funcție recursivă denumită Instr() care să determine numărul de teste necesare pentru verificarea instrucțiunii curente din algoritm. Instrucțiunea curentă (mai exact, cuvântul-cheie corespunzător) o vom determina prin apelarea funcției cuvant_cheie(). Cuvântul-cheie corespunzător instrucțiunii curente nu va fi transmis ca parametru, ci va fi reținut în variabila globală cuvant.

```
ifstream fin("algor.in");
char cuvant[30];
void cuvant_cheie()
```

```
//extragă cuvantul cheie de pe o linie
char linie[Max];
fin.getline(linie, Max);
strcpy(cuvant, strtok(linie, " ")); }
```

Funcția Instr() determină numărul de teste necesare instrucțiunii curente, în funcție de cuvântul-cheie:

```
long Instr()
/* determină numărul de teste necesare testarea
instrucțiunii curente */
{long nr=1, nrAtunci, nrAltfel, nrCaz, nrTemp;
int contor, implicit;
if (!strcmp(cuvant, "atrib")) //atribuire
    {cuvant_cheie(); nr=1;}
else
    if (!strcmp(cuvant, "daca")) //alternativa
        { // Atunci
            cuvant_cheie(); cuvant_cheie();
            nrAtunci=Instr();
            //Altfel
            nrAltfel=1;
            if (!strcmp(cuvant, "altfel"))
                {cuvant_cheie();
                 nrAltfel=Instr();}
            nr=nrAtunci+nrAltfel; }
    else
        if (!strcmp(cuvant, "alege")) //selectie
            {cuvant_cheie();
             implicit=nr=0;
             while (!strcmp(cuvant, "caz") ||
                     !strcmp(cuvant, "caz_implicit"))
                 {if (!strcmp(cuvant, "caz_implicit"))
                     implicit=1;
                  cuvant_cheie();
                  nrCaz=Instr();
                  nr+=nrCaz; }
             if (!implicit) nr++;}
        else
            if (!strcmp(cuvant, "cat_timp"))
                {cuvant_cheie();
                 nr=Instr()+1; }
    else
        if (!strcmp(cuvant, "repeta"))
            {cuvant_cheie();
             nr=Instr();
             cuvant_cheie(); }
```

```

else
if (!strcmp(cuvant, "pentru"))
{ cuvant_cheie();
nr=Instr()+1;
}
else
if (!strcmp(cuvant, "inceput")) //compusa
{contor=0;
do
{if (!strcmp(cuvant, "inceput"))
{cuvant_cheie();
contor++;}
else
if (!strcmp(cuvant, "sfarsit"))
{cuvant_cheie();
contor--;}
else
{nrTemp=Instr();
nr*=nrTemp;}
while (contor);
}
else nr=1;
}
return nr;
}

```

10. Generare program

Fie a și n două numere întregi ($n < 100$). Să considerăm un limbaj de programare imaginar, care conține instrucțiunea de atribuire și operatorul de înmulțire. Scrieți un program care generează un program scris în acest limbaj pentru calculul lui $b=a^n$, cu număr minim de înmulțiri.

(IOI, 1990, Minsk – Bielorusia)

Exemplu

Pentru $n=13$ programul generat poate fi (acoladele includ comentarii, nu vor fi generate în program):

```

X1:=a;      {=a}
X2:=X1*X1;  {=a^2}
X3:=X2*X2;  {=a^4}
X4:=X3*X1;  {=a^5}
X5:=X3*X3;  {=a^8}
X6:=X5*X4;  {=a^13}
b:=X6;

```

Soluție

Vom utiliza o funcție recursivă de generare a programului, denumită `gen()`. Principiul care stă la baza generării programului cu număr minim de înmulțiri utilizează următoarea definiție recurrentă a puterii unui număr întreg:

$$a^n = \begin{cases} 1, & \text{dacă } n=0 \\ a \cdot a^{n-1}, & \text{dacă } n \text{ impar} \\ a^{n/2} \cdot a^{n/2}, & \text{dacă } n > 0, \text{ par} \end{cases}$$

Funcția va construi printr-un apel recursiv într-o variabilă auxiliară pe a^{n-1} sau pe $a^{n/2}$ (în funcție de paritatea lui n), apoi va calcula pe a^n într-o altă variabilă (funcția va afișa în fișierul de ieșire instrucțiunea de atribuire corespunzătoare). Prin urmare, la fiecare moment trebuie să cunoaștem numărul variabilei auxiliare curente. Pentru aceasta vom utiliza o variabilă globală `nv`. În funcția `main()` vom trata separat cazurile particulare $n=0$ sau $n=1$.

```

#include <iostream.h>
int nv=1;           //numarul variabilei auxiliare
ofstream fout("program.out"); //fisierul de ieșire

void gen(int n)
{
if (n>1)
{ if (n%2)           //n impar
    { gen(n-1);
    fout<<"X"<<nv+1<<":="<<"X1*X"<<nv<<endl; }
  else                  //n par
    { gen(n/2);
    fout<<"X"<<nv+1<<":=X"<<nv<<"*X"<<nv<<endl; }
  nv++; }
}

void main ()
{ cout<<"n="; cin>>n;
if (!n) fout<<"b:=1\n";
else
if (n==1) fout<<"b:=a\n";
else
{ fout<<"X1:=a\n";
gen(n);
fout<<"b:=X"<<nv<<endl; }
fout.close(); }

```

Exercițiu

Descrieți o funcție iterativă care să rezolve aceeași problemă.

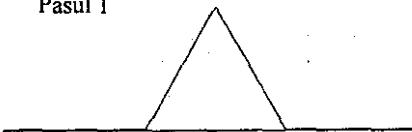
11. Linia de coastă Koch

Linia de coastă Koch este o formă fractală creată în 1904 de către matematicianul suedez Helge von Koch astfel:

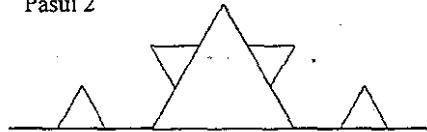
- se pleacă de la un segment de dreaptă;
- pe treimea din mijloc a segmentului de dreaptă se construiește un triunghi echilateral;

- la fiecare pas pe fiecare segment de dreaptă din figura obținută la pasul precedent se desenează pe treimea din mijloc câte un triunghi echilateral.

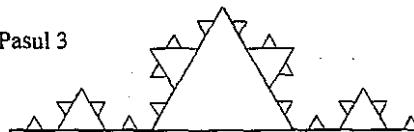
Pasul 1



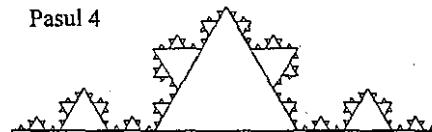
Pasul 2



Pasul 3



Pasul 4



Scripteți un program care citește de la tastatură lungimea segmentului de dreaptă inițial și numărul de iterații și desenează pe ecran în mod grafic o linie de coastă Koch.

(OJI, Iași, 2000, clasele a XI-a – a XII-a)

Soluție

Vom descrie o funcție recursivă denumită `Fractal()`, care va genera o linie de coastă Koch, plecând de la un segment dat. Funcția va primi ca parametru coordonatele extremităților segmentului, precum și numărul pasului curent. În funcția `main()` vom inițializa modul grafic și vom apela funcția `Fractal()`:

```
void main()
{
    cout<<"L= "; cin>>L;
    cout<<"n= "; cin>>n; cin.get();
    Radical = sqrt(3)/2;
    //initializez modul grafic
    int gdriver=DETECT, gmode;
    int Eroare;
    initgraph(&gdriver, &gmode, "c:\\borlandc\\bgi");
    Eroare=graphresult();
    if (Eroare == grOk) //am initializat corect modul grafic
        //stabilesc culoarea de fundal si cea de desenare
        setbkcolor (WHITE);
        setcolor (RED);
        line (0, 200, L, 200);
        Fractal (0, 200, L, 200, 1);
        cin.get(); //lăsă tastatura să se blocheze
    closegraph(); } //inchid modul grafic
```

În funcția `Fractal(x1,y1,x2,y2,Nr)` verificăm în primul rând condiția de terminare. Dacă numărul de pași `Nr` nu depășește `n` (numărul de pași pe care trebuie să îi

efectuăm), determinăm coordonatele punctelor $A(x_a, y_a)$ și $B(x_b, y_b)$ care împart segmentul $[(x_1, y_1), (x_2, y_2)]$ în trei părți egale, apoi construim triunghiul echilateral de latura AB . Apelăm apoi recursiv funcția `Fractal()`, pentru fiecare din cele 4 segmente obținute.

```
void Fractal(int x1,int y1,int x2,int y2,int Nr)
{
    //generez un fractal pornind de la segmentul
    //[(x1,y1),(x2,y2)]
    int xa, ya, xb, yb, xc, yc, xm, ym, Cadran;
    float Latura2, ml;
    if (Nr <= n) //dacă numarul pasului curent nu depăseste n
    {
        xa= (2*x1+x2)/3; ya=(2*y1+y2)/3;
        xb= (x1+2*x2)/3; yb=(y1+2*y2)/3;
        Latura2=Distanta2 (xa, ya, xb, yb);
        if (Latura2 >=2)
            //construiesc triunghiul echilateral cu latura AB
            if (ya == yb)
                { xc=(xa+xb)/2;
                  if (xa<xb) yc=ya-(int)(Latura2*Radical);
                   else yc=ya+(int)(Latura2*Radical); }
            else
                { ml = (yb-ya)/(xb-xa);
                  xm = (xa+xb)/2; ym = (ya+yb)/2;
                  if (ml<0)
                      if (yb>ya) Cadran=1; else Cadran=3;
                  if (ml>0)
                      if (yb<ya) Cadran=2; else Cadran=4;
                  switch (Cadran)
                  { case 1:
                      xc=xm+(int)(sin(M_PI/3)*Latura2*Radical);
                      yc=ym+(int)(cos(M_PI/3)*Latura2*Radical);
                      break;
                    case 2:
                      xc=xm-(int)(sin(M_PI/3)*Latura2*Radical);
                      yc=ym+(int)(cos(M_PI/3)*Latura2*Radical);
                      break;
                    case 3:
                      xc=xm-(int)(sin(M_PI/3)*Latura2*Radical);
                      yc=ym-(int)(cos(M_PI/3)*Latura2*Radical);
                      break;
                    case 4:
                      xc=xm+(int)(sin(M_PI/3)*Latura2*Radical);
                      yc=ym-(int)(cos(M_PI/3)*Latura2*Radical);
                      break;
                  }
                }
        line(xa, ya, xc, yc); line(xb, yb, xc, yc);
    }
}
```

```
//construiesc recursiv fractalii pentru cele 4 segmente
Fractal (x1, y1, xa, ya, Nr+1);
Fractal (xa, ya, xc, yc, Nr+1);
Fractal (xc, yc, xb, yb, Nr+1);
Fractal (xb, yb, x2, y2, Nr+1); }
```

12. Un Excel de jucărie

O foaie de calcul (*spreadsheet*) este un tablou dreptunghiular de celule. Fiecare celulă poate conține date sau expresii, care pot fi evaluate pentru a obține date. Liniile din foaia de calcul sunt numerotate, începând cu 0, iar coloanele se numesc cu litere majuscule ale alfabetului englez (primele 26), apoi folosind combinații de litere (AA, AB etc.). O celulă poate fi referită specificând coloana și linia corespunzătoare (de exemplu, prima celulă este A0).

O foaie de calcul de jucărie conține maxim 26 de coloane (denumite de la A la Z) și maxim 10 linii (numerotate de la 0 la 9). Fiecare celulă poate conține date de tip întreg și expresii aritmetice în care se folosesc doar operatorii binari +, -, * și /, cu semnificația de adunare, scădere, înmulțire, respectiv împărțire întreagă. Operanții pot fi valori întregi sau referințe de celule.

Problema constă în a evalua, dacă este posibil, o foaie de calcul de jucărie dată. Prin evaluarea unei foi de calcul înțelegem înlocuirea expresiilor cu valorile lor.

Date de intrare

Fișierul de intrare EXCEL.IN conține pe prima linie n (numărul de linii din foaia de calcul) și m (numărul de coloane), separate prin spațiu.

Fișierul conține în continuare n*m liniile de date, câte una pentru fiecare celulă. Celulele sunt specificate în ordinea liniilor, iar pe fiecare linie în ordinea coloanelor. Pentru fiecare celulă este specificat conținutul (valoarea sau expresia).

Date de ieșire

Fișierul de ieșire EXCEL.OUT va conține mesajul EVALUARE IMPOSSIBILA sau foia de calcul evaluată (n linii, fiecare linie conținând cele m valori ale celulelor componente, separate prin spațiu).

Restricții și precizări

- Expresiile din foaia de calcul sunt sintactic corecte; pot conține paranteze rotunde și nu conțin spații.
- Prioritatea operatorilor este cea ușuală.
- Valorile absolute ale expresiilor sunt mai mici decât 2.0E+9.
- Expresiile conțin cel mult 100 de caractere.

Exemplul 1

EXCEL.IN	EXCEL.OUT
1 2	EVALUARE IMPOSSIBILA
1+B0	
2+A0	

Exemplul 2

EXCEL.IN	EXCEL.OUT
2 2	1 6
1	5 18
A1+A0	
5	
B0*3	

(ONI, Oradea, 1998, clasa a XI-a)

Soluție

Vom reprezenta o foaie de calcul de jucărie ca o matrice cu 10 linii și 26 de coloane. Fiecare componentă a matricei este o structură ce conține trei câmpuri:

- data: expresia conținută în celulă, citită din fișierul de intrare;
- x: câmp care poate avea doar valorile 0, 1 sau 2, cu semnificația 0 – expresie neevaluată; 1 – expresie în curs de evaluare; 2 – expresie evaluată;
- v: valoarea calculată a expresiei, dacă x=2, sau nedefinit în rest.

```
typedef char Expresie[LgMax];
struct Celula
{
    Expresie data;
    long v;
    int x;};
typedef Celula* Foaie[10][26];
Foaie F;
int n, m, i;
```

Pentru rezolvarea problemei, vom parcurge foaia de calcul în ordinea crescătoare a liniilor, iar pe fiecare linie în ordinea crescătoare a coloanelor. Pentru fiecare celulă neevaluată (câmpul x corespunzător setat pe valoarea 0), vom seta câmpul x pe valoarea 1 (celulă în curs de evaluare) și vom apela o funcție de evaluare a expresiei din celulă. La revenirea din funcția de evaluare setăm câmpul x pe valoarea 2. Dacă pe parcursul evaluării unei expresii apare o referință la o celulă a cărei expresie este de asemenea în curs de evaluare, afișăm mesajul EVALUARE IMPOSSIBILA și întrerupem execuția programului.

```
for (int l=0; l<n; l++)
    for (int c=0; c<m; c++)
        if (!F[l][c].x)
            {F[l][c].x=1;
            i=0;
            F[l][c].v=EvaluareExpresie(F[l][c].data, i);
            F[l][c].x=2;}
```

Evaluarea unei expresii presupune analizarea sa sintactică, în scopul identificării unităților sintactice elementare ce intervin în definirea noțiunii de expresie aritmetică (*termen*, respectiv *factor*) și evaluarea fiecărei unități sintactice. Evaluarea unei expresii implică evaluarea expresiilor corespunzătoare tuturor celulelor referite în cadrul expresiei (practic, o *parcurgere în postordine* a arborescenței cu rădăcina în celula ce conține expresia de evaluat).

```

ofstream fout("excel.out");
long EvaluareExpresie (Expresie e, int &i);
long EvaluareFactor(Expresie e, int &i)
{long f;
if (e[i]=='(') //factorul este o expresie între paranteze
{i++; //trec peste ')'
f=EvaluareExpresie(e,i); //evaluatez expresia
i++; } //trec peste ')'
else //factorul este un operand
if ('A'<=e[i] && e[i]<='Z') //o referinta de celula
{int ll=e[i+1]-'0', cc=e[i]-'A';
if (F[ll][cc].x== 2) f=F[ll][cc].v;
else
if (F[ll][cc].x == 1)
{fout<<"EVALUARE IMPOSSIBILA";
fout.close(); exit(0);}
else
{int ii=0;
f=EvaluareExpresie(F[ll][cc].data, ii);}
i+=2;
}
else //factorul este o valoare întreaga
{int ii=i+1;
while ('0'<=e[ii] && e[ii]<='9') ii++;
char aux[LgMax];
strncpy(aux,e+i, ii-i); aux[ii-i]=NULL;
f=atol(aux);
i=ii; }
return f; }

long EvaluareTermen(Expresie e, int &i)
{ char op;
long f=EvaluareFactor(e,i); //evaluatez primul factor
while (e[i]=='*' || e[i]== '/') //mai urmează factori
{op=e[i];
i++; //trec peste '*'
if (op=='*') f*=EvaluareFactor(e,i);
else f/=EvaluareFactor(e,i); //mai urmează factori
}
return f; }

long EvaluareExpresie(Expresie e, int &i);
{ char op;

```

```

long t=EvaluareTermen(e,i); //evaluatez primul termen
while (e[i]=='+' || e[i]=='-') //mai urmează termeni
{op=e[i];
i++; //trec peste operator
if (op=='+') t+=EvaluareTermen(e,i);
else t-=EvaluareTermen(e,i);}

return t; }

```

Observație

Putem asocia unei foi de calcul de jucărie un digraf astfel:

- fiecare celulă constituie un vârf al digrafului;
- există arc de la un vârf v_1 al digrafului la un alt vârf v_2 dacă și numai dacă în expresia din celula corespunzătoare lui v_1 intervine o referință la celula corespunzătoare lui v_2 . Evaluarea foii de calcul este posibilă dacă și numai dacă *digraful asociat nu conține circuite*.

Probleme propuse

1. Rotire sir

Scrieți o funcție care să realizeze permutarea circulară la stânga a caracterelor dintr-un sir transmis ca parametru. De exemplu, după apelarea funcției, sirul "car" va fi "arc".

2. Cmmdc

- Scrieți o funcție iterativă, bazată pe algoritmul lui *Euclid*, care să returneze cel mai mare divizor comun (*cmmdc*) al celor două numere naturale specificate ca parametru.
- Scrieți o funcție recursivă, bazată de asemenea pe algoritmul lui *Euclid*, de calcul al *cmmdc* al celor două numere naturale specificate ca parametru.
- Fie $n \in \mathbb{N}^*$ și a_1, a_2, \dots, a_n o secvență de n ($1 \leq n \leq 20$) numere naturale, citite de la tastatură. Scrieți un program care să calculeze *cmmdc(a₁, a₂, ..., a_n)*. Programul va utiliza una din funcțiile descrise la punctul a. sau la punctul b.

3. Hermit

Următoarele relații sunt cunoscute sub numele de *polinoame Hermit*:

$$H_n(x) = \begin{cases} 1 & , \text{ pentru } n = 0 \\ 2x & , \text{ pentru } n = 1 \\ 2H_{n-1}(x) - 2(n-1)H_{n-2}(x) & , \text{ pentru } n > 1 \end{cases}$$

Scrieți o funcție recursivă și o funcție iterativă care să calculeze valoarea polinomului $H_n(x)$ pentru valorile lui n și x transmise ca parametru.

4. Raza minimă

Se consideră N ($N \leq 1000$) puncte în plan date prin coordonatele lor carteziene. Să se elaboreze un program care să determine în care dintre aceste puncte putem alege centrul cercului de rază minimă ce conține în interior cele N puncte date, precum și rază cercului.

5. Amnistie

Executând prevederile unei amnistii parțiale, un gardian trebuie să elibereze din cele N ($1 \leq N \leq 2.0E9$) celule ale închisorii cel mult MAX ($1 \leq \text{MAX} \leq N$) deținuți. Celulele sunt așezate în rând, sunt numerotate de la 1 la N și fiecare celulă este ocupată de un singur deținut. Regulamentul de amnistiere trebuie respectat întotdeauna:

- Gardianul trebuie să deschidă toate celulele închisorii.
- La pasul următor, gardianul trebuie să închidă fiecare a doua celulă.
- La pasul 3, gardianul trebuie să ia celulele din 3 în 3, răsucind cheia în broască (cele deschise vor fi închise, iar cele închise deschise). Aceste operații trebuie să se repete, astfel încât la pasul i gardianul va lua celulele din i în i și va răsuci cheia în broasca lor. Numărătoarea trebuie să înceapă întotdeauna din dreptul primei celule.
- Deținuții ale căror celule au rămas deschise după efectuarea tuturor operațiilor vor fi eliberați, dacă numărul lor nu depășește MAX , numărul maxim de amnistieri permise. În caz contrar, procedeul se reia, gardianul luând în considerare la numărătoare în continuare numai celulele deținuților selectați la pasul precedent, până când numărul de deținuți selectați în urma aplicării complete a procedeului nu depășește MAX .

Cunoscând N , numărul de celule din închisoare, și MAX , numărul maxim de amnistieri permise, determinați celulele norocoase.

Exemplu

Pentru $N=20$ și $\text{MAX}=3$, se va afișa: 1 16

(Tabăra Poiana Pinului, 1998)

6. Traseu

Un membru al cercului de orientare turistică de la Clubul Copiilor Lugoj primește harta unei zone din oraș reprezentată prin intersecții și drumuri. Intersecțiile sunt numerotate de la 1 la n . Harta conține, pentru fiecare intersecție, numerele intersecțiilor în care se poate ajunge de la ea mergând în stânga, dreapta sau înainte. De asemenea, el primește un itinerar codificat cu ajutorul caracterelor 's', 'd', 'i' ce simbolizează direcția pe care trebuie să o urmeze din intersecția curentă (stânga, dreapta, înainte).

Scrieți un program care determină intersecția în care ajunge cercetașul, cunoscând intersecția din care pleacă, p.

Date de intrare

Fisierul de intrare TRASEU.IN conține:

TRASEU.IN	Semnificație
$n \ p$	- numărul de intersecții și intersecția de pornire.
$s_1 \ d_1 \ i_1$	- următoarele n linii conțin fiecare câte 3 valori naturale s, d, i separate prin câte un spațiu reprezentând, în ordine, trei intersecții: intersecția în care se ajunge (din intersecția curentă k , linia k) mergând în stânga (s), dreapta (d), înainte (i).
$s_2 \ d_2 \ i_2$	
...	
$s_n \ d_n \ i_n$	
$c_1 c_2 \dots c_k$	- ultima linie conține un sir de caractere ('s', 'd', 'i') ce reprezintă itinerariul.

Date de ieșire

În cazul în care traseul este corect, prima linie a fișierului text TRASEU.OUT va conține un întreg pozitiv reprezentând codul intersecției în care ajunge cercetașul. În cazul în care traseul indicat este incorrect, prima linie va conține valoarea -1, apoi, după un spațiu codul intersecției unde s-a oprit traseul.

Restricții

- $1 < n \leq 1000$
- harta poate conține erori

Exemple

TRASEU.IN	TRASEU.OUT	TRASEU.IN	TRASEU.OUT
5 1	4	6 5	-1 3
4 3 2		1 5 2	
1 2 4		4 3 2	
2 4 4		1 7 4	
1 4 5		1 4 5	
2 4 3		3 4 3	
sdissdi		5 6 4	
		sdissdi	

Timp maxim de execuție: 1 secundă/test.

(Lugoj, 2002, clasa a IX-a)

7. Numere

Se dau două perechi de numere (a, b) și (c, d) . Asupra unei perechi de numere (x, y) se pot face următoarele operații:

1. $(x, y) \rightarrow (x+y, y)$
2. $(x, y) \rightarrow (x, x+y)$
3. $(x, y) \rightarrow (x-y, y)$ numai dacă $x >= y$
4. $(x, y) \rightarrow (x, y-x)$ numai dacă $y >= x$
5. $(x, y) \rightarrow (y, x)$

Se cere ca de la o pereche dată (a, b) să se ajungă la o pereche de asemenea dată (c, d) printr-un număr minim de operații de tipul precizat mai sus.

(OJ, Iași, 1999, clasele a XI-a – a XII-a)

Restricții

- a, b, c, d sunt numere naturale, $1 \leq a, b, c, d \leq 2000000000$

Date de ieșire

Rezultatele se vor afișa în fișierul NUMERE.OUT cu structura:

- | | |
|--|-----------------------------------|
| p | - numărul de operații executate |
| op ₁ op ₂ ...op _p | - operațiile executate, în ordine |

Daca nu există soluție atunci va apărea pe prima linie -1.

Exemplu

Pentru $a=7$, $b=10$, $c=2$, $d=7$, fișierul de ieșire poate conține:

```
8
4 3 3 4 5 2 2 2
```

8. Cod Booth

Reprezentarea Booth a permis accelerarea spectaculoasă a efectuării operațiilor aritmetice în calculator. Codul Booth este o reprezentare a numerelor în care baza este 3, dar cifrele sunt 0, 1 și, în loc de 2, -1, care va fi reprezentat prin '!''. De exemplu, 9 este reprezentat în cod Booth ca 100, 8 ca 10!, iar 14 ca 1!!!.

Stim că pentru orice număr natural reprezentarea sa Booth este unică (rezultatul rămâne valabil și pentru numere întregi și raționale).

- Să se scrie un program, care, citind reprezentarea Booth a unui număr natural, afișează numărul în baza 10.
- Să se scrie un program care, citind un număr natural în baza 10, afișează reprezentarea în cod Booth a numărului.
- Să se scrie un program care, citind reprezentările Booth a două numere naturale, afișează reprezentările Booth ale sumei și produsului celor două numere, fără a efectua trecerea în altă bază.

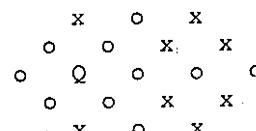
(OJI, Iași, 1996)

9. Hexagon

Toată lumea știe cum se mișcă dama pe o tablă de șah. Să ne imaginăm însă că avem o tablă sub formă unui hexagon. Mai jos se află o astfel de tablă de dimensiune 3, unde pe fiecare latură există câte 3 spații.(notate cu 'x').



Atunci o damă – notată cu Q – va controla câmpurile notate cu 'o':



Pare ceva simplu: aici dama poate controla maxim 3 diagonale (nu se poate acționa pe verticală), spre deosebire de tabla de șah unde pot fi controlate până la 4 diagonale.

Fiind dată o tablă hexagonală, determinați care este numărul minim de dame ce pot fi așezate, astfel încât fiecare câmp al tablei să fie controlat de cel puțin o damă și pe orice diagonală a tablei să se afle cel mult o damă.

SUBPROGRAME. RECURSIVITATE**Date de intrare**

Fișierul de intrare HEXAGON.IN conține pe prima linie n – numărul de câmpuri de pe o latură a tablei ($1 \leq n \leq 40$).

Date de ieșire

În fișierul de ieșire HEXAGON.OUT veți afișa pe prima linie numărul minim de dame care se pot afla pe tablă, astfel încât orice câmp al tablei să fie controlat de cel puțin o damă și pe orice diagonală a tablei să se afle cel mult o damă. Pe a doua linie se află o așezare a damelor, care reprezintă o soluție a problemei:

$p_1 \ q_1 \ p_2 \ q_2 \ \dots \ p_k \ q_k$

unde $p_i \ q_i$ reprezintă poziția damei i pe tabla (al q_i -lea element de pe linia p_i).

Exemplu

HEXAGON.IN

3

1 1 2 3 3 2

HEXAGON.OUT

3

1 1 2 3 3 2

10. Trenuri

Pe o linie de cale ferată sunt amplasate $n+1$ stații, numerotate consecutiv de la 0 la n . Se știe pentru fiecare i ($1 \leq i \leq n$) că distanța de la stația 0 la stația i constituie s_i metri. Evident, $s_{i-1} < s_i$, pentru $2 \leq i \leq n$. Pe această linie circulă două trenuri. Fiecare tren circulă de la stația inițială 0 până la stația 1 cu viteza constantă v_1 m/s, apoi staționează d_1 secunde, merge până la stația 2 cu viteza constantă v_2 m/s, apoi staționează d_2 secunde. Trenul continuă mișcarea astfel, până când ajunge în stația n . Generalizând, fiecare tren merge de la stația $i-1$ la stația i cu viteza constantă v_i m/s, apoi staționează timp de d_i secunde, unde $i=1, 2, \dots, n$.

Primul tren începe mișcarea din stația inițială în momentul de timp 0 și ajunge la stația finală peste T secunde (vezi figura). Trenul al doilea circulă exact la fel, dar începe mișcarea la D secunde după pornirea primului tren.

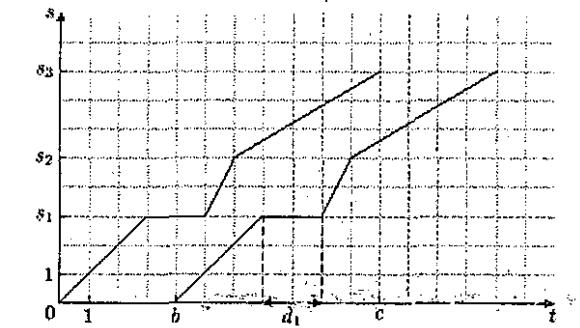
Scrieți un program care să determine D minim, astfel încât în orice moment de timp t ($D < t < T$) distanța dintre trenuri să fie cel puțin x metri.

(Olimpiada Republicană de Informatică, Republica Moldova, 2001, clasele a X-a – a XII-a)

Date de intrare

Fișierul Trenuri.in conține:

$n \ x$
 $v_1 \ s_1 \ d_1$



v2 s2 d2

vn-1 sn-1 dn-1
vn sn

Date de ieșire

Fisierul de ieșire Trenuri.out va conține pe unica linie numărul D, specificat cu două zecimale.

Exemplu

Pentru situația ilustrată în figură:

Trenuri.in
3 2.00
1.00 3.00 2.00
2.00 5.00 0.00
0.60 8.00

Trenuri.out
4.00

Restricții

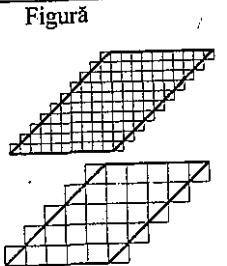
- $1 \leq n, di, xi \leq 1000$, pentru $\forall i \in \{1, 2, \dots, n\}$
- $0.5 \leq si, vi \leq 1000$, pentru $\forall i \in \{1, 2, \dots, n\}$
- timp de execuție 1 secundă/test.

11. Dimensiune fractală

Clement a învățat la școală cum să calculeze dimensiunea fractală a unei figure. El desenează un caroaj de dimensiune 1 prin punctele de geometrice. În acest scop, el desenează pe acest caroaj o figură cu N vârfuri ($2 < N < 1000$, coordonate întregi). Apoi desenează pe acest caroaj o figură cu N vârfuri ($2 < N < 1000$, coordonate întregi). Apoi desenează pe acest caroaj o figură cu N vârfuri ($2 < N < 1000$, coordonate întregi). Apoi desenează pe acest caroaj o figură cu N vârfuri ($2 < N < 1000$, coordonate întregi). Cele N vârfuri definesc un poligon convex și ale cărui laturi nu se intersectează. El trebuie să determine numărul K1 de pătrate, cu latura 1, ale caroajului care se intersectează cu figura. Prin intersecție se înțelege că pătratul are puncte comune cu interiorul figurii. Apoi el șterge din caroaj liniile care trec prin coordonate impare și determină numărul K2 de pătrate din noul caroaj obținut, care intersectează figura. Dimensiunea fractală se calculează printr-o formulă oarecare în funcție de K1 și K2. Scrieți un program care să determine K1 și K2.

(Balcaniada de Informatică, Macedonia, 2000)

Exemplu

FRACTAL.IN	FRACTAL.OUT	Figură
4		
0 0		
10 0		
20 10		
10 10	110 30	

12. Telescaun

Un fermier montan dorește să construiască un telescaun în ferma proprie. În acest scop a prospectat terenul și a identificat o zonă de formă dreptunghiulară de maxim interes turistic. A construit apoi o hartă a zonei, a împărțit-o în pătrătele de latura 1 și a înscris în fiecare pătrătel cota portiunii de teren corespunzătoare. Telescaunul trebuie să circule pe un traseu rectiliniu, care să unească un punct de cotă maximă cu un punct de cotă minimă și, pentru a nu face investiții suplimentare, cotele de-a lungul traseului trebuie să fie în ordine descrescătoare. În cazul în care există mai multe posibilități fermierul va prefera, evident, traseul de lungime minimă.

(ONI, Mediaș, 1999, clasa a XI-a)

Date de intrare

Datele de intrare se citesc din fisierul COTE.IN care are următoarea structură:

n m	– lungimea, respectiv lățimea hărții;
$c_{11} c_{12} \dots c_{1m}$	– cotele.
$c_{21} c_{22} \dots c_{2m}$	
...	
$c_{n1} c_{n2} \dots c_{nm}$	

Date de ieșire

Fisierul de ieșire COTE.OUT va conține mesajul IMPOSSIBIL pe prima linie, dacă nu se poate găsi nici un traseu care să corespundă cerințelor, sau, în cazul în care un asemenea traseu există, va avea structura:

L	– lungimea traseului, cu două zecimale cu rotunjire;
$x_m y_m x_m y_m$	– coordonatele punctului de cotă maximă și coordonatele punctului de cotă minimă.

Exemplu

COTE.IN	COTE.OUT
4 4	3.61
1 2 2 2	3 4 1 1
2 2 3 3	
3 3 3 4	
2 3 2 2	

13. Un dreptunghi

Se consideră o tablă pe care sunt desenate $M \times N$ ($1 \leq N, M \leq 20000$) pătrate. Pe această tablă este plasat la coordonate necunoscute un dreptunghi de dimensiuni de asemenea necunoscute. Pentru aflarea poziției și a dimensiunilor dreptunghiului Alinuța aruncă bile de la marginile tablei în lungul liniilor sau coloanelor. Dacă bila izbește dreptunghiul se întoarce, în caz contrar trece mai departe. Alinuța ne comunică rezultatele aruncărilor, falsificând unele dintre ele. Să se determine poziția și dimensiunile dreptunghiului astfel încât numărul de rezultate falsificate să fie minim. Dacă există mai multe soluții posibile, determinați-o pe cea pentru care aria dreptunghiului este maximă.

(Baraj Mediaș, 1999)

Observații

- Se poate arunca de mai multe ori pe aceeași linie sau coloană (eventual unele rezultate fiind falsificate).
- Dimensiunile dreptunghiului pot fi și 0 (dreptunghiul poate fi un segment sau un punct).
- Un segment oprește doar bilele care vin perpendicular pe el, iar un punct nu oprește bile.

Date de intrare

Fișierul de intrare DREPT.IN conține:

- Pe prima linie sunt scrise două numere naturale M și N, reprezentând numărul de linii, respectiv numărul de coloane ale tablei. Pe cea de a doua linie este scris un număr natural Q ($1 \leq Q \leq 20000$), reprezentând numărul de aruncări pe linii. Pe următoarele Q linii sunt perechi de numere naturale:

L₁ R₁

L₂ R₂

⋮

L_Q R_Q

unde linia L_i R_i are semnificația: aruncarea pe linia L_i are rezultatul R_i (valoarea 0 semnifică faptul că bila a trecut, valoarea 1 faptul că bila s-a întors).

- Următoarea linie conține un număr natural P ($1 \leq P \leq 20000$), reprezentând numărul de aruncări pe coloane. Pe următoarele P linii sunt câte două numere:

C₁ S₁

C₂ S₂

⋮

C_P S_P

unde C_i S_i are semnificația: aruncarea pe coloana C_i are rezultatul S_i (valoarea 0 semnifică faptul că bila a trecut, valoarea 1 faptul că bila s-a întors).

Date de ieșire

Fișierul de ieșire DREPT.OUT va conține patru numere naturale: L C H W

unde L, C reprezintă linia și coloana colțului stânga-jos al dreptunghiului determinat (pătratul 1, 1 este colțul stânga-jos al tablei), iar H, W reprezintă înălțimea și lățimea dreptunghiului.

Exemplu

DREPT.IN

4 5

3

4 0

1 0

3 0

4

2 1

3 0

4 1

5 0

DREPT.OUT

2 1 1 4

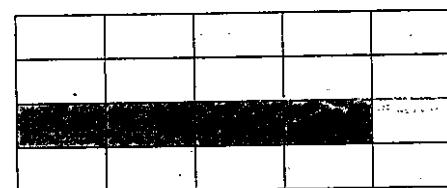


Figura corespunde exemplului:

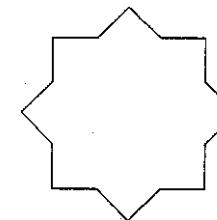
Timp maxim de execuție: 1 secundă/test.

14. Graftali

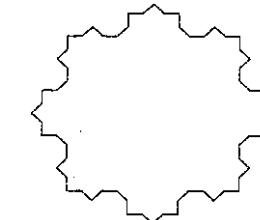
Vom numi *graftali* obiecte geometrice definite recursiv după anumite reguli. În problema noastră, alegem următoarele reguli. Simbolurile elementare sunt:

Simbol	Semnificație
l	– linie
h	– linie ascunsă (invizibilă)
t	– schimbarea sensului de deplasare, în sens trigonometric
a	– schimbarea sensului de deplasare în sens invers trigonometric

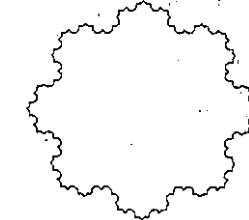
Un graftal este definit printr-o secvență de simboluri elementare. De exemplu, considerând sirul s=laalaalaal și înlocuind apoi în mod recursiv fiecare l din secvența s cu ltlalaaltl, obținem:



Pasul 1



Pasul 2



Pasul 3

În același mod se pot înlocui recursiv simbolurile h, a sau t.

Scrieti un program care citește datele de la tastatură după modelul alăturat și afișează pe ecran graftalul generat.

```
sirul principal (laalaalaal);
sirul lui l (ltlalaal);
sirul lui h (h);
sirul lui t (t);
sirul lui a (a);
coordonate origine :
pt x ( 220);
pt y ( 140);
factori de scalare:
pt x ( 0.60);
pt y ( 0.60);
lungimea liniei ( 4.00);
direcția de plecare ( 0.0);
unghiul de variație ( 45.0);
numarul de pasi (5);
```

Metoda backtracking

Prezentare generală

În variantă elementară metoda *backtracking* se aplică problemelor pentru care soluția se poate reprezenta ca un vector $x = (x_0, x_1, \dots, x_{n-1})$. Fiecare componentă x_i a vectorului poate lua valori într-o anumită mulțime S_i ($i=0, n-1$). Produsul cartezian $S_0 \times S_1 \times \dots \times S_{n-1}$ se numește *spațiul soluțiilor posibile*. Problemele care se rezolvă prin *backtracking* nu impun generarea tuturor soluțiilor posibile (generare exhaustivă), ci doar generarea celor soluții care îndeplinește anumite condiții, specifice problemei, denumite *condiții interne*. Soluțiile posibile care respectă condițiile interne sunt denumite *soluții rezultat*. Unele probleme impun obținerea unei singure soluții rezultat și anume cea care îndeplinește o anumită condiție de optim. Aceasta este denumită *soluție optimă*.

Pentru a evita generarea tuturor soluțiilor posibile, metoda *backtracking* atribuie pe rând valori elementelor vectorului x . Mai exact, componenta x_k primește o valoare numai în cazul în care componentele x_0, x_1, \dots, x_{k-1} au primit deja valori. În acest caz, componentei x_k î se atribuie pe rând acele valori posibile (valori din mulțimea S_k) care îndeplinește condițiile de continuare. *Condițiile de continuare* sunt condiții derive din condițiile interne, care stabilesc dacă pentru o anumită valoare pentru x_k are sau nu sens să continuăm construcția soluției. Spunem că o anumită valoare pentru x_k nu îndeplinește condițiile interne dacă oricum am atribui valori componentelor x_{k+1}, \dots, x_{n-1} nu obținem o soluție rezultat (soluție care să respecte condițiile interne). După atribuirea unei valori posibile care respectă condițiile interne componentei x_k se poate continua construcția soluției în mod recursiv (de data aceasta sunt fixate $k+1$ poziții).

Pentru a descrie formalul general al metodei *backtracking* vom utiliza funcția *BKT()*. Considerăm că n , numărul de componente ale vectorului, și vectorul x sunt variabile globale.

```
void BKT (int k)
//cand apelam functia BKT cu parametrul k presupunem ca
//pozitiile 0,1,...,k-1 din vectorul x sunt fixate
Element MC[]; int nc;
//tipul elementelor din MC depinde de problema concretă
if (k==n) //solutia este completa
    Prelucrare_Solutie();
else //continuam generarea
    {Candidat(MC, nc, k);
/*functia Candidat retine în vectorul MC cele nc elemente din
mulțimea  $S_k$  care respectă condițiile de continuare*/
    for (int i=0; i<nc; i++)
        {x[k]:= MC[i]; //extrag un candidat din MC
         BKT(k+1); //apel recursiv
        }
    }
```

Ideeza căutării cu revenire poate fi generalizată și se poate aplica și problemelor în care căutarea se realizează în plan (într-o matrice) sau în spațiu (într-un tablou tridimensional).

1. Rearanjări

Să considerăm că n persoane ($n \leq 10$) așezate pe un rând de n scaune doresc să își schimbe locurile, astfel încât nici o persoană să nu mai stea pe scaunul pe care era așezată inițial. Scrieți un program care să genereze toate rearanjările posibile.

Soluție

Problema cere generarea tuturor permutărilor de ordin n , care nu au puncte fixe. Vom considera că atât scaunele, cât și persoanele sunt numerotate de la 1 la n . Vom reprezenta o soluție ca un vector p cu n componente având semnificația $p[i] =$ numărul scaunului pe care se va așeza persoana i .

Condiții interne

1. $p[i] \in \{1, 2, \dots, N\}, \forall i \in \{1, 2, \dots, N\};$
2. $p[i] \neq p[j], \forall i, j \in \{1, 2, \dots, N\}, i \neq j;$
3. $p[i] \neq i, \forall i \in \{1, 2, \dots, N\}.$

Pentru a nu asocia un scaun mai multor persoane, vom utiliza un vector Uz , cu n componente, având semnificația $Uz[i] = 0$, dacă scaunul i este liber, respectiv 1 dacă scaunul i este ocupat.

Generarea tuturor rearanjărilor o vom realiza prin metoda *backtracking*:

```
void bkt (int k)
{ if (k>n) afis();
  for (int i=1; i<=n; i++)
    if (!Uz[i] && i!=k)
        {Uz[i]=1; p[k]=i; bkt(k+1); Uz[i]=0; }
```

2. Anagrame distințe

Se citește un sir de maxim 20 de caractere, ce pot fi doar litere mici și spații. Să se genereze toate anagramele distințe ale sirului citit, abstracție făcând de spații.

Soluție

Problema cere generarea permutărilor cu repetiție. Pentru aceasta vom determina pentru fiecare literă din alfabet numărul de apariții în sirul citit. Apoi vom genera toate sirurile de lungime n (unde cu n vom nota numărul total de litere din sirul de intrare), în care numărul de apariții a fiecărei litere din alfabet corespunde cu numărul de apariții ale literei în sirul de intrare.

Reprezentarea informațiilor

1. Vom reține numărul de apariții ale fiecărei litere din alfabet într-un vector $nr[26]$ ($nr[i] =$ numărul de apariții ale celei de a i litere din alfabet, considerând că litera 'a' este numerotată cu 0).

```

int nr[26], nrs[26], n;
void citire()
{
    int i;
    char sir[21];
    cin.getline(sir, 21);
    for (i=0; sir[i]; i++)
        if (sir[i]!=' ') {n++; nr[sir[i]-'a']++;}
}

```

2. Vom memora o soluție într-un vector s[21].
3. Deoarece la fiecare moment al generării trebuie să reținem câte litere de fiecare tip am utilizat în soluția generată, vom reține numărul de apariții în soluție ale fiecărei litere din alfabet într-un vector nrs[26].

```

ofstream fout("anag.out");
char s[21];
void bkt(int k)
{
    if (k==n) fout<<s<<endl;
    else
        for (int i=0; i<26; i++)
            if (nr[i]>nrs[i])
                {nrs[i]++; s[k]=i+'a'; bkt(k+1); nrs[i]--;
    }
}

```

3. Calculatoare și concurență

La ONI Suceava 1996, participă N ($N \leq 100$) elevi numerotăți de la 1 la N . Repartizarea celor N calculatoare, numerotate de la 1 la N , se face după preferințe. Astfel, fiecare elev i , cu i de la 1 la N , va depune în timp util lista celor K_i ($K_i \leq N$) calculatoare preferate. De asemenea, cele N calculatoare au și ele preferințele lor în rândul celor N elevi. Ca urmare, fiecare calculator i , cu i de la 1 la N , va prezenta în timp util o listă a celor P_i ($P_i \leq N$) elevi preferați.

O pereche elev – calculator se numește „în consens” dacă cele două părți se preferă reciproc. Repartizarea elevilor la calculatoare se numește „stabilă” dacă toate perechile sunt „în consens”.

Scrieți un program care pe baza celor două seturi de preferințe să furnizeze o repartizare stabilă.

Date de intrare

Fișierul de intrare Prefer.in conține datele pe $2N+1$ linii cu structura:

N	numărul de elevi (calculatoare)
$C_{i1} C_{i2} \dots C_{iN}$	lista calculatoarelor preferate de elevul 1
\dots	
$C_{N1} C_{N2} \dots C_{NK_N}$	lista calculatoarelor preferate de elevul N
$e_{11} e_{12} \dots e_{1P_1}$	lista elevilor preferați de calculatorul 1
\dots	
$e_{N1} e_{N2} \dots e_{NP_N}$	lista elevilor preferați de calculatorul N

Date de ieșire

Fișierul de ieșire Prefer.out va avea structura:

```

i1 j1
i2 j2
...
iN jN

```

În cazul în care nu există nici o repartizare stabilă, fișierul de ieșire va conține mesajul: Repartizare imposibila.

Exemplu

PREFER.IN	PREFER.OUT
4	1 3
1 2 3 4	2 2
2	3 4
3 4	4 1
1	
3 4	
2 3	
1	
2 3 4	

(ONI, Suceava, 1996)

Soluție

Reprezentarea informațiilor

1. Vom reține preferințele elevilor și ale calculatoarelor într-o matrice p cu N linii și N coloane, având semnificația: $p[i][j]=1$, dacă elevul i și calculatorul j se preferă reciproc, și 0 altfel.
2. Vom reprezenta o soluție într-un vector c cu N componente, având semnificația $c[i]$ este calculatorul repartizat elevului i .
3. Pentru a nu repartiza mai multor elevi același calculator, vom utiliza un vector Uz cu N componente, având semnificația $Uz[i]=1$, dacă am repartizat deja calculatorul, i și 0 altfel.

Condiții interne

1. $c[i] \in \{1, 2, \dots, N\}, \forall i \in \{1, 2, \dots, N\}$;
2. $c[i] \neq c[j], \forall i, j \in \{1, 2, \dots, N\}, i \neq j$;
3. $p[i][c[i]]=1, \forall i \in \{1, 2, \dots, N\}$.

Pentru construirea matricei de preferințe p , vom citi mai întâi preferințele elevilor marcând cu 1 în matrice pozițiile corespunzătoare unei preferințe elev – calculator. Vom citi apoi preferințele calculatoarelor, incrementând componentele matricei corespunzătoare preferințelor citite. În final, preferințele reciproce elev – calculator vor fi marcate cu 2.

```

void citire()
{
    int i,j;
    ifstream fin("prefer.in");
    fin>>n; fin.get();
}

```

```

for (i=1; i<=n; i++)
    { //citesc lista preferintelor elevului i
        while (fin.peek() != '\n')
            {fin>>j;
             p[i][j]=1; }
        fin.get(); }
for (i=1; i<=n; i++)
    { //citesc lista preferintelor calculatorului i
        while (fin.peek() != '\n')
            {fin>>j;
             if (fin.good()) p[j][i]++;
             fin.get(); }
for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
        if (p[i][j]==2) p[i][j]=1;
        else p[i][j]=0;
fin.close(); }

```

Pentru generarea soluțiilor vom utiliza metoda *backtracking*. Prima soluție găsită va fi afișată și generarea se termină.

```

void bkt(int k)
{ if (k>n) {afis(); exit(0);}
  for (int i=1; i<=n; i++)
    if (!Uz[i] && p[i][k])
      {Uz[i]=1; c[k]=i; bkt(k+1); Uz[i]=0;}
}

```

4. Sistemul de criptare Richelieu

În scopul codificării unui text, să se construiască toate şabloanele în formă de matrice de dimensiuni $2k \times 2k$ ($k \leq 10$) care conțin $k \times k$ găuri în diferite poziții în așa fel încât prin rotirea consecutivă de patru ori a şablonului cu 90° , fiecare element al matricei să devină vizibil exact o singură dată.

Exemplu

Pentru $k=2$, există 256 de soluții. De exemplu (1 reprezintă o gaură):

1	..	k	$k+1$..	$2k$
I		II			
..					
k					

1100

0010
0000
0100

Soluție

În primul rând trebuie să determinăm pentru o poziție dată (i,j) care sunt celelalte 3 poziții corespunzătoare (cele peste care să suprapune poziția (i,j) la rotirea eventual repetată a şablonului cu 90°). Să considerăm că poziția (i,j)

METODA BACKTRACKING

se găsește în cadranul I al şablonului. La o rotire cu 90° , poziția (i,j) va corespunde poziției $(j, 2*k-i+1)$ din cadranul II. La o rotire cu 180° , poziția (i,j) va corespunde poziției $(2*k-i+1, 2*k-j+1)$ din cadranul III. La o rotire cu 270° , poziția (i,j) va corespunde poziției $(2*k-j+1, i)$ din cadranul IV. Prin urmare, pentru a genera toate şabloanele corecte distințe, vom reține pentru fiecare din cele $k \times k$ găuri posibile cadranul în care va fi plasată gaura (valoarea 1, 2, 3 sau 4).

```

int k, g[21][21];
void GenSablon(int i, int j)
{
    if (i>k) Afisare();
    else
        for (int v=1; v<=4; v++)
            {g[i][j]=v;
             if (j==k) GenSablon(i+1,1);
              else GenSablon(i,j+1); }
}

```

Când vom afișa o soluție vom plasa gaura (i,j) în poziția corespunzătoare cadranului din care face parte.

```

ofstream f("rich.out");
int s[41][41];

void Afisare()
{int i, j;
for (i=1; i<=k; i++)
    for (j=1; j<=k; j++)
        {s[i][j]=s[j][2*k-i+1]=0;
         s[2*k-i+1][2*k-j+1]=s[2*k-j+1][i]=0;
         switch (g[i][j])
            { case 1: s[i][j]=1; break;
              case 2: s[j][2*k-i+1]=1; break;
              case 3: s[2*k-i+1][2*k-j+1]=1; break;
              case 4: s[2*k-j+1][i]=1; }
        }
f<<"Solutia nr. "<<++nrsol<<endl;
for (i=1; i<=2*k; i++)
    {for (j=1; j<=2*k; j++) f<<s[i][j];
     f<<endl; }
}

```

5. Problema cofetăriei

Tânărul Gigel primește de la mama sa o sumă de bani S , pe care este ferm hotărât să o cheltuiască în întregime la cofetărie. Cunoscând prețul fiecărui sortiment de prăjitură și faptul că Gigel nu vrea să mănânce două prăjitură de același fel, afișați toate variantele în care Gigel își poate cheltui banii.

De exemplu, pentru $S=1000$, $n=5$ sortimente de prăjituri și prețurile $(200, 300, 600, 100, 400)$ programul va afișa:

Solutia nr. 1

1 2 4 5

Solutia nr. 2

2 3 4

Solutia nr. 3

3 5

Soluție

Reprezentarea informațiilor

Vom reține costul fiecărui sortiment de prăjituri într-un vector c , cu n componente, unde n este numărul de sortimente de prăjituri.

O soluție va fi reprezentată printr-un vector f , în care reținem sortimentele de prăjituri cumpărate.

Condiții interne

- $f[i] \in \{1, 2, \dots, n\}$, $\forall i \in \{0, 1, 2, \dots, k-1\}$ (k este numărul prăjiturilor cumpărate);
- $f[i] < f[i+1]$, $\forall i \in \{0, 1, \dots, k-2\}$ (nu contează ordinea în care sunt așezate prăjiturile pe farfurie, deci convenim să le așezăm în ordinea crescătoare a sortimentelor);
- $c[f[0]] + c[f[1]] + \dots + c[f[k-1]] = S$ (suma S este cheltuită în întregime).

Funcția *backtracking* de generare a tuturor soluțiilor este:

```
void Cumpara(int k)
//am cumparat deja prajiturile f[0], f[1], ..., f[k-1]
{if (Sum==S) //am cheltuit intreaga suma?
  Afis(k); //afisez solutia curenta
else
  for (int i=f[k-1]+1; i<=n; i++)
    if (Sum+c[i]<=S) //am bani pentru prajitura i?
      {f[k]=i; //cumpar prajitura i
       Sum+=c[i]; //am cheltuit c[i]
       Cumpara(k+1); //generez solutia in continuare
       Sum-=c[i];} //revin din apelul recursiv
}
```

Exerciții

- Deoarece Gigel nu cumpără două prăjituri de același fel, putem reprezenta o soluție ca un vector f , cu n componente, unde:

$f[i] = \begin{cases} 1, & \text{dacă Gigel cumpără o prăjitură din sortimentul } i \\ 0, & \text{altfel.} \end{cases}$

În acest caz, singura condiție internă este:

$$\sum_{i=1}^n f[i] \cdot c[i] = S$$

Scriți un program care să rezolve problema folosind acest mod de reprezentare a informațiilor.

2. Modificați programul astfel încât să obțineți numărul maxim de prăjituri pe care Gigel le poate cumpăra cu suma S , în ipoteza că nu cumpără două prăjituri de același fel.

3. Modificați programul astfel încât să obțineți toate variantele în care Gigel poate cheltui suma S , în ipoteza că eventual el poate cumpăra mai multe prăjituri de același fel.

6. Labirint

Fie L un labirint codificat printr-o matrice cu n linii și m coloane ($0 < n, m < 40$) cu elemente numere naturale din intervalul $[0, 255]$. Fiecare bit 1 din reprezentarea binară a unui element din labirint corespunde unei direcții posibile de mișcare din poziția respectivă, în următoarea ordine: N, NE, E, SE, S, SV, V, NV, biști fiind numerotată începând cu cel mai puțin semnificativ.

Scriți un program care să determine un traseu posibil de ieșire din labirint, de lungime minimă, a unui șoarece aflat într-o poziție inițială x_0, y_0 dată, dacă un astfel de traseu există.

(OJI, Iași, 1999, clasa a X-a)

Date de intrare

Datele de intrare se citesc din fișierul LABIRINT.IN sub forma:

```
n m
L11 L12 ... L1m
L21 L22 ... L2m
...
Ln1 Ln2 ... Lnm
x0 y0
```

Date de ieșire

Rezultatele vor fi afișate în fișierul de ieșire LABIRINT.OUT sub forma:

```
LgMin
x0 y0
x1 y1
x2 y2
...
xLgMin-1 yLgMin-1
cu semnificația:
```

L – lungimea traseului parcurs de șoarece spre ieșire, exprimată în număr de elemente „traversate”.

$L[x_0, y_0], L[x_1, y_1], \dots, L[x_{LgMin}-1, y_{LgMin}-1]$ – căsuțele din labirint, în ordinea în care sunt traversate de șoarece.

Dacă problema nu are soluție, fișierul de ieșire conține mesajul IMPOSSIBIL.

Exemple

LABIRINT.IN

```

5 6
1 2 3 4 5 6
7 8 9 10 11 12
13 14 15 16 17 18
19 20 21 22 23 24
25 26 27 28 29 30
3 4

```

Soluție

Aceasta este o problemă „clasică” de *backtracking* în plan.

Vom reprezenta labirintul ca o matrice Labirint. Pentru a marca pozițiile din labirint prin care am trecut deja vom utiliza o matrice suplimentară t (matrice martor) în care pozițiile din labirint prin care am trecut deja sunt marcate cu 1, restul componentelor fiind 0. Vom reține drumul curent într-un vector Drum, iar drumul minim în vectorul DrumM. Pozițiile din matrice prin care trece drumul nu le vom reține sub forma *linie, coloană*, ci vom memora numărul de ordine al poziției respective în matrice (considerând matricea liniarizată). Pentru deplasarea pe cele 8 direcții vom utiliza 3 vectori constanți (x – pentru deplasările relative pe linie în cele 8 direcții, y – pentru deplasările relative pe coloane și respectiv Bit – un vector de constante, utilizat pentru a testa, în ordine, biții corespunzători celor 8 direcții de deplasare).

```

#define Max 40
const int x[]={-1,-1,0,1,1,1,0,-1}; //directii de miscare
const int y[]={ 0, 1,1,1,0,-1,-1,-1};
const int Bit[]={1,2,4,8,16,32,64,128}; //verificare biti
int Labirint[Max][Max], t[Max][Max]; //labirint si martor
unsigned Drum[Max*Max], DrumM[Max*Max];
int x0, y0, n, m;
unsigned LgMin, Pas=1;

```

În funcția main() vom citi datele de intrare, vom inițializa drumul cu poziția de plecare, vom genera apelând funcție recursivă Traseu() toate traseele posibile din poziția inițială către marginea labirintului, apoi vom apela funcția de afișare a drumului minim:

```

void main()
{
    Cit_Labirint();
    t[x0][y0]=1; //soarecele porneste de aici
    LgMin=MAXINT; //traseu de lungime maxima
    Drum[0]=(m+1)*x0+y0; //retin coordonatele (x0, y0)
    Pas=1;
    Traseu(x0, y0);
    Afisare();
}

```

Funcția Traseu() are ca parametri coordonatele poziției curente în labirint.

```

void Traseu(int i, int j)
{int ii, jj;

```

LABIRINT.OUT
3
3 4
4 4
5 4

```

for (int k=0; k<8; k++) //incearcă în cele 8 direcții
{ii=i+x[k]; jj=j+y[k]; //linia și coloana nouă
 if (ii>0&&ii<=n&&jj>0&&jj<=m) //daca e în labirint
     if (Labirint[ii][jj]&Bit[k] && !t[ii][jj])
         //e voie în direcția asta și nu am mai fost pe aici
         {t[ii][jj]=Pas;
          Drum[Pas]=ii*(m+1)+jj;
          //daca a ajuns la o margine
          if (ii==1 || ii==n || jj==1 || jj==m)
              VerificTraseuMinim(Pas);
          //verifica dacă drumul gasit e mai scurt
          else
              {Pas++;
               if (Pas<LgMin)
                   //daca nu a depășit drumul minim curent
                   Traseu(ii, jj); //incercă în continuare
               Pas--;} //în caz contrar pentru alt traseu
          t[ii][jj]=Drum[Pas]=0;}
}

```

Funcția VerificTraseuMinim() compară lungimea traseului curent cu lungimea traseului minim, schimbând minimul, dacă este cazul:

```

void VerificTraseuMinim(int Pas)
{
    if (Pas+1<LgMin) //am gasit un traseu mai mic?
        {LgMin=Pas+1; //retine lungime
         for (int k=0; k<=Pas; k++)
             DrumM[k]=Drum[k]; } //si drum
}

```

7. Antitero

Într-o clădire cu h etaje sunt deținuți, la parter, câțiva prizonieri de către T teroriști. Fiecare etaj al clădirii are mxn camere identice. Fiecare cameră are un cod numeric (nu neapărat unic) exprimat printr-un număr natural din intervalul [0,255]. O trupă antitero, formată din K specialiști, trebuie să elibereze prizonierii. Trupa antitero este parașutată pe clădire și încearcă să ajungă la parter. Se cunoaște locul (x, y) unde fiecare membru al trupei a aterizat pe acoperiș. Greutatea fiecărui membru al trupei este un număr natural din intervalul [1,255]. Un membru al trupei poate trece „în jos” printr-o cameră a clădirii doar dacă „greutatea lui trece prin camera respectivă”, conform următoarei definiții.

Spunem că „*a trece prin b*” ($a \gg b$) dacă, în reprezentare binară, numărul de cifre 1 ale lui a este mai mic sau egal cu numărul de cifre 1 ale lui b și cifrele 1 ale lui a sunt comune cu unele cifre 1 ale lui b.

Exemplu „44 trece prin 174” ($44 \gg 174$) deoarece

$$44 = 00101100$$

$$174 = 10101110$$

Pentru detectarea unei camere prin care să poată trece, un membru al trupei antitero se poate, eventual, deplasa cu un „pas” în cele 8 direcții alăturate poziției curente în care a ajuns prin aterizare sau trecerea „în jos”. Prin „pas”-ul respectiv se ajunge la una din cele 8 camere vecine. Prizonierii pot fi eliberați doar dacă la parter ajung minim T membri ai trupei antitero. Să se determine dacă prizonierii pot fi eliberați sau nu, precum și numărul de membri ai trupei antitero care pot să ajungă la parter.

(ONI, Constanța, 2000, clasa a IX-a)

Date de intrare

Fișierul text ANTITERO.IN are structura următoare: pe prima linie valorile m, n, h, K, T despărțite prin câte un spațiu, cu semnificațiile descrise mai sus; următoarele h linii reprezintă codurile celor m × n camere ale unui etaj, despărțite prin câte un spațiu; ultimele K linii ale fișierului conțin greutatea și coordonatele x și y ale poziției de aterizare pe acoperiș ale celor K membri ai trupei antitero, pentru fiecare pe câte o linie, despărțite prin câte un spațiu:

```
m n h K T
C111 C112 ... C11n C121 C122 ... C12n ... C1m1 C1m2 ... C1mn
C211 C212 ... C21n C221 C222 ... C22n ... C2m1 C2m2 ... C2mn
...
Ch11 Ch12 ... Chin Ch21 Ch22 ... Ch2n ... Chm1 Chm2 ... Chmn
G1 x1 y1
G2 x2 y2
...
GK xK yK
```

Date de ieșire

Fișierul text ANTITERO.OUT conține pe prima linie mesajul DA sau NU, iar pe cea de a doua linie numărul de membri ai trupei de comandă ajunși la parter.

Restricții

- $2 \leq m, n, h \leq 30$
- $1 \leq x_i \leq m$
- $1 \leq y_i \leq n$
- $1 \leq T, K, G_i \leq 255$
- $0 \leq c_{ijk} \leq 255$
- Toate valorile sunt întregi

Exemplu

ANTITERO.IN

```
5 5 5 3 2
0 0 0 0 0 0 0 33 0 0 0 0 2 0 0 0 0 0 3 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 44 2 0 0 0 0 0 3 0 0 0 0 0 0
0 0 0 0 0 11 0 0 0 0 0 0 2 22 0 0 0 0 0 3 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 66 2 0 0 0 0 0 7 0 15 0 0 0 0
1 2 2
2 3 3
3 4 4
```

ANTITERO.OUT

```
DA
2
```

Soluție

Vom reprezenta clădirea ca un tablou tridimensional c. Informațiile despre specialistii antitero le vom reține într-un vector Com cu componente de tip struct Tero (o structură în care reținem poziția de pornire de pe acoperiș, greutatea și dacă antiteroristul respectiv a ajuns sau nu la parter). Pentru deplasarea pe cele 9 direcții posibile vom utiliza doi vectori Dx – pentru a reține deplasările relative pe direcția Ox, respectiv Dy – pentru deplasările relative pe Oy.

```
#define DimMax 31
#define MaxK 201
const char Dx[9]={0, 0, 1, 1, 1, 0, -1, -1, -1};
const char Dy[9]={0, -1, -1, 0, 1, 1, 1, 0, -1};
typedef unsigned char Byte;
struct Tero
{
    Byte x, y; //locul de pornire de pe acoperis
    Byte Greu; //greutatea
    Byte Ajuns; }; //initial FALSE pentru toti
Byte c[DimMax][DimMax][DimMax];
Tero Com[MaxK];
int ContorAjunsi;
int m, n, h;
int Kapa, K, T;
```

În funcția main() vom citi datele de intrare, vom considera pe rând fiecare specialist antitero și vom încerca să îl deplasăm de pe nivelul 1, din poziția sa inițială până la parter. Vom contoriza apoi numărul de antiteroriști ajunși la parter și vom afișa rezultatele.

```
void main()
{Citeste_Date();
ofstream f("antitero.out");
for (Kapa=1; Kapa<=K; Kapa++)
    Trece(Kapa, Com[Kapa].x, Com[Kapa].y, 1);
    //plec de pe nivelul 1
for (Kapa=1; Kapa<=K; Kapa++) //contorizez cei ajunsi
    if (Com[Kapa].Ajuns) ContorAjunsi++;
    if (ContorAjunsi >= T) f<<"DA"<<endl;
    else f<<"NU"<<endl;
f<<ContorAjunsi;
f.close(); }
```

Funcția Trece() are 4 parametri: indicele specialistului antitero curent și poziția sa (reprezentată prin coordonatele sale carteziene).

```
void Trece(Byte Kapa, Byte xx, Byte yy, Byte Nivel)
{if (!Com[Kapa].Ajuns)
    if ((Nivel == "h")
        {Com[Kapa].Ajuns=1; //Kapa a ajuns !!! la parter
        return;}
    Byte i, x, y;
```

```

for (i=0; i<9; i++) //incearca in cele 9 directii posibile
    {x=xx+Dx[i]; y=yy+Dy[i]; //noua pozitie
     if (c[x][y][Nivel]&&Com[Kapa].Greu== Com[Kapa].Greu)
        //daca trece prin aceasta camera
        Trece(Kapa,x,y,Nivel+1);}//trece la nivelul urmator
    }
}

```

Acest algoritm de *backtracking* în spațiu suportă optimizări importante. Mai exact, vom utiliza un tablou tridimensional suplimentar *p* în care vom marca, pentru fiecare specialist antitero pozițiile prin care trece. Mai exact, pentru fiecare antiterorist vom inițializa tabloul *p* cu 0, apoi vom plasa 1 pe pozițiile prin care trece antiteroristul. Dacă un antiterorist ajunge în una dintre pozițiile în care a mai fost, înseamnă că nu a ajuns deja la parter, deci trecerea prin acea poziție nu mai trebuie efectuată din nou. Funcția *Terce()* va fi modificată astfel:

```

Byte p[DimMax][DimMax][DimMax];
void Trece(Byte Kapa, Byte xx, Byte yy, Byte Nivel)
{
if (!Com[Kapa].Ajuns)
    if (Nivel == h)
        {Com[Kapa].Ajuns=1; //Kapa a ajuns !!! la parter
         return;}
Byte i, x, y;
for (i=0; i<9; i++) //incearca in cele 9 directii posibile
    {x=xx+Dx[i]; y=yy+Dy[i]; //noua pozitie
     if (c[x][y][Nivel]&&Com[Kapa].Greu==Com[Kapa].Greu)
        //daca trece prin aceasta camera
        if (!p[Nivel+1][x][y])
            //nu am mai trecut prin aceasta pozitie
            {p[Nivel][xx][yy]=1; //marchez pozitia
             Trece(Kapa,x,y,Nivel+1);}
    }
}

```

Probleme propuse

1. Doi bandiți

După un jaf făcut la o bancă, cei doi bandiți urmează să împartă în mod egal sumă de bani. Știind că toți banii sunt în săculeți, iar în fiecare săculeț suma S_i ($i \in \{1, 2, \dots, n\}$, $n \leq 20$), scrieți un program care să-i ajute pe cei doi bandiți să împartă săculeții, astfel încât să obțină fiecare aceeași sumă de bani. Pentru fiecare bandit programul va afișa numărul fiecărui săculeț repartizat, precum și suma din săculețul respectiv. Dacă banii nu se pot împărti în mod egal se va afișa mesajul **DUCEM BANII INAPOI**.

Exemplu

Pentru $n=8$ și săculeții cu valorile 1,3,2,2,4,2,1,3 o soluție este:

Bandit 1:
 sac : 3, 4, 5, 7
 continut: 2, 2, 4, 1
 Bandit 2:
 sac : 1, 2, 6, 8
 continut: 1, 3, 2, 3

2. Excursioniști

Un grup format din n ($n \leq 20$) excursioniști au k corturi, de capacitate c_1, c_2, \dots, c_k ($c_1+c_2+\dots+c_k \geq n$). Determinați toate posibilitățile de a repartiza pe cei n excursioniști în cele k corturi, astfel încât nici un cort să nu rămână gol.

3. Problema icosaedrului

Se consideră un icosaedru (poliedru regulat convex cu 20 de fețe), având fețele numerotate de la 1 la 20. Prin rostogolire completă a icosaedrului înțelegem trecerea succesivă de la o față la alta, adiacentă ei, astfel încât să fie parcuse toate fețele exact o dată. Costul unei rostogoliri complete este dat de $c=1*f(1)+2*f(2)+\dots+20*f(20)$, unde $f(1), f(2), \dots, f(20)$ sunt în ordine fețele care se parcurg, plecând de la fața numerotată 1. Prin fețe adiacente înțelegem:

- două fețe care au o muchie comună;
- două fețe care au cel puțin un punct comun.

Scrieți un program care să determine o rostogolire completă de cost minim.

4. Secvențe binare

Să se genereze o secvență binară de lungime 2^n+n-1 ($n \leq 10$) astfel încât oricare două secvențe de n poziții consecutive să fie diferite.

5. Tura pe tabla de șah

Pe o tablă de șah, de dimensiune $n \times n$, este plasată o tură în poziția (x_0, y_0) . Scrieți un program care să verifice dacă tura poate ajunge în poziția (x_1, y_1) , deplasându-se numai pe poziții libere (pe care nu se află plasată o altă piesă).

6. Pătrate

Se consideră 16 pătrate egale cu laturile colorate în patru culori numerotate de la 1 la 4. Același pătrat poate avea aceeași culoare pe 2 sau mai multe laturi. Să se scrie un program care primește o listă de 16 pătrate și construiește o aranjare a lor sub forma de matrice de 4×4 astfel încât două laturi adiacente au aceeași culoare. Pătratele se vor considera date în ordinea: Nord, Est, Sud, Vest și nu se vor roti.

Exemplu

Pentru datele de intrare:

4	4	1	1
4	3	1	2
2	2	2	3
4	1	3	2
1	1	3	1

```

1 3 1 1
2 4 1 3
3 1 1 4
3 2 4 1
1 3 2 2
1 4 4 3
1 1 2 4
4 3 3 2
2 1 1 3
4 4 3 1
2 4 4 4

```

O soluție este (numerele de mai jos corespund numărului de ordine al pătratului în lista de intrare):

```

2 3 4 9
10 7 8 1
16 12 6 11
13 14 5 15

```

Care corespunde aranjării:

```

4 2 4 3
2 3 2 1 2
1 2 3 4
2 3 4 1 4
2 1 1 1
4 4 1 3 4
4 2 1 4
2 3 1 1 4
3 1 3 3

```

Dacă problema nu are soluție, se va afișa valoarea -1.

7. Cofetari

La un concurs de gastronomie, secțiunea Cofetărie, proba eliminatorie constă în pregătirea rapidă a celei mai ieftine prăjituri. Cofetarii participanți au la dispoziție N tipuri de ingrediente, numerotate distinct cu valori de la 1 la N. Fiecare ingredient este disponibil în cantități nelimitate. Pentru fiecare ingredient este cunoscut prețul pentru 100 grame. Se stie că există ingrediente care nu pot fi combinate în aceeași prăjitură (le vom numi ingrediente incompatibile). Prăjitura trebuie să conțină exact M ingrediente compatibile, în proporții fixe date $p_{r1}, p_{r2}, \dots, p_{rM}$. Mai exact, primul ingredient ales reprezintă $p_{r1}\%$ din greutatea prăjiturii, al doilea ingredient reprezintă $p_{r2}\%$ din greutatea prăjiturii etc.

Scripti un program care să determine cele M ingrediente compatibile, cu care, în proporțiile date $p_{r1}, p_{r2}, \dots, p_{rM}$, să poate fi preparată cea mai ieftină prăjitură.

Date de intrare

Datele de intrare se citesc din fișierul text COFET.IN cu următoarea structură

```

N           // numărul de ingrediente disponibile
p1 p2 ... pN // prețurile pe 100 gr ale fiecărui ingredient
k           // numărul de relații de incompatibilitate
xi yi       // ingredientele xi și yi sunt incompatibile

```

```

x2 y2           // ingredientele x2 și y2 sunt incompatibile
...
xk yk           // ingredientele xk și yk sunt incompatibile
M               // numărul de ingrediente ce trebuie folosite în prăjitură
pr1 pr2 ... prM // proporțiile în care apar cele M ingrediente în prăjitură

```

Date de ieșire

Datele de ieșire se vor scrie în fișierul COFET.OUT cu următoarea structură:

```

Cmin           // costul unui kg din cea mai ieftină prăjitură
c1 c2 ... cM // ingredientele alese

```

Restricții și precizări

- Toate valorile care intervin în enunțul problemei sunt numere naturale nenule.
- Datele de intrare sunt corecte și pentru datele de test există întotdeauna soluție.
- $1 < M < N < 50$
- $0 < p_i < 10000$
- $p_{r1} + p_{r2} + \dots + p_{rM} = 100$
- $x_i, y_i \in \{1, 2, \dots, N\}, x_i \neq y_i, \forall i \in \{1, 2, \dots, k\}$

(OLI, Iași, 2002)

9. Valoare maximă

Fie n un număr natural, $n \leq 20$, și a_1, a_2, \dots, a_n n valori întregi. Să se determine toate permutările de ordin n, $p = (p_1, p_2, \dots, p_n)$, pentru care expresia $a_1 * p_1 + a_2 * p_2 + \dots + a_n * p_n$ are valoare maximă.

10. Numere amuzante

Numărul 123456789 are câteva caracteristici amuzante. Astfel, înmulțit cu 2, 4, 7 sau 8 dă ca rezultat tot un număr de nouă cifre distincte (în afară de 0):

$123456789 * 2 = 24691357$	$123456789 * 4 = 493827156$
$123456789 * 7 = 864197523$	$123456789 * 8 = 987654312$

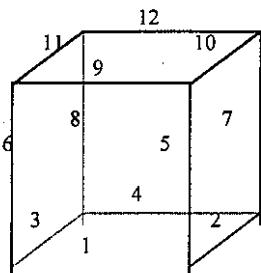
Această proprietate nu funcționează pentru numerele 3, 6 sau 9. Există totuși multe numere de nouă cifre distincte (fără 0) care înmulțite cu 3 au aceeași proprietate. Să se listeze toate aceste numere în care ultima cifră este 9.

(Concurs Tuymaada, 1995)

11. Cubul magic

Să se dispună pe cele 12 muchii ale unui cub toate numerele de la 1 la 12, astfel încât suma numerelor aflate pe muchiile unei fețe să fie aceeași pentru toate fețele.

Considerând următoarea numerotare a muchiilor



ieșirea va fi într-un fișier text având numele CUB.OUT care va conține câte o soluție pe fiecare linie sub forma:

1 m₂ m₃ m₄ m₅ m₆ m₇ m₈ m₉ m₁₀ m₁₁ m₁₂

⋮

1 m₂ m₃ m₄ m₅ m₆ m₇ m₈ m₉ m₁₀ m₁₁ m₁₂

unde m_i este valoarea din {2,...,12} plasată pe muchia i, i ∈ {2,...,12} (muchia 1 conține valoarea 1 în toate soluțiile).

(Tabără de pregătire, Sibiu, 1996)

12. Numere Fibonacci prime

Să se listeze toate numerele prime x (10 ≤ x ≤ 65535) care:

i. sunt numere Fibonacci;

ii. permutând cifrele lui x se obține cel puțin încă un număr prim distinct.

13. Numere superprime

Numim număr *superprim* un număr prim pentru care orice prefix al său este de asemenea număr prim. De exemplu, numărul 719 este superprim deoarece 7, 71, 719 sunt numere prime.

Să se scrie un program care să genereze în ordine crescătoare toate numerele superprime cu N cifre (1 ≤ N ≤ 9).

(OJI, Iași, 2001, clasa a IX-a)

Exemplu

Pentru N=2, programul va genera 23 29 31 37 53 59 71 73 79.

14. Trepte

În Estul îndepărtat, pe vârful unui munte, se va construi un templu. Trepte care vor urca până la acesta trebuie proiectate astfel încât dimensiunile lor să fie diferite, obligând pe cei care se vor urca pe ele să se apropie de zei cu capul plecat. Proiectantul trebuie să determine toate variantele în care aceste trepte se pot construi astfel încât numărul lor să fie cel mai mare posibil.

Fiecare treaptă are o înălțime și o lățime, iar proiectantului îi se cere ca oricare două înălțimi și oricare două lățimi să fie diferite, de asemenea oricare înălțime trebuie să difere de oricare lățime. Nici o înălțime nu poate fi mai mare decât o anumită valoare dată.

Două variante ale proiectului se consideră „identice” dacă în acestea se regăsesc aceleași înălțimi (eventual în altă ordine) și aceleași lățimi (eventual în altă ordine).

Date de intrare

Fișierul de intrare INPUT.TXT conține trei numere întregi I, L și M (2 < I ≤ 50, 2 < L ≤ 50, 1 ≤ M ≤ 12), reprezentând înălțimea totală, respectiv lățimea totală pe care se vor întinde trepte care trebuie proiectate și înălțimea cea mai mare posibilă a unei trepte.

Date de ieșire

Rezultatul se va scrie în fișierul OUTPUT.TXT care va avea următoarea structură:

- pe prima linie se va scrie numărul de trepte determinat N (cel mai mare posibil, dar cel puțin 2);
- pe următoarele linii se vor scrie variante care nu sunt „identice”; pentru fiecare variantă se vor scrie două linii: pe prima se vor scrie toate înălțimile (numere naturale) separate prin căte un blanc, a doua linie va conține toate lățimile (numere naturale) separate prin căte un blanc.

Dacă nu se poate determina nici un proiect care să respecte cerințele puse, în fișierul de ieșire se va scrie mesajul NU.

Exemplu

INPUT.TXT

8
5
7

OUTPUT.TXT

2
1 7
2 3
2 6
1 4
3 5
1 4

15. Semne

Se dă o matrice cu m linii și n coloane (1 ≤ m+n ≤ 15) cu elemente numere întregi. Se numește *operărie* în acest tablou înmulțirea unei linii sau coloane cu -1.

Scrieți un program care să determine cea mai scurtă secvență de operații care aduce matricea la o formă în care toate sumele de elemente pe linii sau coloane sunt numere nenegative. Programul va afișa pe ecran numărul de operații, precum și operațiile efectuate, specificând în ordine numărul liniei sau coloanelor care sunt înmulțite cu -1, precedate de litera L (pentru linii) sau C (pentru coloane).

(PACO, București, 1997, clasa a X-a)

Exemplu

Date de intrare

5	3	
4	-2	2
3	-1	15
-22	0	-3
4	1	-3
5	-3	2

Date de ieșire

2	
C2	13

16. Triangulizare

O triangulizare a unui poligon convex este o mulțime formată din diagonale ale poligonului care nu se intersectează în interiorul poligonului, ci numai în vârfuri și care împart toată suprafața poligonului în triunghiuri.

Fiind dat un poligon cu n vârfuri notate 1, 2, ..., n să se genereze toate triangulizările distincte ale poligonului. Două triangulizări sunt distincte dacă diferă prin cel puțin o diagonală.

Date de intrare

În fișierul text TRIANG.IN se află pe prima linie un singur număr natural reprezentând valoarea lui n ($n \leq 11$).

Date de ieșire

În fișierul text TRIANG.OUT se vor scrie:

- pe prima linie, numărul de triangulizări distincte;
- pe fiecare din următoarele linii câte o triangulizare descrisă prin diagonalele ce o compun. O diagonală va fi precizată prin două numere reprezentând cele două vârfuri care o definesc; cele două numere ce definesc o diagonală se despart prin cel puțin un spațiu, iar între perechile de numere ce reprezintă diagonalele dintr-o triangulizare se va lăsa de asemenea minimum un spațiu.

(OJI, 2002, clasa a X-a)

Exemplu

TRIANG.IN	TRIANG.OUT
5	5
	1 3 1 4
	2 4 2 5
	5 2 5 3
	3 5 3 1
	4 2 1 4

17. Blatistul

În Utopia biletelor de autobuz au o grilă de $N \times N$ pătrătele, dintre care aparatele de taxat compoztează (găuresc) K pătrătele. Biletul poate fi introdus în aparat numai cu un capăt (celălalt capăt fiind prins în cotor), dar poate fi introdus fie pe față, fie pe dos. În acest fel, unele dintre configurațiile posibile de K găuri din grila de $N \times N$ sunt oglindiri stânga-dreapta ale altor configurații. De exemplu, pentru $N=3$ și $K=2$:

```
+++++-----+
| * | | | |
+++++-----+
| | | | * | |
+++++-----+
| | | | | |
+++++-----+
| | | | | |
+++++-----+
(cod: 1123)
```

este oglindirea lui

```
+++++-----+
| | | * | |
+++++-----+
| | | | | |
+++++-----+
| | | | | |
+++++-----+
(cod: 1321)
```

Un blatist face colecție de bilete perforate și dorește să catalogheze toate configurațiile posibile de găuri, ignorând însă oglindirile (deoarece controlorii nu se supără dacă un bilet este compostat pe dos). În acest scop, el codifică fiecare configurație printr-un sir de forma: $l_1, c_1, l_2, c_2, \dots, l_k, c_k$, unde (l_i, c_i) sunt coordonatele găurii i relativ la colțul din stânga-sus al biletului. Găurile sunt enumerate de la stânga la dreapta și de sus în jos. De exemplu, cele două configurații de mai sus sunt codificate prin "1123" și "1321". Acum blatistul este capabil să elimine oglindirile după următorul criteriu: dacă o configurație este oglindirea alteia, dintre cele două se va clasifica numai cea a cărei codificare este prima în ordine lexicografică, cealaltă ignorându-se. Dintre cele două configurații de mai sus, "1123" precedă din punct de vedere lexicografic configurația codificată "1321".

Scrieți un program care să citească valorile N și K ($2 \leq N \leq 8$, $1 \leq K \leq 3$) și care să genereze în fișierul de ieșire BILET.OUT codificările configurațiilor, în ordine lexicografică, către o configurație pe linie.

18. Pioni

Pe o tablă de șah de 8×8 pătrate se află un număr oarecare de pioni, fiecare într-un pătrat separat. Vecinii unui pion se află pe una din direcțiile orizontală, verticală sau diagonală. Scrieți un program care testează dacă pionii aranjați astfel formează o singură figură plină (fără orificii, cu interiorul figurii format numai din pioni) și care să determine numărul pionilor de pe conturul figurii formate, precum și numărul total de pioni cu care s-a format figura. Un pion se consideră că este pe contur dacă numărul vecinilor săi este strict mai mic decât 8.

Exemplu

Pioni.in

```
. . . . P . . .
. . . P . P . .
. . P . . . P .
. . . . P . . .
. . . . P . P .
. . . . . P .
. . . . . .
. . . . . .
```

Pioni.out

```
Figura nu este plina
Contur=10
Total=10
```

19. Numărând forme

Fie două numere n, k ($n > 0, k > 0, n+k \leq 11$). O configurație a unei $n-k$ -forme este un vector cu n elemente alese din domeniul $[-k..k]$, a căror sumă este zero. Două configurații sunt echivalente dacă se pot obține una din alta prin:

- permutări ciclice cu una sau mai multe poziții;
- scrierea inversă a configurației;
- schimbarea semnului tuturor numerelor;
- combinații între a, b și c.

O clasă de echivalență de configurații este numită o $n-k$ -formă.

De exemplu, $(0, 1, 1, -2)$ este o configurație pentru o $4-2$ -formă. Alte configurații echivalente sunt: $(1, -2, 0, 1)$, $(-2, 1, 1, 0)$, $(0, -1, -1, 2)$ și $(-1, -1, 0, 2)$.

Sunt 14 4-2-forme posibile, ordonate lexicografic astfel:
 $(0, 0, 0, 0)$ $(2, -2, 2, -2)$ $(2, 0, 0, -2)$ $(1, -1, 1, -1)$ $(2, -1, 0, -1)$
 $(2, 1, -2, -1)$ $(1, 0, -1, 0)$ $(2, -1, 1, -2)$ $(2, 1, -1, -2)$ $(1, 0, 0, -1)$
 $(2, 0, -2, 0)$ $(2, 2, -2, -2)$ $(1, 1, -1, -1)$ $(2, 0, -1, -1)$

Se cere un program care să calculeze numărul de $n-k$ -forme pentru n și k date.

Exemple

Pentru $n=8$ și $k=0$, programul va afișa 1.

Pentru $n=4$ și $k=2$, programul va afișa 14.

(ACM, 1993)

20. Ferma

Un fermier are un teren care are forma unui tablou dreptunghiular lung de M unități și lat de N unități ($1 \leq N, M \leq 50$). Pe teren sunt plantați din loc în loc copaci, pe care fermierul nu dorește să-i tăie. Dorind să-și supravegheze cultura, fermierul realizează un mic robot de formă pătrată având latura de 3 unități pe care îl poate teleghida prin fermă, parcurgând unitate cu unitate o anumită suprafață.

Robotul se poate mișca pe verticală și pe orizontală, dar nu poate trece peste copaci, nu îl poate distruge, nu se poate rota și are nevoie pentru mișcare de o suprafață corespunzătoare dimensiunii lui.

Scrieți un program care să determine suprafața maximă pe care fermierul o poate urmări, folosind acest sistem.

(ONI, Bacău, 2001, clasa a IX-a)

Date de intrare

Fișierul de intrare FERMA.IN conține:

$N\ M$
 $C_{11}C_{12}\dots C_{1M}$
 $C_{21}C_{22}\dots C_{2M}$
 \dots
 $C_{N1}C_{N2}\dots C_{NM}$

Ultimele N linii codifică ferma; fiecare linie conține câte M caractere (fără să fie separate prin spații) având semnificația: '.' – teren liber; '+' – locul în care este plantat un copac; 'R' – centrul robotului.

Date de ieșire

Fișierul de ieșire FERMA.OUT va conține:

$C_{11}C_{12}\dots C_{1M}$
 $C_{21}C_{22}\dots C_{2M}$
 \dots
 $C_{N1}C_{N2}\dots C_{NM}$

Modul în care fermierul poate să utilizeze robotul pe terenul său este codificat pe N linii, fiecare linie conține câte M caractere (fără să fie separate prin spații) având semnificația: '.' – teren neacoperit de robot; '*' – teren ce poate fi verificat de robot; '+' – loc în care a rămas copacul.

Metoda programării dinamice

Prezentare generală

Programarea dinamică este o metodă de elaborare a algoritmilor care se aplică în general problemelor pentru care se cere determinarea unui optim în urma adoptării unor decizii.

Nu există un criteriu pe baza căruia să identificăm cu siguranță o problemă pentru rezolvarea căreia trebuie să utilizăm metoda programării dinamice, dar putem formula două proprietăți care sugerează o soluție prin programare dinamică.

- *Substructură optimă*

Problema dată poate fi descompusă în subprobleme și soluția optimă a problemei depinde de soluțiile optime ale subproblemelor sale.

Acest criteriu nu indică neapărat o soluție prin programare dinamică, ar putea fi și un indiciu că se poate aplica metoda Greedy sau metoda „Divide et Impera”.

- *Subprobleme superpozabile*

Subproblemele problemei date nu sunt independente, ci se suprapun.

Datorită faptului că subproblemele problemei date se suprapun, deducem că o abordare prin metoda „Divide et Impera” ar fi dezastroasă din punctul de vedere al timpului de execuție (datorită faptului că problemele se suprapun se ajunge la rezolvarea repetată a aceleiași subprobleme). Prin urmare, vom rezolva subproblemele o singură dată, reținând rezultatele într-o structură de date suplimentară (de obicei un tablou):

Rezolvarea unei probleme prin programare dinamică presupune următorii pași:

1. Se identifică subproblemele problemei date.
2. Se alege o structură de date suplimentară, capabilă să rețină soluțiile subproblemelor.
3. Se caracterizează substructura optimă a problemei printr-o relație de recurență.
4. Pentru a determina soluția optimă, se rezolvă relația de recurență în mod *bottom-up* (se rezolvă subproblemele în ordinea crescătoare a dimensiunii lor).

În cele ce urmează vom exemplifică pas cu pas modul de rezolvare a problemelor prin metoda programării dinamice.

1. Înmulțirea optimă a matricelor

Fie n matrice A_1, A_2, \dots, A_n , de dimensiuni $d_0 \times d_1, d_1 \times d_2, \dots, d_{n-1} \times d_n$. Produsul $A_1 \times A_2 \times \dots \times A_n$ se poate calcula în diverse moduri, aplicând asociativitatea operației de înmulțire a matricelor. Numim înmulțire elementară înmulțirea a două elemente. În funcție de modul de parantezare diferă numărul de înmulțiri elementare necesare pentru calculul produsului $A_1 \times A_2 \times \dots \times A_n$. Determinați o parantezare optimă a produsului $A_1 \times A_2 \times \dots \times A_n$ (costul parantezării, adică numărul total de înmulțiri elementare să fie minim).

Exemplu

Pentru $n=3$ matrice cu dimensiunile $(10,1000)$, $(1000,10)$ și $(10,100)$, produsul $A_1 \times A_2 \times A_3$ se poate calcula în două moduri:

1. $(A_1 \times A_2) \times A_3$, necesitând $1000000 + 10000 = 1010000$ înmulțiri elementare;
2. $A_1 \times (A_2 \times A_3)$, necesitând $1000000 + 1000000 = 2000000$ înmulțiri.

Reamintim că numărul de înmulțiri elementare necesare pentru a înmulți o matrice A cu n linii și m coloane și B o matrice cu m linii și p coloane este $n \cdot m \cdot p$ (vezi problema 8 de la capitolul „Structuri elementare de date”).

Soluție

1. Pentru a calcula $A_1 \times A_2 \times \dots \times A_n$, în final trebuie să înmulțim două matrice, deci vom paranteza produsul astfel: $(A_1 \times A_2 \times \dots \times A_k) \times (A_{k+1} \times \dots \times A_n)$. Această observație se aplică și produselor dintre paranteze. Prin urmare, subproblemele problemei inițiale constau în determinarea parantezării optimale a produselor de matrice de formă $A_i \times A_{i+1} \times \dots \times A_j$, $1 \leq i \leq j \leq n$. Observăm că subproblemele nu sunt independente. De exemplu, calcularea produsului $A_i \times A_{i+1} \times \dots \times A_j$ și calcularea produsului $A_{i+1} \times A_{i+2} \times \dots \times A_{j+1}$ au ca subproblemă comună calcularea produsului $A_{i+1} \times \dots \times A_j$.

2. Pentru a reține soluțiile subproblemelor, vom utiliza o matrice M , cu n linii și n coloane, cu semnificația:

$M[i][j] =$ numărul minim de înmulțiri elementare necesare pentru a calcula produsul $A_i \times A_{i+1} \times \dots \times A_j$, $1 \leq i \leq j \leq n$.

Evident, numărul minim de înmulțiri necesare pentru a calcula $A_1 \times A_2 \times \dots \times A_n$ este $M[1][n]$.

3. Pentru ca parantezarea să fie optimă, parantezarea produselor $A_1 \times A_2 \times \dots \times A_k$ și $A_{k+1} \times \dots \times A_n$ trebuie să fie de asemenea optimă. Prin urmare elementele matricei M trebuie să satisfacă următoarea relație de recurență:

$$M[i][i] = 0, \forall i \in \{1, 2, \dots, n\}.$$

$$M[i][j] = \min \{ M[i][k] + M[k+1][j] + d[i-1] \cdot d[k] \cdot d[j] \mid i \leq k < j \}$$

Cum interpretăm această relație de recurență? Pentru a determina numărul minim de înmulțiri elementare pentru calculul produsului $A_i \times A_{i+1} \times \dots \times A_j$, fixăm poziția de parantezare k în toate modurile posibile (între i și $j-1$) și alegem varianta care ne conduce la minim. Pentru o poziție k fixată, costul parantezării este egal cu numărul de înmulțiri elementare necesare pentru calculul produsului $A_i \times A_{i+1} \times \dots \times A_k$, la care se adaugă numărul de înmulțiri elementare necesare pentru calculul produsului $A_{k+1} \times \dots \times A_j$ și costul înmulțirii celor două matrice rezultate ($d_{i-1} \cdot d_k \cdot d_j$).

Observăm că numai jumătatea de deasupra diagonalei principale din M este utilizată. Pentru a construi soluția optimă este utilă și reținerea indicelui k , pentru care se obține minimul. Nu vom considera un alt tabel, ci-l vom reține pe poziția simetrică față de diagonala principală ($M[j][i]$).

4. Rezolvarea recursivă a relației de recurență de mai sus este neficientă, datorită faptului că subproblemele se suprapun, deci o abordare recursivă ar conduce la rezolvarea aceleiași subprobleme de mai multe ori. Prin urmare vom rezolva relația de recurență în mod *bottom-up*: (determinăm parantezarea optimă a produselor de două matrici, apoi de 3 matrice, 4 matrice etc.).

```
long M[NMax][NMax];
void dinamic()
{
    int nr, i, j, k, kmin;
    long min, Infinit=1000000000;
    for (nr=2; nr<=n; nr++) //nr=cate matrice se inmultesc
        for (i=1; i<=n-nr+1; i++)
            {j=i+nr-1;
             //se inmultesc nr matrice, de la Ai la Aj
             for (k=i, min=Infinit; k<j; k++)
                 //determin minimul si pozitia sa
                 if (min>M[i][k]+M[k+1][j]+d[i-1]*d[k]*d[j])
                     {min=M[i][k]+M[k+1][j]+d[i-1]*d[k]*d[j];
                      kmin=k;}
             M[i][j]=min; M[j][i]=kmin; }
}
```

Reconstituirea soluției optime se face foarte ușor în mod recursiv, utilizând informațiile reținute sub diagonala principală în matricea M :

```
void afisare(int i, int j)
{//afiseaza parantezarea optimala a produsului Aix...xAj
if (i==M[j][i]) cout<<"A"<<i;
else {cout<<"("; afisare(i, M[j][i]); cout<<")";}
cout<<"x";
if (j==M[j][i]+1) cout<<"A"<<j;
else {cout<<"("; afisare(M[j][i]+1, j); cout<<");";}}
```

2. Subșir crescător maximal

Fie un șir $A = (a_1, a_2, \dots, a_n)$. Numim *subșir* al șirului A o succesiune de elemente din A , în ordinea în care acestea apar în A : $a_{i_1}, a_{i_2}, \dots, a_{i_k}$, unde $1 \leq i_1 < i_2 < \dots < i_k \leq n$. Determinați un subșir crescător al șirului A , de lungime maximă.

Exemplu

Pentru $A = (8, 3, 6, 50, 10, 8, 100, 30, 60, 40, 80)$ o soluție poate fi:

$$(3, 6, \quad 10, \quad 30, 60, \quad 80).$$

Soluție

1. Fie $A_{i_1} = (a_{i_1} \leq a_{i_2} \leq \dots \leq a_{i_k})$ cel mai lung subșir crescător al șirului A . Observăm că el coincide cu cel mai lung subșir crescător al șirului $(a_{i_1}, a_{i_1+1}, \dots, a_n)$. Evident $A_{i_2} = (a_{i_2} \leq a_{i_3} \leq \dots \leq a_{i_k})$ este cel mai lung subșir crescător al lui $(a_{i_2}, a_{i_2+1}, \dots, a_n)$ etc. Prin urmare, o subproblemă a problemei inițiale constă în determinarea celui mai lung subșir crescător care începe cu a_i , $\forall i \in \{1, \dots, n\}$. Subproblemele nu sunt independente: pentru a determina cel mai lung subșir crescător care începe cu a_i , este necesar să determinăm cele mai lungi subșiruri crescătoare care încep cu a_j , $a_i \leq a_j$, $\forall j \in \{i+1, \dots, n\}$.

2. Pentru a reține soluțiile subproblemelor vom considera doi vectori suplimentari l și poz , fiecare cu câte n componente, având semnificația:
- $l[i]$ = lungimea celui mai lung subșir crescător care începe cu $a[i]$;
- $poz[i]$ = poziția elementului care urmează după $a[i]$ în cel mai lung subșir crescător care începe cu $a[i]$, dacă un astfel de element există, sau -1 dacă un astfel de element nu există.
3. Relația de recurență care caracterizează substructura optimală a problemei este:
- $$l[n]=1; \quad poz[n]=-1;$$
- $$l[i]=\max\{l[j] \mid a[i] \leq a[j], j=i+1, n\}$$
- $$poz[i]=\text{indicele } j \text{ pentru care se obține maximul } l[i].$$
4. Rezolvăm relația de recurență în mod *bottom-up*:

```
int i, j;
l[n]=1; poz[n]=-1;
for (i=n-1; i>0; i--)
    for (l[i]=1, poz[i]=-1, j=i+1; j<=n; j++)
        if (a[i] <= a[j] && l[i]<l[j])
            {l[i]=l[j]; poz[i]=j;}
```

Pentru a determina soluția optimă a problemei, determinăm maximul din vectorul l , apoi afișăm soluția, începând cu poziția maximului și utilizând informațiile memorate în vectorul poz :

```
//determin maximul din vectorul l
int max=l[1], pozmax=1;
for (int i=2; i<=n; i++)
    if (max<=l[i]) {max=l[i]; pozmax=i;}
cout<<"Lungimea celui mai lung subșir crescător: " <<max;
cout<<"\nCel mai lung subșir:\n";
for (i=pozmax; i!= -1; i=poz[i])
    cout<<a[i]<<" ";
```

3. Sumă în triunghi

Să considerăm un triunghi format din n linii ($1 \leq n \leq 100$), fiecare linie conținând numere întregi din domeniul $[1, 99]$, ca în exemplul următor:

			7		
	3	8			
8	1	0			
2	7	4	4		
4	5	2	6	5	

Problema constă în scrierea unui program care să determine cea mai mare sumă de numere aflate pe un drum între numărul de pe prima linie și un număr de pe ultima linie. Fiecare număr din acest drum este situat sub precedentul, la stânga sau la dreapta acestuia.
(IOI, Suedia, 1994)

Soluție

1. Vom reține triunghiul într-o matrice pătratică T , de ordin n , sub diagonala principală. Subproblemele problemei date constau în determinarea sumei maxime care se poate obține din numere aflate pe un drum între numărul $T[i][j]$, până la un număr de pe ultima linie, fiecare număr din acest drum fiind situat sub precedentul, la stânga sau la dreapta sa. Evident, subproblemele nu sunt independente: pentru a calcula suma maximă a numerelor de pe un drum de la $T[i][j]$ la ultima linie, trebuie să calculăm suma maximă a numerelor de pe un drum de la $T[i+1][j]$ la ultima linie și suma maximă a numerelor de pe un drum de la $T[i+1][j+1]$ la ultima linie.
 2. Pentru a reține soluțiile subproblemelor, vom utiliza o matrice suplimentară S , pătratică de ordin n , cu semnificația
- $S[i][j]$ = suma maximă ce se poate obține pe un drum de la $T[i][j]$ la un element de pe ultima linie, respectând condițiile problemei.
- Evident, soluția problemei va fi $S[1][1]$.
3. Relația de recurență care caracterizează substructura optimală a problemei este:
- $$S[n][i]=T[n][i], \forall i \in \{1, 2, \dots, n\}$$
- $$S[i][j]=T[i][j]+\max\{S[i+1][j], S[i+1][j+1]\}$$
4. Rezolvăm relația de recurență în mod *bottom-up*:

```
int i, j;
for (i=1; i<=n; i++) S[n][i]=T[n][i];
for (i=n-1; i>0; i--)
    for (j=1; j<=i; j++)
        {S[i][j]=T[i][j]+S[i+1][j];
         if (S[i+1][j]<S[i+1][j+1])
             S[i][j]=T[i][j]+S[i+1][j+1]);}
```

Exercițiu

Afișați și drumul în triunghi pentru care se obține soluția optimă.

4. Subșir comun maximal

Fie $X=(x_1, x_2, \dots, x_n)$ și $Y=(y_1, y_2, \dots, y_m)$ două siruri de n , respectiv m numere întregi. Determinați un subșir comun de lungime maximă.

Exemplu

Pentru $X=(2, 5, 5, 6, 2, 8, 4, 0, 1, 3, 5, 8)$ și $Y=(6, 2, 5, 6, 5, 5, 4, 3, 5, 8)$ o soluție posibilă este: $Z=(2, 5, 5, 4, 3, 5, 8)$

Soluție

1. Notăm cu $X_k=(x_1, x_2, \dots, x_k)$ (prefixul lui X de lungime k) și cu $Y_h=(y_1, y_2, \dots, y_h)$ prefixul lui Y de lungime h . O subproblemă a problemei date constă în determinarea celui mai lung subșir comun al lui X_k , Y_h . Notăm cu $LCS(X_k, Y_h)$ lungimea celui mai lung subșir comun al lui X_k , Y_h . Utilizând aceste notări, problema cere determinarea $LCS(X_n, Y_m)$, precum și un astfel de subșir.

Observație

1. Dacă $X_k=Y_h$, atunci $LCS(X_k, Y_h)=1+LCS(X_{k-1}, Y_{h-1})$.
 2. Dacă $X_k \neq Y_h$, atunci $LCS(X_k, Y_h)=\max(LCS(X_{k-1}, Y_h), LCS(X_k, Y_{h-1}))$.
- Din observația precedentă deducem că subproblemele problemei date nu sunt independente și că problema are substructură optimală.
2. Pentru a reține soluțiile subproblemelor vom utiliza o matrice cu $n+1$ linii și $m+1$ coloane, denumită `lcs`. Linia și coloana 0 sunt utilizate pentru inițializare cu 0, iar elementul `lcs[k][h]` va fi lungimea celui mai lung subșir comun al sirurilor X_k și Y_h .
 3. Vom caracteriza substructura optimală a problemei prin următoarea relație de recurență:

```

lcs[k][0]=lcs[0][h]=0, ∀k∈{1,2,...,n}, ∀h∈{1,2,...,m}
lcs[k][h]= $\begin{cases} 1+lcs[k-1][h-1], & \text{dacă } x[k]=y[h] \\ \max\{lcs[k][h-1], lcs[k-1][h]\}, & \text{dacă } x[k]\neq y[h] \end{cases}$ 

```

4. Rezolvăm relația de recurență în mod *bottom-up*:

```

for (int k=1; k<=n; k++)
    for (int h=1; h<=m; h++)
        if (x[k]==y[h])
            lcs[k][h]=1+lcs[k-1][h-1];
        else
            if (lcs[k-1][h]>lcs[k][h-1])
                lcs[k][h]=lcs[k-1][h];
            else
                lcs[k][h]=lcs[k][h-1];

```

Deoarece nu am utilizat o structură de date suplimentară cu ajutorul căreia să memorăm soluția optimă, vom reconstituî soluția optimă pe baza rezultatelor memorate în matricea `lcs`. Prin reconstituire vom obține soluția în ordine inversă, din acest motiv vom memora soluția într-un vector, pe care îl vom afișa de la sfârșit către început:

```

cout<<"Lungimea subșirului comun maximal: " <<lcs[n][m];
int d[100];
cout<<"\nCel mai lung subșir comun este: ";
for (int i=0, k=n, h=m; lcs[k][h]; )
    if (x[k]==y[h])
        {d[i++]=x[k]; k--; h--;}
    else
        if (lcs[k][h]==lcs[k-1][h])
            k--;
        else
            h--;
for (k=i-1; k>=0; k--) cout<<d[k]<<' ';

```

5. Stivă de jetoane

Un joc este constituit dintr-o stivă de n ($n \leq 1000$) jetoane, de două culori. Jetoanele sunt numerotate de la 1 la n , jetonul cu numărul 1 fiind cel de la vârful stivei. Cei doi jucători (să-i numim Ana și Barbu) mută alternativ. La o mutare, un jucător poate lua din stivă oricăte jetoane (cel puțin unul), cu condiția ca toate jetoanele luate să fie de aceeași

cupoare. Căștigă jucătorul care ia ultimul jeton. Să presupunem că întotdeauna Ana face prima mutare.

Scrieți un program care să determine dacă Ana are strategie sigură de căștig și dacă da, să afișeze pe ecran mutările Anei (câte jetoane ia din stivă atunci când îi vine rândul). Programul va citi de la tastatură mutările lui Barbu.

Soluție

Vom reține culorile jetoanelor într-un vector `c[]`. Inițializarea configurației de joc constă din citirea numărului de jetoane și generarea aleatoare a culorilor acestora, codificând prima culoare cu 0 și cea de a doua culoare cu 1.

```

#define NMax 1002
int n, c[NMax];
void Init()
{
    cout<<"n="; cin>>n;
    randomize();
    for (int i=1; i<=n; i++) c[i]=random(2);
    cout<<"Stiva de jetoane este ";
    for (i=1; i<=n; i++) cout<<c[i];
    cout<<endl; }

```

1. Subproblemele problemei date constau în determinarea existenței unei strategii sigure de căștig pentru jucătorul care face prima mutare pentru o stivă constituată din jetoanele $i..n$.
2. Pentru a reține soluțiile subproblemelor, vom utiliza un vector `S[]`, având următoarea semnificație: $S[i]=1$, dacă jucătorul care extrage jetonul i are strategie sigură de căștig și 0 altfel. Evident, dacă $S[1]=1$, deducem că Ana are strategie sigură de căștig.
3. Caracterizăm substructura optimală a soluției prin următoarea relație de recurență: $S[n]=1$; Dacă $c[i]\neq c[i+1]$, atunci $S[i]=1-S[i+1]$. Dacă $c[i]=c[i+1]$, atunci $S[i]=1$ (în cazul în care $S[i+1]=1$, jucătorul va lua și jetonul i pe lângă jetoanele care îi asigură căștigul pentru $i+1..n$; dacă $S[i+1]=0$, atunci jucătorul va lua numai jetonul i).
4. Rezolvăm această relație de recurență în mod *bottom-up*:

```

int S[NMax];
void Dinamic()
{S[n]=1;
 for (int i=n-1; i>0; i--)
    if (c[i+1]!=c[i]) S[i]=1-S[i+1];
    else S[i]=1; }

```

În cazul în care Ana are strategie sigură de căștig, pentru a vizualiza și mutările care conduc la căștigarea jocului, utilizăm informațiile deja memorate în `S[]`:

```

if (!S[1])
    {cout<<"Ana nu are strategie sigura de castig!";
     return;}
cout<<"Ana are strategie sigura de castig! Sa jucam!\n";

```

```

for (int i=1; i<=n; )
    for (int k=i; k<=n && S[k]; k++)
        //Ana ia jetoanele de la i la k
        cout<<"Ana ia: "<<k-i<<" jetoane\n";
        if (k>n)
            {cout<<"Ana a castigat!\n"; return;}
    //citim mutarea lui Barbu
    cout<<"Stiva ";
    for (int j=k; j<=n; j++) cout<<c[j];
    cout<<"\nBarbu muta: "; cin>>nr;
    //validez mutarea lui Barbu
    for (j=k+1; j<k+nr; j++)
        if (c[j]!=c[j-1])
            {cout<<"Mutare gresita! Barbu pierde!\n";
             return;}
    i=k+nr;
}

```

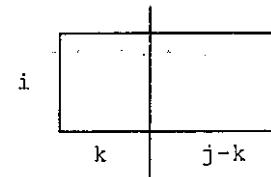
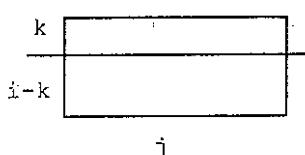
6. Număr minim de pătrate

Fie un dreptunghi de dimensiuni $N \times M$ ($1 \leq N, M \leq 100$). Dreptunghiul poate fi descompus în pătrate cu laturile paralele cu laturile dreptunghiului dat, prin efectuarea unor tăieturi complete. O tăietură este considerată completă dacă este executată de la o latură până la latura opusă a dreptunghiului care se tăie.

Să se determine numărul minim de pătrate în care poate fi descompus dreptunghiul dat.
(OJI, Arad, 2001, clasele a XI-a - a XII-a)

Soluție

- Subproblemele problemei date constau în determinarea numărului minim de pătrate în care poate fi descompus un dreptunghi de dimensiuni $i \times j$ ($\forall i \in \{1, 2, \dots, N\}$, $\forall j \in \{1, 2, \dots, M\}$).
- Vom reține soluțiile subproblemelor într-o matrice Nr , cu N linii și M coloane, având semnificația $Nr[i][j] =$ numărul minim de pătrate în care poate fi descompus un dreptunghi de dimensiuni $i \times j$. Evident, soluția problemei date va fi memorată în $Nr[N][M]$.
- Caracterizăm substructura optimă prin următoarea relație de recurență:

$$\begin{aligned}
 Nr[i][j] &= 1 \quad \forall i=j \\
 Nr[i][j] &= \min \left(\begin{array}{l} \min(Nr[k][j]+Nr[i-k][j]), \text{ //tăiere orizontală în poziția } k \\ \quad k=1, i-1 \\ \min(Nr[i][k]+Nr[i][j-k]) \end{array} \right) \quad \text{//tăiere verticală în poziția } k \\ &\quad k=1, j-1
 \end{aligned}$$


4. Rezolvăm relația de recurență în mod bottom-up:

```

Nr[1][1]=1;
for (i=1; i<=n; i++)
    for (j=1; j<=m; j++)
        if (i==j) Nr[i][j]=1;
        else
            {Nr[i][j]=i*j;
             for (k=1; k<i; k++) //tăietura orizontală
                 if (Nr[i][j]>Nr[k][j]+Nr[i-k][j])
                     Nr[i][j]=Nr[k][j]+Nr[i-k][j];
             for (k=1; k<j; k++) //tăietura verticală
                 if (Nr[i][j]>Nr[i][k]+Nr[i][j-k])
                     Nr[i][j]=Nr[i][k]+Nr[i][j-k];
            }
}

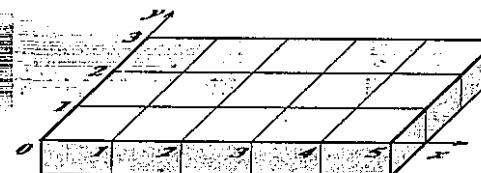
```

Exercițiu

Putem îmbunătăți substanțial acest algoritm observând că $Nr[i][j]=Nr[i/d][j/d]$, unde cu d am notat cmmdc(i, j). Demonstrați corectitudinea acestei observații și utilizați-o pentru a optimiza algoritmul!

7. Laser

Se consideră o placă dreptunghulară cu dimensiunile $m \times n$. Această placă trebuie tăiată în $m \cdot n$ bucăți mai mici, fiecare bucătă fiind un pătrat cu dimensiunile 1×1 . Întrucât placă este neomogenă, pentru fiecare bucătă se indică densitatea d_{xy} , unde x, y sunt coordonatele colțului stânga-jos al pătratului respectiv.



Pentru operațiile de tăiere se folosește un strung cu laser. Fiecare operație de tăiere include:

- fixarea unei plăci pe masa de tăiere;
- stabilirea puterii laserului în funcție de densitatea materialului de tăiat;
- o singură deplasare a laserului de-a lungul oricărei drepte paralele cu una din axele de coordinate;
- scoaterea celor două plăci de pe masa de tăiere.

Costul unei operații de tăiere se determină după formula $c=d_{max}$, unde d_{max} este densitatea maximă a bucătilor 1×1 peste marginile cărora trece raza laserului. Evident, costul total T poate fi determinat adunând costurile individuale c ale tuturor operațiilor de tăiere necesare pentru obținerea bucătilor 1×1 .

Scriți un program care calculează costul minim T .

(Olimpiada Republicană de Informatică, Republica Moldova, 2001)

Date de intrare

Fișierul text LASER.IN conține pe prima linie numerele m și n separate prin spațiu. Următoarele m linii conțin câte n numere d_{xy} separate prin spațiu.

Date de ieșire

Fișierul text LASER.OUT conține pe o singură linie numărul natural T .

Exemplu

LASER.IN
3 5
1 1 1 1 5
1 7 1 1 1
1 1 1 6 1

LASER.OUT
52

Restriții

- $m, n \in \mathbb{N}, 2 \leq m, n \leq 20$
- $d_{xy} \in \mathbb{N}, 1 \leq d_{xy} \leq 100$

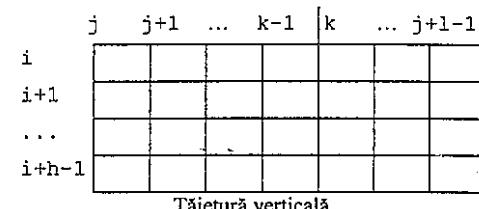
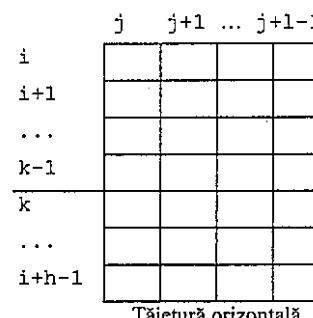
Soluție

1. Subproblemele problemei date constau în tăierea unui dreptunghi din placă dată, care are colțul stânga-sus în poziția (i, j) , are lungimea l și înălțimea h .
2. Pentru a reține soluțiile subproblemelor trebuie să utilizăm un tablou cu 4 dimensiuni $C_{n \times m \times n \times m}$, cu semnificația $C[i, j, h, l] =$ costul total de tăiere a unui dreptunghi din placă dată, care are colțul stânga-sus în poziția (i, j) , are lungimea l și înălțimea h . Evident, soluția problemei date va fi $C[1, 1, n, m]$.

Deoarece nu avem spațiu suficient, vom aloca parțial tabloul c în mod dinamic și îl vom inițializa cu 0:

```
int i, j, k, l, h;
for (i=0; i<=n; i++)
    for (j=0; j<=m; j++)
        for (h=0; h<=n; h++)
            for (l=0; l<=m; l++)
                c[i][j][h][l]=0;
```

3. Pentru a tăia un dreptunghi din placă dată, care are colțul stânga-sus în poziția (i, j) , are lungimea l și înălțimea h trebuie să executăm o primă tăietură completă. Aceasta poate fi trasată fie orizontal (între liniile $k-1$ și k , unde k variază de la $i+1$ până la $i+h-1$), fie vertical (între coloanele $k-1$ și k , unde k variază de la $j+1$ până la $j+l-1$). Vom alege succesiv poziția de tăiere k în toate modurile posibile și vom reține varianta care asigură optimul.



Tăietură verticală

Tăietură orizontală

Prin urmare putem caracteriza substructura optimală a problemei prin următoarea relație de recurență:

```
C[i][j][1][1]=0; ∀i∈{1, 2, ..., n}, ∀j∈{1, 2, ..., m}
C[i][j][h][1]=min
{min{c[i][j][k-i][1]+c[k][j][h-k+i][1]+dmaxL(k, j, 1)} //orizontal
 k=i+1, i+h-1
min{c[i][j][h][k-j]+c[i][k][h][l-k+j]+dmaxC(k, i, h)}} //vertical
k=j+1, j+l-1
```

În relațiile de recurență precedente am utilizat valorile:

- $dmaxL(k, j, 1)$, reprezentând densitatea maximă a bucătilor 1×1 peste marginile căroră trece raza laserului, la o tăietură între liniile $k-1$ și k , de lungime 1, începând de la coloana j ;
 - $dmaxC(k, i, h)$, reprezentând densitatea maximă a bucătilor 1×1 peste marginile căroră trece raza laserului, la o tăietură între coloanele $k-1$ și k , de lungime h , începând cu linia i .
4. Calculul soluției optime în mod *bottom-up*:

```
for (h=1; h<=n; h++)
    for (l=1; l<=m; l++)
        for (i=1; i<=n-h+1; i++)
            for (j=1; j<=m-l+1; j++)
                if (l>1 || h>1)
                    //determin c[i][j][h][1]
                    c[i][j][h][1]=infinity;
                    //tăieturi orizontale
                    for (k=i+1; k<i+h; k++)
                        {a=c[i][j][k-i][1]+c[k][j][h-k+i][1]+dmaxL(k, j, 1);
                         if (c[i][j][h][1]>a) c[i][j][h][1]=a;}
                    //tăieturi verticale
                    for (k=j+1; k<j+l; k++)
                        {a=c[i][j][h][k-j]+c[i][k][h][l-k+j]+dmaxC(k, i, h);
                         if (c[i][j][h][1]>a) c[i][j][h][1]=a;}
                }
```

Observație

O optimizare posibilă a algoritmului constă în evitarea apelurilor funcțiilor `dmaxL()` și `dmaxC()`, prin precalcularea valorilor respective o singură dată și memorarea lor în câte un tablou tridimensional. Implementați această optimizare!

8. Problema patronului

Un patron de gogoșerie a cumpărat un calculator și dorește să învețe să lucreze pe el. Pentru aceasta va umple un raft de cărți din colecția „Informatica în lecții de 9 minute și 60 secunde”. Raftul are lungimea L cm. Seria dispune de n titluri, numerotate 1, 2, ..., n având respectiv grosimile de g_1, g_2, \dots, g_n cm.

Scrieți un program care să selecteze titlurile pe care le va cumpăra patronul, astfel încât raftul să fie umplut complet (suma grosimilor cărților cumpărate să fie egală cu lungimea raftului) și numărul cărților achiziționate să fie maxim. Programul va afișa numărul cărților selectate, precum și grosimile acestora. Dacă nu există soluție, va fi afișat mesajul IMPOZIBIL.

(ONI, Suceava, 1996, clasa a XI-a)

Restricții

- $n \in \mathbb{N}$, $1 \leq n \leq 60$
- $L \in \mathbb{N}$, $1 \leq L \leq 200$
- $g_i \in \mathbb{N}^*$, $g_i \leq 30000$, $\forall i \in \{1, 2, \dots, n\}$
- Timp maxim de execuție: 1 secundă/test.

Soluție

1. Problema constă în selectarea unor elemente a căror sumă să fie egală cu L . În cazul în care există mai multe soluții, este preferată soluția cu număr maxim de elemente. O subproblemă a acestei probleme constă în selectarea unor elemente a căror sumă să fie egală cu S , $S \leq L$.
2. Pentru a reține soluțiile subproblemelor vom utiliza un vector `Nr`, cu $L+1$ componente, având semnificația: $\text{Nr}[S] =$ numărul maxim de titluri ce pot fi cumpărate pentru a umple un raft de lungime S ($0 \leq S \leq L$) sau -1 (dacă nu putem umple un raft de lungime S). Pentru a reține și cărțile cumpărate, vom utiliza o matrice `Uz` cu $L+1$ linii și n coloane, având semnificația $\text{Uz}[S][i]$ este 1, dacă am utilizat titlul i pentru a umple optimal raftul de lungime S și 0, în caz contrar.

```
#define NMax 61
#define LMax 201
int n, L; //numarul de titluri si lungimea raftului
int g[NMax]; //grosimile
int Nr[LMax]; //numarul maxim de carti selectate
int Uz[LMax][NMax]; //indicii titlurilor utilizate
```

3. Să considerăm că ultimul titlu selectat pentru a umple un raft de lungime S este i . Prin urmare, în rest sunt selectate (obligatoriu în mod optim) cărți a căror grosime totală

trebuie să fie egală cu $S-g[i]$. Evident, pentru a putea alege titlul i , condiția este ca acest titlu să nu fie deja utilizat pentru umplerea restului raftului (de lungime $S-g[i]$). Alegerea titlului trebuie să fie făcută, de asemenea, în mod optim (să maximizeze numărul total de cărți selectate):

$\text{Nr}[0]=0;$

$\text{Nr}[S]=\max\{-1, \max_{i=1,n} \{1+\text{Nr}[S-g[i]] | S-g[i] \geq 0 \text{ și } \text{Nr}[S-g[i]] \neq -1\} \text{ și } \text{Uz}[S-g[i]][i]=0\}$

4. Rezolvarea relației de recurență:

```
int S, k, i;
for (S=1; S<=L; S++)
    for (Nr[S]==-1, i=1; i<=n; i++)
        if (g[i]<=S && Nr[S-g[i]]!= -1 && !Uz[S-g[i]][i])
            if (Nr[S]<1+Nr[S-g[i]])
                {Nr[S]=1+Nr[S-g[i]];
                 for (k=1; k<=n; k++) Uz[S][k]=Uz[S-g[i]][k];
                 Uz[S][i]=1;}
```

Pentru a afișa soluția, utilizăm informațiile memorate în tabloul `Uz`:

```
if (Nr[L]==-1) cout<<"IMPOZIBIL";
else
    cout<<"Numarul maxim de carti este: "<<Nr[L]<<endl;
    for (int k=1; k<=n; k++)
        if (Uz[L][k]) cout <<g[k]<< " ";
```

9. Telecomanda

Cu ocazia olimpiadei naționale de informatică, televiziunica locală organizează un nou joc în direct. Organizatorii utilizează un calculator, care generează și afișează pe un monitor două numere de maxim 100 de cifre fiecare (X și Y).

afisaj				
0	1	2	3	4
5	6	7	8	9
+	-	*	/	# =

Fiecare concurent dispune de o telecomandă prevăzută cu un afișaj de o cifră și cu anumite taste, ca în figura alăturată. Telecomanda are și o memorie, în care sunt reținute în ordine cifrele obținute de concurenți.

Cifrele primului număr (X) sunt afișate succesiv pe afișajul telecomenzii fiecărui concurent, în ordine de la stânga la dreapta. Concurenții trebuie să transforme primul număr, obținând în memoria telecomenzii proprii pe cel de al doilea, utilizând tastele pe care le au la dispoziție pe telecomandă. După efectuarea unei operații asupra cifrei curente (cea de pe afișaj), pe afișaj apare automat următoarea cifră din X (dacă mai există). Efectele apăsării tastelor sunt următoarele:

Taste actionate	Efect
+ urmat de o cifră	Se generează suma dintre cifra de pe afișaj și cifra tastată (operație posibilă doar dacă suma este tot o cifră). Cifra sumă este reținută în memorie.

- urmat de o cifră	Se generează diferența dintre cifra de pe afișaj și cifra tastată (operare posibilă doar dacă se obține tot o cifră). Cifra obținută este reținută în memorie.
* urmat de o cifră	Se reține în memorie valoarea tastei care se activează după tasta *. Deoarece asupra cifrei curente din X nu se efectuează nici o operare, aceasta nu dispără de pe afișaj.
/	Se sterge cifra curentă din X.
#	Se sterg din X cifra curentă și toate cifrele care urmează, până la sfârșit.
=	Se copiază în memorie cifra curentă.

Acțiunea se încheie atunci când toate cifrele lui X au fost prelucrate. Am obținut o soluție când în memoria telecomenzi se află cifrele numărului Y. O soluție este *optimă* dacă numărul de taste activeate este minim. Căștigătorii jocului sunt acei concurenți care descoperă o soluție optimă.

Date fiind X și Y, scrieți un program care să determine o soluție optimă de transformare a numărului X în numărul Y.

(ONI, Bacău 2001, clasele a XI-a – a XII-a)

Date de intrare

Fișier de intrare TELE.IN conține două linii:

X
Y

Date de ieșire

Fișier de ieșire TELE.OUT conține două linii:

min

t₁t₂...t_{min}
unde:

- min este un număr natural nenul, reprezentând numărul minim de taste activeate pentru transformarea lui X în Y.
- t₁t₂...t_{min} este o succesiune de min caractere, care reprezintă tastele activeate; între caractere nu se vor pune separatori.

Exemplu

TELE.IN	TELE.OUT
372	4
78	/=+6

Timp maxim de execuție: 1 secundă/test.

Soluție

1. Să notăm cu m numărul de cifre din X și cu n numărul de cifre din Y. De asemenea vom nota Xⁱ sufîxul care începe cu cifra i din X (secvența de cifre X_iX_{i+1}...X_n) și cu Y^j sufîxul care începe cu cifra j din Y (secvența de cifre Y_jY_{j+1}...Y_n). Subproblemele problemei date constau în determinarea transformării de cost minim a lui Xⁱ în Y^j, $\forall i \in \{1, 2, \dots, m\}$ și $j \in \{1, 2, \dots, n\}$.

2. Pentru a reține soluțiile subproblemelor vom utiliza două matrice cm și o, fiecare cu m linii și n coloane, având următoarea semnificație:

- cm[i][j] = numărul minim de activeări de taste necesar pentru a transforma Xⁱ în Y^j.
- o[i][j] = prima operare ce va fi executată pentru a transforma optimal Xⁱ în Y^j.

```
typedef char Operatie[3];
char x[DimMax], y[DimMax];
int cm[DimMax][DimMax];
Operatie o[DimMax][DimMax];
int n, m;
```

Evident, cm[1][1] reprezintă numărul minim de activeări de taste necesar pentru a transforma numărul X în numărul Y.

3. Relația de recurență care caracterizează substructura optimă a problemei este:

```
cm[i][j]=min
    {1+cm[i+1][j], //operatia /
     2+cm[i][j+1], //operatia *Y[j]
     2+cm[i+1][j+1], //operatia - X[i]-Y[j], daca X[i]>Y[j]
     2+cm[i+1][j+1], //operatia + Y[j]-X[i], daca X[i]<Y[j]
     1+cm[i+1][j+1], //operatia =
     1+cm[m+1][j]) //operatia #
```

4. Secvența de inițializare:

```
void Init()
{
    m=strlen(x); n=strlen(y);
    for (int i=1; i<=m; i++)
        {cm[i][n+1]=1; strcpy(o[i][n+1],"#");}
    for (i=1; i<=n; i++)
        {cm[m+1][i]=2*(n-i+1);
         for (int j=i; j<=n; j++)
             {o[m+1][j][0]='*';
              o[m+1][j][1]=y[j]; o[m+1][j][2]=NULL;}}
    }
```

Rezolvăm relația de recurență în mod *bottom-up*:

```
Operatie omin;
int min;
for (int j=n; j>0; j--)
    for (int i=m; i>0; i--)
        {min=1+cm[i+1][j]; strcpy(omin,"/");
         if (min>2+cm[i][j+1])
             {min=2+cm[i][j+1];
              omin[0]='*'; omin[1]=y[j]; omin[2]=NULL; }
         if (x[i]>y[j])
```

```

    {if (min>2+cm[i+1][j+1])
        {min=2+cm[i+1][j+1];
        omin[0]='-'; omin[1]=x[i]-y[j]+'\0';
        omin[2]=NULL;}
    }
    else
    if (x[i]<y[j])
        {if (min>2+cm[i+1][j+1])
            {min=2+cm[i+1][j+1];
            omin[0]='+'; omin[1]=y[j]-x[i]+'\0';
            omin[2]=NULL;}
        }
        else
        {if (min>1+cm[i+1][j+1])
            {min=1+cm[i+1][j+1]; strcpy(omin,"=");}
        }
    if (min>1+cm[m+1][j])
        {min=1+cm[m+1][j]; strcpy(omin,"#");}
    cm[i][j]=min;
    strcpy(o[i][j],omin);
}

```

O soluție optimă se obține utilizând matricea o, a operațiilor efectuate la fiecare pas.

```

ofstream f("tele.out");
f<<cm[1][1]<<endl;
for (int i=1, j=1; i<=m+1 && j<=n+1 && !(i==m+1 && j==n+1);)
    {f<<o[i][j];
     if ((o[i][j][0]=='+')||o[i][j][0]=='-'||o[i][j][0]=='=')
         {i++; j++;}
     else
     if (o[i][j][0] =='*') j++;
     else
     if (o[i][j][0] =='/') i++;
     else i=m+1;
    }
f.close();

```

10. Codificare optimală

Fie un text de lungime maximă 100, ce conține doar litere. Textul poate fi codificat, înlocuind aparițiile consecutive ale subșirurilor sale cu subșirul urmat de numărul său de apariții.

Exemplu

Textul $T = "aaacaaaacbbdefdef"$ poate fi codificat: "a3c1a3c1b2def2", dar și "aac2b2def2". Evident, cel de-al doilea mod de codificare este mai scurt, deci mai convenabil.

Scrieți un program care să codifice optimal un text dat (codul rezultat să fie de lungime minimă).

Soluție

1. Spațiul subproblemelor problemei inițiale este format din determinarea unei codificări optimale pentru caracterele din text de la i până la j ($T[i..j]$), $0 \leq i \leq j < n$, unde n este lungimea textului. Evident, subproblemele se suprapun. De exemplu, determinarea codificării optime pentru $T[i..j]$ necesită determinarea unor codificări optime pentru $T[i..k]$ și pentru $T[k+1..j]$, $k \in \{0, 1, \dots, j-1\}$.
2. Pentru a memora soluțiile subproblemelor, vom utiliza o matrice l , pătratică de ordin n , având semnificația $l[i][j] =$ lungimea codificării optime pentru $T[i..j]$; $0 \leq i \leq j < n$. Observați că este utilizată numai jumătatea de deasupra diagonalei principale.
3. Problema are substructură optimală, caracterizată de următoarea relație de recurență: Orice caracter x se codifică pe două poziții ($\times 1$), deci:

$$\begin{aligned} l[i][i] &= 2, \quad \forall i \in \{0, 1, \dots, n-1\} \\ l[i][j] &= \min \{j-i+2, \min I, \min II\} \end{aligned}$$

unde:

- $j-i+2$ provine din codificarea întregului sir, urmat de 1
- $\min I = \min \{l[i][k] + l[k+1][j]\}$
 $k=i, j-1$

Codificăm optimal $T[i..j]$ concatenând codificările optimale ale sirurilor $T[i..k]$ și $T[k+1..j]$ (poziția $k \in \{i, i+1, \dots, j-1\}$ se alege în mod optimă).

- $\min II = \min \{\text{strlen}(s) + \text{NrCifre}(k)\}$, unde s subșir al lui $T[i..j]$, astfel încât $T[i..j] = ss..s$ (de k ori). Deci $T[i..j]$ se codifică sk .

Deoarece jumătatea de sub diagonala principală din matricea l este neutilizată, pentru a putea reconstituia soluția optimă, o vom utiliza astfel:

- $l[j][i] = 0$, dacă $l[i][j] = j-i+2$
- $l[j][i] = k$, unde k este poziția de splitare a textului pentru $l[i][j] = \min I$
- $l[j][i] = -k$, unde k este lungimea subșirului s care se repetă, pentru $T[i][j] = \min II$.

4. Rezolvarea relației de recurență:

```

for (int i=0; i<n; i++) l[i][i]=2;
for (int d=2; d<=n; d++) //codific subsiruri de lungime d
    for (i=0; i<=n-d; i++) //care incep la pozitia i
        {int j=i+d-1; //si se termina la pozitia j
        //cazul I- codificarea este intregul sir urmat de 1
        l[i][j]=d+1; l[j][i]=0; }
    //cazul II
        for (int k=i; k<j; k++)
            if (l[i][j]>l[i][k]+l[k+1][j])
                {l[i][j]=l[i][k]+l[k+1][j];
                 l[j][i]=-k;} //pozitia optima de splitare

```

```
//cazul III
for (k=1; k<=d/2; k++)
    if (d%k==0) //determin un subsir de lungime k
        if (SeRepeta(i,j,k))
            {l[i][j]=k+NrCifre(d/k);
             l[j][i]=-k; //retin lungimea subsirului
             break;}
    }
```

Observați că am utilizat funcția NrCifre() care determină numărul de cifre dintr-un număr natural, precum și funcția SeRepeta():

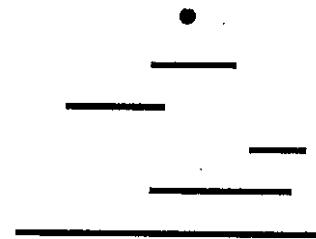
```
int SeRepeta (int i, int j, int k)
{ /*verifica daca sirul T[i..j] este format prin
concatenarea succesiva a lui T[i..i+k-1] */
char s[DimMax], s1[DimMax];
strncpy(s,T+i,k); s[k]=NULL; s1[k]=NULL;
for (int h=i+k; h<=j; h+=k)
    {strncpy(s1,T+h,-k);
     if (strcmp(s,s1)) return 0;}
return 1;}
```

Afișarea soluției optime o vom realiza prin apelarea funcției recursive Afisare(0, n-1):

```
ofstream f("cod.out");
void Afisare(int i, int j)
{
if (i==j) cout<<T[i]<<1; //un singur caracter
else
{int k=l[j][i];
char s[DimMax];
if (!k) // cazul I
    {strncpy(s,T+i,j-i+1); s[j-i+1]=NULL;
     cout<<s<<1; }
else
if (k>0) //cazul II
    {Afisare (i, k);
     Afisare (k+1,j);}
else //cazul III
    {strncpy(s,T+i,-k); s[-k]=NULL;
     cout<<s<<(j-i+1)/(-k); }
}
}
```

11. Falling

Considerăm un joc bazat pe un dispozitiv ca în imaginea de mai jos:



Dispozitivul constă dintr-o mulțime de platforme orizontale de diferite lungimi, plasate la diferite înălțimi. Platforma cea mai joasă este podeaua (este plasată la înălțimea 0 și are lungime infinită).

Dintr-o poziție specificată, este lăsată să cadă o mingă, la momentul 0. Mingea cade cu viteză constantă de 1 metru pe secundă. Când mingea ajunge pe o platformă, începe să se rostogolească spre unul din capetele acesteia, la alegerea jucătorului, cu aceeași viteză de 1 metru pe secundă. Când mingea ajunge la capătul platformei, își continuă cădere liberă, pe verticală. Minge nu are voie să cadă liber mai mult de MAX metri o dată (între două platforme).

Scrieți un program care determină un mod de rostogolire a mingii pe platforme, astfel încât să ajungă la podea cât mai repede posibil.

(CEOI, România, 2000)

Date de intrare

Fisierul FALL.IN:

Linia 1: N X Y MAX

- Patru numere întregi, separate prin câte un spațiu: numărul de platforme (excluzând podeaua), poziția de plecare a mingii (coordonatele pe orizontală și verticală) și distanța maximă pe care are voie să cadă; platformele sunt numerotate de la 1 la N.

Liniile 2..N+1: X_{1,i} X_{2,i} H_i

- Trei numere întregi, separate prin spații; platforma *i* este situată la înălțimea *H_i*, între coordonatele orizontale *X_{1,i}* și *X_{2,i}* inclusiv (*X_{1,i}<X_{2,i}, i=1..N*).

Observații

- Diametrul mingii și grosimea platformelor se ignoră. Dacă mingea cade exact pe marginea unei platforme, se consideră că a căzut pe platformă.
- Oricare două platforme nu au puncte comune.
- Pentru datele de test există întotdeauna soluție.
- Toate dimensiunile sunt exprimate în metri.

Ieșire

Numele fisierului: FALL.OUT

Linia 1: TIME

- Un număr întreg, reprezentând timpul la care mingea atinge podeaua, conform soluției voastre.

Celelalte linii, până la sfârșitul fisierului:

P T D

- Trei numere întregi, separate prin spații. Mingea atinge platformă P la momentul T și se îndreaptă în direcția D (S pentru stânga și D pentru dreapta).
- Impactul cu podeaua nu trebuie să apară în aceste linii.
- Impacturile cu platformele se precizează astfel încât valorile succesive ale lui T să fie în ordine crescătoare.

Observație

Dacă există mai multe soluții posibile, se cere numai una.

Restricții

- $1 \leq N \leq 1000$
- $-20000 \leq x_{1,i}, x_{2,i} \leq 20000, \forall i \in \{1, 2, \dots, N\}$
- $0 < h_i < Y \leq 20000$

Exemplu

FALL.IN	FALL.OUT
3 8 17 20	23
0 10 8	2 4 D
0 10 13	1 11 D
4 14 3	3 16 D

Soluție

Vom reține informațiile despre platforme în 3 vectori ($x_1[]$, $x_2[]$, $h[]$). Pentru a uniformiza datele problemei vom introduce două platforme artificiale: platformă 0, reprezentată de punctul de plecare ($x_1[0]=x$, $x_2[0]=x$, $h[0]=y$) și podeaua, platformă $n+1$ ($x_1[n+1]=-30000$, $x_2[n+1]=30000$, $h[0]=0$).

```
#define NMax 1003
int x1[NMax], x2[NMax], h[NMax];
int n, hmax;
void citire()
{
ifstream fin("fall.in");
int x, y;
fin>>n>>x>>y>>hmax;
for (int i=1; i<=n; i++)
    fi>>x1[i]>>x2[i]>>h[i];
fi.close();
x1[0]=x; x2[0]=x; h[0]=y;
x1[n+1]=-30000; x2[n+1]=30000; h[n+1]=0;
n+=2;
fin.close();
}
```

Sorțez platformele în ordinea descrescătoare a înălțimilor lor. În vectorul $p[]$ voi reține indicii platformelor, în ordinea respectivă:

```
int p[NMax];
void sort()
{
int sch, aux;
for (int i=0; i<n; i++) p[i]=i;
```

```
do {sch=0;
    for (i=2; i<n; i++)
        if (h[p[i]]>h[p[i-1]])
            {aux=p[i]; p[i]=p[i-1]; p[i-1]=aux; sch=1;}
    while (sch); }
```

De asemenea, pentru fiecare platformă putem calcula indicele platformei pe care va cădea mingea în cazul în care se rostogolește prin capătul din stânga (memorat în vectorul $st[]$), respectiv indicele platformei pe care va cădea mingea în cazul în care se rostogolește prin capătul din dreapta (memorat în vectorul $dr[]$):

```
int st[NMax], dr[NMax];
void StangaDreapta()
{
for (int i=0; i<n; i++)
{st[p[i]]=dr[p[i]]=0;
for (int j=i+1; h[p[i]]-h[p[j]]<=hmax &&
        (!st[p[i]] || !dr[p[i]]); j++)
    if (!st[p[i]] &&
        x1[p[j]]<=x1[p[i]] && x1[p[i]]<=x2[p[j]])
        st[p[i]]=p[j];
    if (!dr[p[i]] &&
        x1[p[j]]<=x2[p[i]] && x2[p[i]]<=x2[p[j]])
        dr[p[i]]=p[j];
}
}
```

Utilizând aceste informații precalculate, vom aborda rezolvarea problemei prin metoda programării dinamice.

1. Subproblemele problemei date constau în determinarea timpului minim în care ajunge mingea pe podea în cazul în care cade prin marginea din stânga a platformei i , respectiv prin marginea din dreapta a platformei i , $\forall i \in \{0, 1, \dots, n-1\}$.

2. Vom reține soluțiile subproblemelor utilizând 4 vectori $mst[]$, $mdr[]$, $undest[]$ și $undedr[]$, având următoarea semnificație:

- $mst[i]$ = timpul minim în care mingea ajunge pe podea, dacă pornește din marginea din stânga a platformei i ;
- $mdr[i]$ = timpul minim în care mingea ajunge pe podea, dacă pornește din marginea din dreapta a platformei i ;
- $undest[i]$ = direcția în care trebuie să se deplaseze mingea atunci când cade din marginea stângă a platformei i ('S' pentru stânga, 'D' pentru dreapta);
- $undedr[i]$ = direcția în care trebuie să se deplaseze mingea atunci când cade din marginea dreaptă a platformei i ('S' pentru stânga, 'D' pentru dreapta).

Evident, timpul minim de cădere va fi memorat în $mst[0] = mdr[0]$.

3. Rezolvarea relației de recurență în mod *bottom-up* o vom face parcurgând platformele de jos în sus:

```
long int mst[NMax], mdr[NMax];
char undest[NMax], undedr[NMax];
void calcMin()
```

```

{ int i, j, lgst, lgdr;
mst[n-1]=mdr[n-1]=0;
for (j=n-2; j>=0; j--)
    {i=p[j]; mst[i]=mdr[i]=Infinit;
     if (st[i]) //mingea cade prin stanga platformei i
        {
            if (st[i]==n-1) lgst=lgdr=0;
            else
                {lgst=mst[st[i]]+abs(x1[i]-x1[st[i]]));
                 lgdr=mdr[st[i]]+abs(x1[i]-x2[st[i]]));
            if (lgst<lgdr)
                {mst[i]=lgst+(h[i]-h[st[i]]));
                 undest[i]='S'; //cade pe st[i] spre stanga
                else
                    {mst[i]=lgdr+(h[i]-h[st[i]]));
                     undest[i]='D';
                }
            if (dr[i]) //mingea cade prin dreapta platformei i
                {
                    if (dr[i]==n-1) lgst=lgdr=0;
                    else
                        {lgst=mst[dr[i]]+abs(x2[i]-x1[dr[i]]));
                         lgdr=mdr[dr[i]]+abs(x2[i]-x2[dr[i]]));
                    if (lgst<lgdr)
                        {mdr[i]=lgst+h[i]-h[dr[i]]; undedr[i]='S';
                        else
                            {mdr[i]=lgdr+h[i]-h[dr[i]]; undedr[i]='D';
                        }
                }
            }
        }
    }
}

```

4. Reconstituirea soluției o vom face utilizând informațiile deja memorate:

```

void Afisare ()
{
    ofstream fout("fall.out");
    long t; int i, x; char d;
    fout<<mst[0]<<endl;
    for (x=x1[0], i=t=0, d='S'; i<n-2; )
        //sunt pe platforma i și ma deplasez in directia d
        if (d=='S')
            {t+=h[i]-h[st[i]]+x-x1[i];
             x=x1[i]; d=undest[i]; i=st[i];
            }
        else
            {t+=h[i]-h[dr[i]]+x2[i]-x;
             x=x2[i]; d=undedr[i]; i=dr[i];
            }
    fout<<i<<' '<<t<<' '<<d<<endl;
}
fout.close();

```

12. Florărie

Presupunem că aveți o florărie și că dorîți să aranjați vitrina într-un mod cât mai plăcut. Aveți F buchete de flori diferite și cel puțin tot atâtea vase, aranjate pe un raft. Vazele sunt lipite de raft și sunt numerotate în ordine, de la stânga la dreapta, de la 1 la V, unde V este numărul de vase. Mai exact vaza 1 este cea mai din stânga, iar vaza V cea mai din dreapta. Buchetele sunt și ele numerotate distinct, cu numere între 1 și F. Aceste numere de identificare a buchetelor au o semnificație: ele determină ordinea de apariție cerută pentru buchete în vase. Mai exact, buchetul i trebuie să fie într-o vază din stânga vasei care conține buchetul j, dacă i < j.

Să presupunem, de exemplu, că aveți un buchet de azalee (cu numărul de identificare 1), un buchet de begonii (cu numărul de identificare 2) și un buchet de trandafiri (cu numărul de identificare 3). Acum, toate buchetele trebuie puse în vase, astfel încât ordinea numerelor lor de identificare să se păstreze. Bucetul de azalee trebuie să fie într-o vază din stânga begoniilor, care trebuie să fie într-o vază din stânga trandafirilor. Dacă există mai multe vase decât buchete, cele în plus vor rămâne goale. O vază poate conține un singur buchet de flori.

Fiecare vază are o caracteristică proprie (asemănător florilor). Prin urmare, așezând un buchet de flori într-o anumită vază, obțineți un anumit efect estetic, exprimat printr-o valoare întreagă. Valorile estetice sunt reprezentate sub forma unui tablou, ca în exemplul de mai jos. Dacă o vază rămâne goală, valoarea estetică este 0.

		V A Z E				
		1	2	3	4	5
Buchete	1 (azalee)	7	23	-5	-24	16
	2 (begonii)	5	21	-4	10	23
	3 (trandafiri)	-21	5	-4	-20	20

Conform acestui tabel, azaleele, de exemplu, vor arăta grozav în vaza 2 și groaznic în vaza 4.

Pentru a obține cel mai plăcut efect, trebuie să maximizați suma valorilor estetice ale vazelor, păstrând ordinea buchetelor. Dacă există mai multe soluții, se cere numai una.

(IOI, Turcia, 1999)

Restricții

- $1 \leq F \leq 100$, unde F este numărul de buchete de flori.
- $F \leq V \leq 100$, unde V este numărul de vase.
- $-50 \leq A_{ij} \leq 50$, unde A_{ij} este valoarea estetică care se obține punând buchetul i în vaza j.
- Timp maxim de execuție: 2 secunde/test.

Date de intrare

Fișierul de intrare se numește flower.in. Prima linie conține două numere: F V. Fiecare din următoarele F linii conține V numere întregi, astfel încât A_{ij} este al j-lea număr de pe linia (i+1) din fișierul de intrare.

Date de ieșire

Fișierul de ieșire flower.out conține două linii. Prima linie conține suma valorilor estetice a aranjamentului floral. A doua linie reprezintă aranjamentul floral, ca o listă de F numere, astfel încât cel de-al k-lea număr de pe linie identifică vaza în care este pus buchetul k.

Exemplu

flower.in	flower.out
3 5	53
7 23 -5 -24 16	2 4 5
5 21 -4 10 23	
-21 5 -4 -20 20	

Soluție

- Subproblemele problemei date constau în amplasarea optimă (cu maximizarea sumei valorilor estetice) a buchetelor $1 \dots i$ în vasele $1 \dots j$, respectând ordinea buchetelor ($i \in \{1, 2, \dots, F\}$, $j \in \{j_i, \dots, V-F+i\}$).
- Pentru a reține soluțiile subproblemelor vom utiliza o matrice c cu F linii și V coloane ($c[i][j] =$ valoarea estetică optimă care se poate obține amplasând buchetele $1 \dots i$ în vasele $1 \dots j$, respectând ordinea buchetelor). Evident, valoarea estetică optimă se obține în $c[F][V]$. Pentru reconstituirea soluției optime, vom utiliza o matrice poz cu F linii și V coloane ($poz[i][j] =$ vaza în care plasăm buchetul i , într-o amplasare estetică optimă a buchetelor $1 \dots i$ în vasele $1 \dots j$).
- Pentru a amplasa optim buchetele $1 \dots i$ în vasele $1 \dots j$, respectând ordinea buchetelor ($i \in \{1, 2, \dots, F\}$, $j \in \{j_i, \dots, V-F+i\}$) avem două posibilități (dintre care o alegem pe cea mai convenabilă): fie amplasăm buchetul i în vaza j și amplasăm optimul buchetele $1 \dots i-1$ în vasele $1 \dots j-1$; fie amplasăm buchetele $1 \dots i$ în vasele $1 \dots j-1$. Prin urmare, caracterizăm substructura optimă a problemei prin următoarea relație de recurență:

```
c[i][i-1]=-∞; ∀i∈{1, 2, ..., F}
c[0][j]=0; ∀j∈{0, ..., V}
c[i][j]=max{A[i][j]+c[i-1][j-1], c[i][j-1]}, ∀i∈{1, 2, ..., F},
∀j∈{j_i, ..., V-F+i}
```

- Pentru implementare observăm că putem inițializa $c[i][j]$ cu valoarea A_{ij} , astfel încât să nu mai fie necesar să utilizăm o matrice suplimentară A. Rezolvăm relația de recurență în mod bottom-up astfel:

```
int i, j;
for (i=1; i <= F; i++) c[i][i-1]=MINF;
for (i=1; i <= F; i++)
    for (j=i; j <= (V - F + i); j++)
        if (c[i-1][j-1]+c[i][j]>c[i][j-1])
            {c[i][j] += c[i-1][j-1];
             poz[i][j]=j;}
```

```
else
    {c[i][j]=c[i][j-1];
     poz[i][j]=poz[i][j-1];}
```

Afișarea soluției o vom face utilizând valorile memorate în matricea poz:

```
ofstream f("flower.out");
f<<c[F][V]<<endl;
int sol[MaxNr];
for (int i=F, j=V; i>0; j--)
    if (poz[i][j]==j)
        {sol[i]=j; i--;}
for (i=1; i<=F; i++) f<<sol[i]<<' ';
f.close();
```

13. Vin bun

La un depozit specializat din Recaș sosesc, pe rând, n clienți care solicită fiecare o cantitate dată de Cabernet. În butoiul din magazinul de deservire (considerat de capacitate nelimitată) nu se găsește inițial nici o picătură de vin. Dan Septică, administratorul depozitului, are un algoritm propriu de deservire a clientilor: în funcție de ceea ce are în butoi, dar și de inspirația de moment, el poate să răspundă: „Vă ofer întreaga cantitate cu cea mai mare placere!” sau „Nu vă pot oferi acum ceea ce dorîți, reveniți cu solicitarea altădată!”.

Pe clientul servit îl eliberează de grija banilor corespunzători costului licorii cumpărate, iar pe cel refuzat îl salută politicos și are grija ca, imediat ce a plecat clientul, să coboare și să aducă în butoiul din magazin exact cantitatea solicitată de clientul respectiv (cel ce nu a fost servit).

Clienții sunt restaurante de lux și nu cumpără mai mult de 1 hectolitru.

Cunoscând cele n cantități cerute de clienti, să se determine un mod de a răspunde solicitărilor astfel încât, în final, cantitatea de vin vândută să fie maximă.

Date de intrare

Din fișierul de intrare CERERI.IN se citesc:

n	– numărul de clienti ($n \leq 600$);
c1 c2 ... cn	– cantitățile (în litri) cerute de clienti, în ordinea sosirii lor.

Date de ieșire

În fișierul VANZARI.OUT se va scrie cantitatea vândută (tot în litri).

Exemplu

CERERI.IN	VANZARI.OUT
4	6
5 3 4 2	

Interpretare

Acțiunile lui Șeptică sunt: refuză primul client și pune 5 litri în butoi, îl refuză pe al doilea și mai pune încă 3 litri în butoi, iar pe următorii doi clienți îi servește, realizând o vânzare de 6 litri.

(ONI, Timișoara, 1997, clasa a X-a)

Soluție

Subproblemele problemei date constau în determinarea cantității maxime de vin ce poate fi vândută luând în considerare cererile clientilor $i \dots n$, în ipoteza că inițial avem în butoi o cantitate j . Soluțiile acestor subprobleme le vom reține într-o matrice v .

Relația de recurență o obținem astfel:

$$\begin{aligned} v[i][0] &= 0, \quad i \in \{1, 2, \dots, n\} \\ v[i][j] &= \max\{v[i+1][j+c[i]] \text{ (cazul în care clientul } i \text{ este refuzat)} \\ &\quad c[i]+v[i+1][j-c[i]], \text{ dacă } j \geq c[i] \\ &\quad \text{(cazul în care clientul } i \text{ este servit)} \end{aligned}$$

Soluția problemei inițiale o vom obține în $v[1][0]$.

O problemă de implementare o constituie dimensiunile matricei v : i variază de la 1 la n , deci numărul de linii este maxim 600. Numărul de coloane este determinat de cantitatea maximă care se poate acumula în butoi (în cazul cel mai defavorabil cei 600 de clienți comandă fiecare câte 100 de litri), deci numărul maxim de coloane ar fi 60000. Nu putem aloca în memorie o matrice de 600×60000 , dar observăm că pentru determinarea liniei i din matricea v este necesară numai linia $i+1$. Prin urmare este suficient să utilizăm doi vectori de 60000 de componente – $v1$ pentru a reține linia precedentă (linia $i+1$) și $v2$ pentru a calcula linia curentă (linia i). Acești doi vectori vor fi alocati dinamic, numărul de elemente alocate fiind \max (suma tuturor cantităților cerute de clienti):

```
unsigned *v1, *v2;
v1=new unsigned[max];
v2=new unsigned[max];
if(!v1 || !v2)
    { cout<<"Nu s-a putut aloca memorie.";
      exit(0); }
for(i=0; i<max; i++) v1[i]=0;
```

Rezolvarea relației de recurență:

```
unsigned i, j, *aux;
for(i=n; i>=1; i--)
    { for(j=0; j<c[i]; j++) v2[j]=v1[j+c[i]];
      for ( ; j<max-c[i]; j++)
          { v2[j]=v1[j+c[i]];
            if (v2[j]< c[i]+v1[j-c[i]])
                v2[j]=c[i]+v1[j-c[i]]; }
      aux=v1; v1=v2; v2=aux;
    }
```

14. Siruri 2-3-monotone

Fie n ($1 \leq n \leq 30$) un număr natural. Numim *sir 2-3-monoton* de lungime n un sir s_1, s_2, \dots, s_n , format din n elemente ale mulțimii $\{1, 2, \dots, n\}$ care verifică următoarele două relații:

- $s_i < s_{i+2}$, $\forall i \in \{1, 2, \dots, n-2\}$
- $s_i < s_{i+3}$, $\forall i \in \{1, 2, \dots, n-3\}$

Scripti un program care să determine numărul de siruri 2-3-monotone de lungime n și să afișeze ultimele 4 cifre din acest număr.

(Tabăra de pregătire, București, 2001)

Exemple

Pentru $n=2$, programul va afișa 4. Pentru $n=3$, va afișa 9.

Soluție

Din definiția sirului 2-3-monoton, deducem că $s_i < \min s_{i+k}$, $\forall k \geq 2$. Prin urmare, într-un sir 2-3-monoton pot exista cel mult doi termeni consecutivi egali. Să notăm $F(p, q)$ numărul sirurilor 2-3-monotone de lungime q , formate din termeni care aparțin unei mulțimi cu p elemente (de exemplu, mulțimii $\{1, 2, \dots, p\}$). Evident, $n_r = F(n, n)$.

Observații

- $F(p, 1) = p$;
- $F(p, 2) = p^2$;
- $F(1, q) = 1$, dacă $q = 1$ sau $q = 2$
- 0, dacă $q > 2$;
- $F(p, 2p) = 1$ (sirul unic: 1122334455 ...)
- $F(p, q) = 0$, dacă $q > 2p$;
- $F(2, 3) = 2$ (sirurile 112 și 122).

Relația de recurență:

Cazul 1: $s_1=1, s_2=1$

Numărul de siruri 2-3-monotone care se obțin astfel este egal cu $F(p-1, q-2)$.

Cazul 2: $s_1=1, s_2>1$

Numărul de siruri 2-3-monotone care se obțin astfel este egal cu $F(p-1, q-1)$.

Cazul 3: $s_1=k$ ($k > 1$), $s_2=j$, $j \leq k$

Numărul de siruri 2-3-monotone care se obțin astfel este $k * F(p-k, q-2)$.

Cazul 4: $s_1=k$ ($k > 1$), $s_2>k$

Numărul de siruri 2-3-monotone care se obțin astfel este egal cu $F(p-k, q-1)$.

Reunind aceste cazuri obținem astfel relația de recurență:

$$F(p, q) = \sum_{k=1}^{p-1} F(p-k, q-1) + \sum_{k=1}^{p-1} k * F(p-k, q-2)$$

Rezolvăm această recurență în mod *bottom-up*.

```
int i, j, k, mod=10000;
for (i=1; i<=n; i++)
    {f[i][1]=i; f[i][2]=i*i; f[i][2*i]=1;}
f[1][2]=1; f[2][3]=2;
for (i=3; i<=n; i++)
    for (j=3; j<2*i; j++)
        for (k=1; k<i; k++)
            f[i][j]=(f[i][j]+f[i-k][j-1]+k*f[i-k][j-2]) % mod;
```

Probleme propuse

1. Triangulizare optimă a unui poligon convex

Fie $P = (P_1, P_2, \dots, P_n)$ vârfurile, în ordine, ale unui poligon convex. Numim diagonală a poligonului un segment (P_i, P_j) , $i \neq j$, diferit de laturile poligonului.

Numim *triangulizare* o mulțime de diagonale astfel încât partajează suprafața poligonului în triunghiuri. Definim *costul* unei triangulizări ca fiind suma lungimilor diagonalelor care constituie triangulizarea. Determinați o triangulizare optimă a poligonului convex dat (o triangulizare de cost minim).

2. O relație de ordine ghidușă

Se dă un sir de n ($0 < n < 501$) numere naturale. Spunem că x este *mai ghiduș* decât y dacă reprezentarea binară a lui x conține mai puține cifre 1 decât reprezentarea binară a lui y . Să se formeze un nou sir cu un număr maxim de elemente din sirul dat, fără a modifica ordinea inițială, astfel încât orice element al nouului sir este mai ghiduș decât următorul.

(OJ, Alba, clasa a X-a)

Exemplu

Pentru $n=10$ și sirul 15 2 64 12 12 8 7 2 15 62 o soluție posibilă este:

64 12 7 15 62

3. Pod

Între două maluri ale unei văi adânci s-a construit un pod suspendat format din N bucăți de scândură, legate cu jiane. Vom considera că scândurile sunt numerotate de la 1 la N , începând de pe malul pe care ne aflăm. În timp, unele bucăți de scândură s-au deteriorat, iar altele chiar au dispărut. Pentru traversarea podului se știe că:

- se pot face pași doar de lungime 1, 2 sau 3;
- scândurile deteriorate sunt nesigure, deci pe ele și de pe ele se pot face doar pași de lungime 1.

Evident, nu se poate pași pe o scândură care lipsesc.

Scrieți un program care să determine numărul de modalități de traversare a podului (mai exact, de a ajunge pe celălalt mal), precum și o soluție de traversare, dacă o astfel de soluție există.

Date de intrare

Fișierul de intrare POD.IN are structura:

POD.IN	Semnificație
N	– Numărul total de scânduri
$k \ s_1 \ s_2 \ \dots \ s_k$	– Numărul de scânduri lipsă și numerele lor de ordine
$h \ d_1 \ d_2 \ \dots \ d_h$	– Numărul de scânduri deteriorate și numerele lor de ordine

Date de ieșire

Fișierul de ieșire POD.OUT va conține pe prima linie valoarea -1 dacă nu este posibil să traversăm podul, respectiv numărul de posibilități de a traversa podul, dacă aceasta este posibil. În cazul în care există soluții, pe cea de a doua linie va fi afișată o astfel de soluție, prin indicarea, în ordine, a scândurilor pe care se pășește, sub forma:

POD.OUT	Semnificație
Nr	– Numărul total de posibilități
$P_1 \ P_2 \ \dots \ P_m$	– Soluția determinată, prin indicarea în ordine a scândurilor pe care se pășește

Restricții și precizări

- $3 \leq N \leq 300$
- $0 \leq k, h \leq N$
- $\{s_1, s_2, \dots, s_k\} \subseteq \{2, \dots, N\}$, $\{d_1, d_2, \dots, d_h\} \subseteq \{1, 2, \dots, N\}$;
- $\{s_1, s_2, \dots, s_k\} \cap \{d_1, d_2, \dots, d_h\} = \emptyset$
- Nr are cel mult 80 de cifre.
- Timp maxim de execuție: 1 secundă/test.

Exemplul 1

POD.IN	POD.OUT
5	24
0	3
0	

Exemplul 2

POD.IN	POD.OUT
10	48
2 2 7	3 6 8
1 5	

Exemplul 3

POD.IN	POD.OUT
6	-1
2 2 4	
1 3	

(ONI, Brăila, 2002, clasa a IX-a)

4. Câmpuri energetice

Un robot independent construit astfel încât să-și refacă singur resursele necesare funcționării prin parcurserea unor zone speciale ale planetei „natale” (numite zone energetice) se află într-o misiune militară. El se află la marginea unei zone energetice dreptunghiuare de dimensiuni $M \times N$ pe care trebuie să traverseze de la linia 1 la linia M , deplasările posibile fiind dintr-o căsuță (i, j) a unei linii în oricare dintre căsuțele $(i+1, j-1)$, $(i+1, j)$, $(i+1, j+1)$ ale liniei următoare, știut fiind că două staționări pe o aceeași linie a dreptunghiuului ar conduce la detectarea și distrugerea lui de către robotul înamic ce supraveghează linia respectivă. Deplasarea între două căsuțe, precum și intrarea sau ieșirea din zonă determină un consum de K unități de energie, iar trecerea printr-o celulă determină adăugarea la rezerva energetică a robotului a unei cantități de energie caracteristice celulei respective.

Date de intrare

Fișierul de intrare ROBOT.IN conține:

M N	- dimensiunile zonei
K	- consumul de energie la trecerea într-o nouă căsuță
R	- rezerva inițială a robotului
p ₁₁ p ₁₂ ... p _{1N}	- potențialele energetice ale căsuțelor de pe linia 1
p ₂₁ p ₂₂ ... p _{2N}	- potențialele energetice ale căsuțelor de pe linia 2
...	
p _{M1} p _{M2} ... p _{MN}	- potențialele energetice ale căsuțelor de pe linia N

Date de ieșire

În fișierul de ieșire ROBOT.OUT se vor afișa mesajul IMPOSSIBIL (dacă robotul nu poate traversa câmpul energetic) sau o succesiune de căsuțe scrise ca perechi de numere (i,j) despărțite prin câte un spațiu, precum și rezerva energetică finală a robotului (după părăsirea zonei).

Restricții

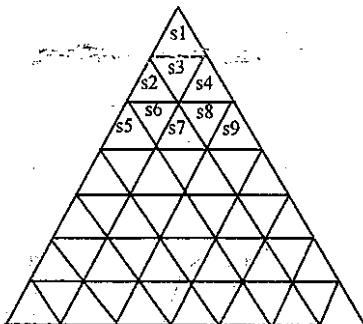
- M, N < 100
- 1 ≤ K ≤ 20
- p_{ij} ≤ 100
- R ≤ 1000

5. FNI triunghiular

Un „FNI triunghiular” este o clădire care are secțiunea în formă de triunghi echilateral, iar încăperile corespund unor triunghiuri echilaterale de latură 1. Pe fiecare latură a clădirii există exact N încăperi. În fiecare încăpere se găsește câte un seif care conține o sumă de bani.

Fiecare perete al încăperilor are o ușă, inițial deschisă, iar pe una din ușile care comunică cu exteriorul în clădire poate intra un hoț cu scopul de „colectă” o sumă cât mai mare de bani din seifurile existente în camere. După intrarea în clădire, indiferent pe care ușă exterioră, toate ușile exterioare se închid și nu mai pot fi deschise decât din interior (pentru a putea părăsi clădirea). De asemenea, după părăsirea unei încăperi din care au fost luați banii, toate cele trei uși ale acesteia se închid automat și nu mai pot fi deschise.

Determinați suma maximă care poate fi colectată de hoț și numărul de încăperi prin care trebuie să trece hoțul pentru a colecta suma respectivă.

*Date de intrare*

Fișierul de intrare FNI.IN conține:

N
s₁
s₂ s₃ s₄
s₅ s₆ s₇ s₈ s₉
...
s_{N(N-2)} ... s_{N*N}

Date de ieșire

Fișierul de ieșire FNI.OUT conține două numere naturale nenule, despărțite prin spațiu, reprezentând suma maximă colectată și numărul de încăperi vizitate.

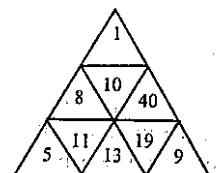
Restricții

- 1 ≤ n ≤ 100
- 0 ≤ s_i ≤ 300 $\forall i \in \{1, 2, \dots, N\}$

Exemplu

FNI.IN
3
1
8 10 40
5 11 13 19 9

FNI.OUT
102 7



(Concursul „Urmașii lui Moisil”, Iași, 2001)

6. Aliniere

În armată, o companie este alcătuită din n soldați. La inspecția de dimineață soldații stau aliniați în linie dreaptă în fața căpitanului. Acesta nu e mulțumit de ceea ce vede; e deosebit că soldații sunt așezăți în ordinea numerelor de cod 1, 2, ..., n din registrul, dar nu în ordinea înălțimii. Căpitanul cere cătorva soldați să iasă din rând, astfel încât cei rămași, fără a-și schimba locurile, doar apropiindu-se unul de altul (pentru a nu rămâne spații mari între ei) să formeze un sir în care fiecare soldat vede, privind de-a lungul șirului, cel puțin una din extremități (stânga sau dreapta). Un soldat vede o extremitate dacă între el și capătul respectiv nu există un alt soldat cu înălțimea mai mare sau egală cu a lui.

Scrieți un program care determină, cunoscând înălțimea fiecărui soldat, numărul minim de soldați care trebuie să părăsească formația, astfel încât șirul rămas să îndeplinească condiția din enunț.

Date de intrare

Pe prima linie a fișierului de intrare ALINIERE.IN este scris numărul n al soldaților din șir, iar pe linia următoare un sir de n numere reale, cu maximum 5 zecimale fiecare și separate prin spații. Al k-lea număr de pe această linie reprezintă înălțimea soldatului cu codul k ($1 \leq k \leq n$).

Date de ieșire

Fișierul ALINIERE.OUT va conține pe prima linie numărul soldaților care trebuie să părăsească formația, iar pe linia următoare codurile acestora în ordine crescătoare, separate două căte două printr-un spațiu. Dacă există mai multe soluții posibile, se va scrie una singură.

Restricții

- $2 \leq n \leq 1000$
- Înălțimile sunt numere reale în intervalul $[0.5; 2.5]$
- Timp maxim de execuție: 1 secundă/test.

Exemplu

ALINIERE.IN

8

1.86 1.86 1.30621 2 1.4.1 1.97 2.2

ALINIERE.OUT

4

1 3 7 8

Explicație

Rămân soldații cu codurile 2, 4, 5, 6 având înălțimile 1.86 2 1.4 1. Soldatul cu codul 2 vede extremitatea stângă. Soldatul cu codul 4 vede ambele extremități. Soldații cu codurile 5 și 6 văd extremitatea dreaptă.

(ONI, Brăila, 2002, clasa a X-a)

7. Balanță

Gigel are o „balanță” mai ciudată pe care vrea să o echilibreze. De fapt, aparatul este diferit de orice balanță pe care ați văzut-o până acum. Balanța lui Gigel dispune de două brațe de greutate, neglijabilă și lungime 15 fiecare. Din loc în loc, la aceste brațe sunt atașate cărlige, pe care Gigel poate atârna greutăți din colecția sa de G greutăți ($1 \leq G \leq 20$) de valori distincte (numere naturale între 1 și 25). Gigel poate atârna oricâte greutăți de orice cărlig, dar trebuie să folosească toate greutățile de care dispune.

Folosindu-se de experiența participării la Olimpiada Națională de Informatică, Gigel a reușit să echilibreze balanță relativ repede, dar acum dorește să știe în câte moduri poate fi ea echilibrată.

Cunoscând amplasamentul cărligelor și setul de greutăți pe care Gigel îl are la dispoziție, scrieți un program care calculează în câte moduri se poate echilibra balanță.

(ONI, Brăila, 2002, clasa a X-a)

Date de intrare

Fișierul de intrare BALANTA.IN are următoarea structură:

- pe prima linie, numărul C de cărlige ($2 \leq C \leq 20$) și numărul G de greutăți ($2 \leq G \leq 20$);
- pe următoarea linie, C numere întregi distincte, ordonate crescător, cuprinse între -15 și 15 inclusiv, reprezentând amplasamentele cărligelor, prin poziția pe axa X față de centrul balanței, în condițiile în care nu este atârnată nici o greutate (deci balanța este echilibrată și se aliniază cu axa X); valoarea absolută a distanțelor reprezintă distanța față de centrul balanței, iar semnul precizează brațul balanței la care este atașat cărligul, '-' pentru brațul stâng și '+' pentru brațul drept);
- pe următoarea linie, G numere naturale distincte ordonate crescător, cuprinse între 1 și 25 inclusiv, reprezentând valorile greutăților pe care Gigel le va folosi pentru a echilibra balanță.

Date de ieșire

Fișierul de ieșire BALANTA.OUT conține o singură linie, pe care se află un număr M, reprezentând numărul de variante de plasare a greutăților care duc la echilibrarea balanței.

Restricții

- $2 \leq C \leq 20, 2 \leq G \leq 20$
- Greutățile folosite au valori naturale între 1 și 25.
- Numărul M cerut este între 1 și 2000000000.
- Timp maxim de execuție: 1 secundă/test.
- Pentru toate testele de la evaluare balanța se poate echilibra.

Observație

Balanța se echilibrează dacă suma produselor dintre greutăți și coordonatele unde ele sunt plasate este 0 (suma momentelor greutăților față de centrul balanței este 0).

Exemplu

BALANTA.IN

2 4

-2 3

3 4 5 8

BALANTA.OUT

2

8. Pești

Pescarul Naum a prins N pești. La mal a determinat greutatea t_i pentru fiecare pește. Naum lucrează pentru o companie de pescuit și are dreptul să ia acasă cel mult T grame de pește. El dorește să ia un număr cât mai mic de pești, dar care să aibă o greutate maximă posibilă L ($\leq T$). Scrieți un program care să decidă ce pești să ia acasă pescarul.

(BOI, Macedonia, 2000)

Restricții

- $N \leq 500000$ este un număr natural care reprezintă numărul de pești.
- $T \leq 7000$ este un număr natural care reprezintă greutatea maximă de pește pe care pescarul o poate lua acasă.
- $L \leq 35000$ este un număr natural care reprezintă greutatea peștelui i.

Exemplu

FISH.IN

10 280

300

10

30

80

200

20

20

60

10

100

FISH.OUT

2

200

80

Observații

- numărul de pești;
- greutățile celor doi pești selectați.

9. Recesiune

În urma unei situații de criză economică, patronul lanțului de benzinării de pe autostrada 66 trebuie să închidă o parte dintre acestea. Nu orice benzinărie poate fi închisă, deoarece benzinăriile rămase trebuie să fie dispuse astfel încât orice mașină cu „plinul făcut” să poată ajunge de la o benzinărie la alta. Mai exact, distanța dintre oricare două benzinării consecutive rămase nu trebuie să depășească o distanță maximă admisă DMax.

Se știe că pe autostradă există n benzinării ($n \leq 1000$), numerotate de la 1 la n. Se cunosc de asemenea distanțele dintre oricare două benzinării consecutive ($d_i =$ distanța dintre benzinăria i și benzinăria $i+1$, $\forall i \in \{1, 2, \dots, n-1\}$), precum și profitul anual obținut de fiecare benzinărie (p_1, p_2, \dots, p_n).

Scrieți un program care să determine numărul maxim de benzinării care pot fi închise, astfel încât distanța dintre oricare două benzinării consecutive să nu depășească distanța maximă admisă DMax. Dacă există mai multe soluții, se va afișa o soluție pentru care profitul anual total al benzinărilor rămase să fie maxim.

10. Labirint

Romeo și Julieta sunt prinși într-un labirint, reprezentat sub forma unei matrice cu M linii și N coloane, cu elemente 0 și 1. Un element 1 reprezintă zid, iar 0 reprezintă spațiu liber. Romeo și Julieta se află inițial în pătrătelele (1,1) (Romeo) respectiv (M,N) (Julieta) ale matricei (care conțin valoarea 0), se pot deplasa numai pe pătratele care conțin 0 și nu pot părași matricea.

Pentru a evada din labirint, Romeo trebuie să ajungă în pătrătelul (i_1, j_1) , iar Julieta în (i_2, j_2) . El încearcă să facă asta conform indicațiilor date de dumneavoastră, participant la Olimpiada Națională de Informatică.

Dumneavoastră dispuneți de un șir de mutări pe care cei doi le pot efectua. Șirul este format din K mutări, codificate prin literele N, E, S și V, reprezentând deplasări de un pătrătel spre Nord, Est, Sud, respectiv Vest. Romeo și Julieta trebuie să efectueze TOATE mutările din acest șir, în ordinea dată.

Dumneavoastră sunteți cel care decide dacă o anumită mutare va fi efectuată de Romeo sau de Julieta.

Se știe că șirul de mutări permite ajungerea celor doi la destinații. Dacă reușiti să-l folosiți în acest scop, fără a încălca restricțiile impuse de labirint, primiți puncte la Olimpiadă!

Scrieți un program care va decide care dintre mutări vor fi efectuate de Romeo și care de Julieta pentru a-i ajuta pe cei doi să ajungă la destinații.

Date de intrare

Fisierul de intrare LABIRINT.IN are următoarea structură:

- pe prima linie numerele M și N;
- pe următoarele M linii descrierea labirintului; pe linia $i+1$ se află descrierea liniei i a labirintului, codificată prin N numere 0 sau 1, separate prin spații;
- pe următoarea linie valorile $i_1 \ j_1 \ i_2 \ j_2$;
- pe următoarea linie numărul K de mutări care trebuie efectuate;
- pe următoarea linie un șir de K litere, fiecare literă fiind N, E, S sau V, cu semnificațiile de mai sus.

Date de ieșire

În fisierul de ieșire LABIRINT.OUT se va afișa o singură linie, conținând K caractere R sau J. Al i-lea caracter este R dacă Romeo va efectua a i-a mutare sau J dacă mutarea va fi efectuată de Julieta.

Restricții și precizări

- $3 \leq M \leq 20$;
- $3 \leq N \leq 60$;
- $1 \leq K \leq 200$.
- Toate testeile vor avea cel puțin o soluție. Dacă există mai multe soluții, se va afișa una dintre ele.
- Timp maxim de executare: 1 secundă/test.
- Prin deplasare la Nord se înțelege deplasarea pe linia precedentă, o deplasare la Sud înseamnă deplasarea pe linia următoare, deplasarea la Vest înseamnă deplasarea pe coloana precedentă, iar deplasarea la Est înseamnă deplasarea pe coloana următoare.

(Baraj Brăila, 2002)

Exemplu

LABIRINT.IN

```
5 5
0 0 1 0 0
0 0 0 1 1
0 0 0 0 0
1 0 0 0 0
0 1 0 0 0
2 2 4 4
6
SNEEVV
```

LABIRINT.OUT poate fi

RJRRRJ

LABIRINT.OUT poate fi

RJRRJR

11. Transformare de cuvinte

Se urmărește transformarea unui cuvânt denumit sursă într-un cuvânt denumit destinație folosind următoarele operații permise:

1. COPY(a)	transferă caracterul a din sursă în destinație
2. DELETE(a)	sterge caracterul a din sursă
3. INSERT(a)	inserează caracterul a în destinație
4. REPLACE(a, b)	înlocuiește caracterul a cu caracterul b (a dispare din sursă, iar în destinație apare b)
5. TWIDDLE(ab)	secvența ab din sursă se transformă în ba în destinație (ab dispare din sursă și apare ba în destinație)
6. KILL(secventa)	după ce s-au efectuat toate operațiile necesare, se poate elimina secvența rămasă (sufixul) din cuvântul sursă.

Exemplu

Pentru transformarea cuvântului sursă ALGORITM în cuvântul destinație ALTRUIST, se poate folosi următoarea secvență de operații:

Operatie	Destinație	Sursă
Copy (A)	A	LGORITM
Copy (L)	AL	GORITYM
Replace (G, T)	ALT	ORITM
Delete (O)	ALT	RITM
Copy (R)	ALTR	ITM
Insert (U)	ALTRU	ITM
Copy (I)	ALTRUI	TM
Insert (S)	ALTRUIS	TM
Copy (T)	ALTRUIST	M
Kill (M)	ALTRUIST	

Asociind către un cost fiecărei operații, problema cere să se determine o secvență de operații de cost minim (suma costurilor operațiilor să fie minimă).

12. Domino

Să considerăm o succesiune formată din n ($n \leq 1000$) piese de domino. O piesă de domino are formă dreptunghiulară și are înscrise pe ea două numere, cuprinse între 1 și 6. Conform regulilor de la domino un *lanț* este constituit din piese care respectă următoarea condiție: pentru oricare două piese consecutive pe lanț, al doilea număr înscris pe prima dintre cele două piese coincide cu primul număr înscris pe cea de-a doua piesă.

Scripteți un program care să determine cel mai lung lanț care se poate obține cu piesele din succesiunea dată, în ordinea în care acestea sunt în succesiune, fiind posibilă și eventuala întoarcere a unor piese.

De exemplu, pentru succesiunea formată din următoarele $n=8$ piese de domino cu valorile (1,5), (1,3), (2,6), (2,5), (4,3), (6,4), (2,4), (1,6), o soluție posibilă este: (5, 1) (1, 3) (3, 4) (4, 6) (6, 1).

13. Cod

Principala misiune a unei expediții științifice este de a studia evoluția vieții pe o planetă nou descoperită. În urma studiilor efectuate, cercetătorii au asociat fiecărui organism viu descoperit pe acea planetă un cod caracteristic. Codul caracteristic este un număr natural de maxim 200 de cifre zecimale nenule.

Cercetătorii au observat că pentru orice organism viu de pe planetă, codurile caracteristice ale strămoșilor săi pe scara evoluției se pot obține prin ștergerea unor cifre din codul caracteristic al organismului respectiv, iar un organism este cu atât mai evoluat cu cât codul său caracteristic are o valoare mai mare.

Date fiind codurile caracteristice ale două organisme și diferențe, scrieți un program care să determine codul caracteristic al celui mai evoluat strămoș comun al lor.

Date de intrare

Fișierul de intrare COD.IN conține

n – codul caracteristic al primului organism

m – codul caracteristic al celui de-al doilea organism

Date de ieșire

Fișierul de ieșire COD.OUT conține pe prima linie:

p – codul celui mai evoluat strămoș comun al lui n și m

Exemplu

COD.IN

7145

847835

COD.OUT

75

Timp maxim de execuție: 1 secundă/test.

(OJII, 2002, clasa a X-a)

14. Virus vocalic

Un virus cu gusturi vocalice este obișnuit să atace fișierele text. El acționează asupra unei singure vocale din întregul text. Vocala începe să alunecă prin text lăsând pe locul ei un caracter **. Alunecarea se face pe verticală, vocala atacând la rândul ei vocalele de pe rândurile inferioare astfel: orice vocală de pe linia următoare aflată în imediata ei vecinătate (sub ea sau lateral-stânga și lateral-dreapta sub ea) sunt împinsă în jos fiind antrenate în cădere. Toate celelalte caractere, în afară de spațiu, sunt imune la virus.

De exemplu în textul:

BRAVO

AERAT

BRAGA

un atac asupra vocaliei A din cuvântul BRÀVO antrenează în cădere vocalele E și A din linia a doua, cele trei vocale antrenând ulterior vocalele A și A din linia a treia. Astfel, la sfârșitul procesului, virusul se va hrăni cu vocalele A E A A A din întregul text. Fișierul rămas arată astfel:

BR*VO

A*R*T

DR*G*

Există însă un caracter care poate împiedica alunecarea vocalelor: spațiu. O vocală care trece peste un spațiu rămâne agățată, fixându-se în locul acestuia și transformându-se într-un caracter !!.

De exemplu în textul:

ALBINA

INAINTE

OM DUCE

un atac asupra vocaliei I din prima linie antrenează vocalele A și I, vocala A rămnând fixată pe spațiu din linia a treia de unde este dislocată doar vocala U. Textul rămas arată astfel:

ALB*NA

IN**NTE

OM.D*CE

Portia totală consumată de virus cuprinde 3 vocali (I I U).

Scripteți un program care să determine poziția vocaliei ce va fi atacată pentru a obține un număr maxim de vocali mâncate de virusul vocalofag.

Fișierul de intrare VIRUS.IN este format din linii cu cel mult 200 de caractere pe fiecare linie, toate liniile considerându-se de aceeași lungime (eventualele liniilor mai scurte vor fi completate cu spații).

Fișierul text de ieșire VIRUS.OUT va conține linia și poziția din linie, pentru vocala atacată, precum și numărul total de vocale înghițite de virus.

(Tabără de pregătire, Sibiu, 1996)

15. Paragrafare optimală

Se dau două numere naturale nenule N și M, precum și o secvență de N cuvinte, câte un cuvânt pe o linie. Se cere să se formeze o pagină de text din aceste cuvinte în felul următor:

- fiecare linie a textului să fie de cel mult M caractere;
- ordinea cuvintelor rămâne cea din secvență inițială;
- două cuvinte vor fi separate prin exact câte un spațiu;
- pe fiecare rând vor fi puse un număr întreg de cuvinte.

Se notează cu s_1, s_2, \dots, s_p numărul de spații rămase libere la sfârșitul fiecărui rând, cu excepția ultimului. Formatarea cerută se va realiza astfel încât suma $s_1^3 + s_2^3 + \dots + s_p^3$ să fie minimă.

Exemplu

Intrare

17 15
A
fost
odata
ca-n
povesti
a
fost
'ca
niciodata
din
rude
mari
imparatesti
o
prea
frumoasa
fata

Ieșire

A fost odata
ca-n povesti
a fost ca
niciodata
din rude mari
imparatesti o
prea frumoasa
fata

16. Remi

Se consideră n grămezi de piese de remi, fiecare grămadă conținând un număr dat de piese. Grămezile sunt așezate pe masă în linie, una lângă cealaltă, cu spații între ele. Toate piesele trebuie așezate în k grupe egale, numărul total al pieselor permățând acest lucru.

Însă regula jocului spune că o grupă va putea fi formată dintr-o grămadă inițială sau prin punerea la un loc a două sau mai multe grămezi vecine (succesive). Cum procedeul nu permite mutarea unor piese dintr-o grămadă în alta, este posibil să nu se poată obține același număr de piese în fiecare grupă, la sfârșit calculându-se o penalizare în felul următor:

- pentru fiecare grupă penalizarea este dată de numărul de piese suplimentare sau numărul de piese lipsă din grupă respectiv față de numărul de piese dorit.

- penalizarea totală este suma penalizărilor pentru cele k grupe construite.

Dé exemplu, să considerăm $n=7$ grămezi cu 12, 9, 2, 11, 15, 5 și respectiv 2 pietre și dacă se cere realizarea a $k=4$ grupe, este de dorit să obținem 14 piese în fiecare grupă. Unind grămezile 1, 2 și 3 și grămezile 6 și 7 se obțin grupele de 23, 11, 15 și respectiv 7 piese. Penalizările vor fi 9 (23-14), 3 (11-14), 1 (15-14) și 7 (14-7), iar penalizarea totală: $9+3+1+7=20$.

Scrieți un program care să determine o modalitate de grupare a grămezilor astfel încât penalizarea totală să fie minimă.

(Baraj București, 1997)

Restriții

- $1 \leq k \leq n \leq 50$
- Numărul de piese din fiecare grămadă $0 \leq nr[i] \leq 255, \forall i \in \{1, 2, \dots, n\}$
- Suma $nr[1] + nr[2] + \dots + nr[n]$ este divizibilă cu k.

Exemplu

Pentru datele de intrare din exemplul precedent, programul va afișa:

1-2 3-4 5 6-7

Penalizare totală: 16

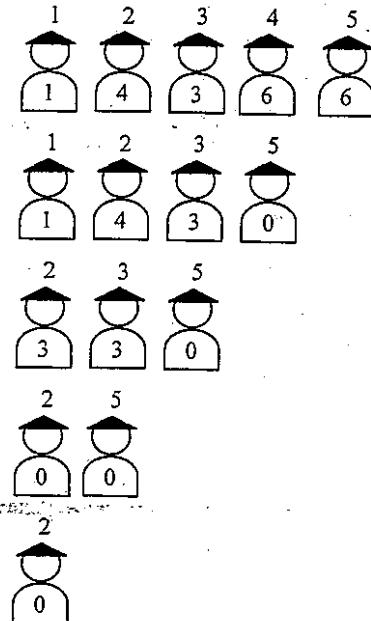
17. Nunta

În fața palatului Printesei Mofturoase se află N peștori așezăți la coadă, unul în spatele celuilalt. Fiecare poartă sub mantie un număr de pietre prețioase pe care dorește să le ofere prințesei ca dar de nuntă. Pentru a nu semăna vrajba în rândurile lor, prințesa a decis să-i determine ca $N-1$ dintre ei să renunțe în chip pașnic, peștorul rămas devinând alesul prințesei (indiferent de numărul de pietre prețioase deținute de acesta).

Doi peștori vecini la coadă se pot înțelege între ei astfel: cel care are mai puține pietre prețioase pleacă de la coadă primind de la celălalt un număr de pietre astfel încât să plece acasă cu un număr dublu de pietre față de câte avea. Dacă doi peștori au același număr de pietre, unul din ei (nu contează care) pleacă luând toate pietrele vecinului său.

Un peștor se poate înțelege la un moment dat cu unul singur dintre cei doi vecini ai săi. După plecarea unui peștor, toți cei din spatele lui avanseză.

De exemplu: pentru configurația alăturată de 5 peștori, un sir posibil de negocieri care conduc la reducerea cozii la un singur peștor



este: se înțeleg vecinii 4 cu 5 și pleacă 4, se înțeleg apoi 1 cu 2 și pleacă 1, se înțeleg apoi 3 cu 2 și pleacă 3, se înțeleg 2 cu 5 și pleacă 5. Astfel, peștorul 2 câștigă mâna preafrumoasei prințese, oferindu-i 0 pietre prețioase ca dar de nuntă.

Fie P numărul de pietre prețioase pe care le are peștorul care va deveni alesul prințesei. Se cer valorile distincte ale lui P la care se poate ajunge prin toate succesiunile de negocieri posibile.

Fișierul de intrare NUNTA.IN conține:

- pe prima linie numărul de peștori: n ($1 \leq n \leq 50$);
- pe a doua linie, n numere naturale din intervalul $[0, 20]$, reprezentând numărul de pietre prețioase pe care le dețin peștorii, în ordinea în care stau la coadă.

Fișierul de ieșire NUNTA.OUT va conține:

- pe prima linie numărul m de valori distincte ce pot fi obținute;
- pe a doua linie cele m valori ordonate crescător, reprezentând valorile care se pot obține.

Exemplu

NUNTA.IN

4
1 4 2 6

Timp maxim de execuție: 1 secundă/test.

NUNTA.OUT

3
1 3 5

(OJI, 2002, clasele a XI-a – a XII-a)

18. Zero

Se consideră numai numerele de L cifre scrise într-o bază B ($2 \leq B < L \leq 20$), în care prima cifră (cea mai semnificativă) este diferită de 0. Dându-se două numere P și Q ($2 \leq P, Q \leq L-1$), se cere să scrieți un program care să determine:

- câte numere există cu cel mult P cifre 0 (zero) consecutive ($NrP0$);
- câte numere există cu cel puțin Q cifre 0 (zero) consecutive ($NrQ0$).

(Tabăra de pregătire, București, 2001)

Exemplu

Pentru $L=3$, $B=2$, $P=1$ și $Q=2$, programul va afișa $NrP0=3$ și $NrQ0=1$.

19. Relee

Fie date altitudinile a N puncte, situate în linie dreaptă de-a lungul axei OX, astfel încât ele corespund unor abscise naturale consecutive. Primul punct are abscisa 1. Din acest punct trebuie trimisă o rază laser în ultimul punct (cel de abscisă N). Raza se propagă doar în linie dreaptă. Pentru a „ocoli” punctele având altitudini care împiedică trecerea razei, în anumite puncte se monteză relee care schimbă unghiul sub care se propagă raza, cu scopul ca ea să poată trece de vârfurile care se află în drumul ei. Releele se vor monta în oricare dintre punctele date, mai puțin în primul punct, de unde rază poate porni sub orice unghi și în ultimul punct unde rază poate fi recepționată sub orice unghi. S-a observat că dacă în anumite puncte releul se montează pe un pilon, numărul releelor necesare se poate micșora. Toți pilonii care se vor monta au aceeași înălțime dată H .

Determinați numărul minim de relee pentru ca raza să ajungă din punctul inițial în cel final, precum și punctele în care acestea se vor monta. Dacă există mai multe soluții cu același număr minim de relee, se va alege cea cu număr minim de piloni.

(ONI, Bacău, 2001, clasele a XI-a – a XII-a)

Date de intrare

Fișier de intrare: RELEE.IN

- | | | | |
|-------|-------|---|--|
| N | H | – numărul punctelor (N), respectiv înălțimea pilonilor (H); | |
| A_1 | A_2 | ... A_N | – altitudinile (înălțimile) punctelor. |

Date de ieșire

Fișier de ieșire RELEE.OUT:

- | | |
|---------------------------------------|--|
| N_{relee} | – numărul releeelor cе se vor monta, fără să fie înălțate pe piloni; |
| N_{piloni} | – numărul releeelor ce se vor monta, înălțate pe piloni; |
| $C_1 C_2 \dots C_{N_{\text{relee}}}$ | – numărul de ordine al punctelor unde se vor monta relee, fără piloni; |
| $D_1 D_2 \dots D_{N_{\text{piloni}}}$ | – numărul de ordine al punctelor unde releele se vor monta pe piloni. |

Restricții

- $1 \leq N \leq 200$
- $1 \leq H \leq 500$
- $1 \leq A_i \leq 2500 \quad \forall i \in \{1, 2, \dots, N\}$
- dacă 3 vârfuri sunt coliniare, atunci pe cel din mijloc nu trebuie amplasat releu.

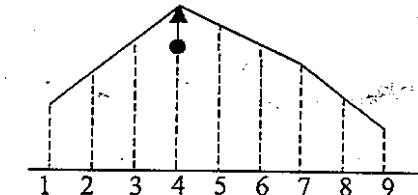
Exemplu

RELEE.IN

9 2
3 2 6 6 4 3 5 3 2

RELEE.OUT

1
1
7
4



Liniile punctate reprezintă altitudinea punctelor date. Releu fără pilon se va monta în punctul 7, iar pe pilon în punctul 4.

20. Job

Costel a găsit un job interesant și bine plătit. În fiecare dimineață, începând de la ora 5.00, el trebuie să stingă toate becurile din satul său. Becurile sunt plasate pe uliță principală, toate pe aceeași parte, astfel încât Costel le-a putut numerota începând cu 1, în ordine, de la intrarea în sat până la ieșire. Costel pleacă la ora 5.00 de lângă unul dintre becuri (cel din apropierea casei sale) și se deplasează cu viteză constantă de 1 m/s. În trecere el poate stinge becurile, astfel încât stingerea unui bec nu necesită un timp suplimentar.

Becurile au puteri diferite și deoarece Costel are o conștiință civică puternică, el intenționează să stingă becurile astfel încât consumul total de energie să fie minim.

Scrieți un program care să determine energia totală minimă consumată de becuri, până când Costel reușește să le stingă pe toate.

Date de intrare

Fișierul de intrare JOB.IN conține:

JOB.IN	Semnificație
N	N – numărul de becuri;
V	V – numărul becului de lângă care pornește Costel;
D ₁ W ₁	D ₁ – distanța de la intrarea în sat până la becul i exprimată în metri;
D ₂ W ₂	D ₂ – puterea becului i (cantitatea de energie consumată într-o secundă);
...	
D _N W _N	

Date de ieșire

Fișierul de ieșire JOB.OUT conține pe prima linie cantitatea minimă de energie consumată: cmin.

Restricții

- N număr natural, $2 \leq N \leq 100$
- $1 \leq V \leq N$
- D_i, W_i sunt numere naturale $0 \leq D_i \leq 1000$, $0 \leq W_i \leq 1000$
- cmin ≤ 1000000000
- Timp maxim de execuție: 1 secundă/test.

Exemple

JOB.IN	JOB.OUT	JOB.IN	JOB.OUT
3	65	4	56
2		3	
1 4		2 2	
6 5		5 8	
9 7		6 1	
		8 7	

(Baraj, Focșani, 2002)

21. Vânători

Cristi și Marius s-au hotărât într-o zi să se ducă la vânătoare. Cristi are o pușcă destul de prăpădită cu care poate vâna doar lupi. Marius, în schimb, are o pușcă performantă cu care poate vâna, în afară de lupi, și mistreți. Ajunși în pădure, ei își dau seama că vânătoarea nu e treabă ușoară; după fiecare animal ei trebuie să alerge mult. Având la dispoziție un număr de doar T minute alocate pentru vânătoare și dorind să vâneze în acest timp cât mai multe animale (lupi și mistreți) ei au calculat, pe baza unor informații date de pădură, câte minute trebuie să alerge, după fiecare animal în parte. Scrieți un program care determină câte animale trebuie să vâneze Cristi și Marius pentru ca împreună să aducă acasă un număr maxim de animale.

(Baraj, Constanța, 2000)

Date de intrare

Prima linie a fișierului de intrare VANATORI.IN conține numărul T, reprezentând durata maximă a vânătoriei. Pe cea de-a doua linie sunt scrise numerele L și M, reprezentând numărul de lupi respectiv de mistreți, despărțite prin spațiu. Pe cea de-a treia linie se găsesc L numere întregi, fiecare număr t_i (i=1, L) reprezentând timpul în care poate fi vânat fiecare dintre cei L lupi. Cea de-a patra linie conține M numere întregi, fiecare număr u_i (i=1, M) reprezentând timpul în care poate fi vânat fiecare dintre cei M mistreți.

Date de ieșire

În fișierul de ieșire VANATORI.OUT se va scrie un singur număr, reprezentând numărul maxim de animale pe care le pot vâna Marius și Cristi.

Restricții

- $1 \leq T \leq 300$
- $0 \leq L \leq 600$
- $0 \leq M \leq 600$
- $0 \leq t_i, u_i \leq T$

Exemplu

VANATORI.IN	VANATORI.OUT
5	5
4 2	
2 1 1 2	
5 4	

Explicație

Cristi vânează primul, al doilea și al patrulea lup, iar Marius al treilea lup și al doilea mistret.

Algoritmi pe grafuri

1. Secvență grafică

Fie $n-1 \geq d_1 \geq d_2 \geq \dots \geq d_n \geq 0$ o secvență descrescătoare de n numere naturale. Scrieți un program care să verifice dacă această secvență reprezintă secvență gradelor vârfurilor unui graf (o astfel de secvență se numește secvență grafică). În caz afirmativ, să se determine un astfel de graf.

De exemplu, pentru $n=7$ și secvența gradelor $D=(5, 4, 3, 2, 2, 1, 1)$ programul va afișa răspunsul DA. Un graf posibil are muchiile $[1, 2], [1, 3], [1, 4], [1, 5], [1, 6], [2, 3], [2, 4], [2, 5], [3, 7]$.

Pentru $n=7$ și secvența gradelor $D=(5, 4, 2, 2, 1, 1, 0)$, programul va afișa răspunsul NU.

Soluție

Vom demonstra următoarea afirmație:

Secvența de numere naturale nenule $D = (d_1 \geq d_2 \geq \dots \geq d_n) \geq 0$ este secvență grafică dacă și numai dacă secvența $D' = (d_2-1, d_3-1, \dots, d_{n-1}-1, d_n-2, \dots, d_1)$ este secvență grafică și $d_1 \geq n-1$.

Suficiența

Presupunând că D' este secvență grafică, deducem că există un graf cu secvență gradelor D' . Introducem în acest graf un nou vârf pe care îl vom uni prin muchii de vârfurile $2, 3, \dots, d_1+1$. Obținem astfel un graf cu secvență gradelor D .

Necesitatea

Să considerăm că D este secvență grafică, prin urmare există cel puțin un graf cu secvență gradelor D . Să considerăm că G este un astfel de graf și anume cel pentru care suma gradelor vârfurilor adiacente cu vârful 1 este maximă. Să demonstrează că în graful G vârful 1 este adjacents chiar cu vârfurile $2, 3, \dots, d_1+1$ (ca urmare putem să eliminăm din G vârful 1 și, evident, toate muchiile incidente cu acesta și să obținem un subgraf G' cu $n-1$ vârfuri și secvență gradelor D').

Să presupunem că există un vârf k ($2 \leq k \leq d_1+1$) astfel încât vârful k nu este adjacents cu vârful 1. Rezultă că există un vârf j ($j > d_1+1$) astfel încât vârfurile 1 și j sunt adjacente. Dacă gradul $d_k = d_j$, putem interschimba vârfurile j și k , fără a altera ordinea din secvență gradelor. Dacă $d_k \neq d_j$, cum $j > k$ deducem că $d_j < d_k$. Dar acest lucru contrazice modul de alegere a grafului G (înlocuind vârful j cu vârful k , am obține un graf în care suma gradelor vârfurilor adjacente cu vârful 1 este mai mare).

Demonstrația acestei propoziții conduce și la un algoritm constructiv de determinare a unui graf cu secvență gradelor dată: la fiecare pas vom uni prin muchii vârful curent i cu fiecare din cele d_i vârfuri care urmează și vom decrementa gradele acestora; după fiecare

pas trebuie să reordonăm vârfurile după grade. Algoritmul se termină fie când am epuizat vârfurile (după $n-1$ pași), fie când găsim un vârf i pe care nu îl putem uni cu cele d_i următoare (am obținut un grad negativ), caz în care vom concluziona că D nu este secvență grafică.

```
ofstream fout("graf.out");
fout<<"DA"<<endl;
for (int k=1; k<n && d[k]; k++)
    {for (int i=1; i<=d[k]; i++)
        { d[k+i]--; //decrementează gradul
        fout<<v[k]<< ' '<<v[i+k]<<endl; //scriu muchia
        if (d[k+i]<0)//imposibil! sterg fisierul, scriu NU
            {fout.close();
            fout.open("graf.out", ios::out);
            fout<<"NU";
            fout.close(); return;}
        }
    sortare(); //reordonează vârfurile după grad
}
fout.close();
```

Observați că fiind necesară reordonarea vârfurilor după grad este necesar să utilizăm un vector suplimentar v , care să conțină vârfurile în ordinea crescătoare a gradelor.

2. Determinarea celui mai scurt drum între două vârfuri

Fie G un digraf și x, y două vârfuri din G . Să se determine cel mai scurt drum de la x la y . Considerăm că lungimea unui drum este egală cu numărul de arce din care este constituit drumul.

Soluție

O proprietate importantă a parcurgerii în lățime a unui graf constă în faptul că fiecare vârf este atins pe cel mai scurt drum care pleacă din vârful de start. Prin urmare, pentru a determina cel mai scurt drum de la x la y vom efectua o parcurgere în lățime începând din x , până când atingem vârful y , sau până când vizităm toate vârfurile accesibile din x .

Reprezentarea informațiilor

Vom reprezenta graful prin liste de adjacență. Mai exact vom reține pentru fiecare vârf lista sa de adjacență, reprezentată ca un vector, în care pe poziția 0 reținem gradul exterior al vârfului x , iar pe următoarele poziții reținem vârfurile y pentru care există în graf arcul (x, y) .

```
#define NMaxVf 101
typedef int ListaAd[NMaxVf];
typedef ListaAd Graf[NMaxVf];
Graf G;
int n, x, y;
void citire()
```

```

{ ifstream f("graf.in");
f>>n>>x>>y;
for (int i=1; i<=n; i++)
{ f>>G[i][0];
  for (int j=1; j<=G[i][0]; j++) f>>G[i][j];
f.close(); }

```

Pentru a reține mulțimea vârfurilor accesibile din x și pentru a reconstituи cel mai scurt drum de la x la y , vom utiliza un vector $viz[]$, cu semnificația:

```

viz[i]=0, dacă  $i$  nu este accesibil din  $x$ 
       $j$ , dacă vârful  $i$  a fost vizitat prin parcurgerea arcului  $(j,i)$ 
      -1, dacă  $i=x$ .

```

Pentru a efectua o parcurgere în lățime, vom utiliza o coadă C , implementată ca un vector:

```

int viz[NMaxVf];
void BFS(int x)
{
int c[NMaxVf], p=0, u=0;
for (c[p]=x, viz[x]=-1; p<=u & !viz[y]; )
{ x=c[p++];
  for (int i=1; i<=G[x][0]; i++)
    if (!viz[G[x][i]])
      {c[++u]=G[x][i]; viz[G[x][i]]=x;}
}
}

```

Pentru a reconstituи cel mai scurt drum de la x la y vom utiliza o funcție denumită $Scrie()$, pe care o vom apela în funcția $main()$ astfel:

```

if (!viz[y])
  cout<<"Nu există drum de la "<<x<<" la "<<y<<endl;
else
  Scrie(y);

```

Funcția $Scrie()$ va afișa drumul de la x la y , în mod recursiv:

```

void Scrie(int y)
{ if (y!= -1)
  {Scrie(viz[y]);
  cout<<y<<' ';}
}

```

Observație

Evident, algoritmul funcționează și pentru determinarea celui mai scurt lanț între două vârfuri, pentru grafuri neorientate.

Exercițiu

Implementați acest algoritm pentru cazul în care liste de adiacență sunt reprezentate prin liste simplu înlățuită alocate dinamic.

3. Descompunerea unui graf în componente conexe

Fie G un graf neorientat. Să se descompună graful în componente conexe.

Soluție

O primă idee ar fi să utilizăm matricea lanțurilor (o matrice G cu n linii și n coloane, cu semnificația $G[i][j]=1$, dacă există lanț de la i la j și 0, altfel; evident că n am notat numărul de vârfuri din graf).

Vom determina matricea lanțurilor plecând de la reprezentarea grafului prin matrice de adiacență, prin algoritmul Roy-Warshall:

```

void Roy_Warshall()
{int i, j, k;
for (k=1; k<=n; k++)
  for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
      if (!G[i][j]) G[i][j]=G[i][k]*G[k][j];
}

```

După determinarea matricei lanțurilor, observăm că i și j sunt în aceeași componentă conexă dacă și numai dacă $G[i][j]=1$:

```

int viz[NMaxVf];
for (int i=1, nrc=0; i<=n; i++)
  if (!viz[i])
    {cout<<"Componenta conexă "<<++nrc<<endl;
     for (int j=1; j<=n; j++)
       if (G[i][j])
         {cout<<j<<' '; viz[j]=1;}
       if (!viz[i]) {cout<<i; viz[i]=1;}
     cout<<endl;}
}

```

Observații

- Putem adapta acest algoritm și pentru determinarea componentelor conexe ale unui digraf.
- Datorită determinării matricei lanțurilor, complexitatea acestui algoritm de descompunere în componente conexe este de $O(n^3)$.

O idee mai eficientă ar fi să utilizăm parcurgeri de grafuri. Vom reprezenta graful prin liste de adiacență și vom realiza o parcurgere (în adâncime sau în lățime) pentru fiecare componentă conexă. În funcția de parcurgere afișăm vârfurile vizitate.

```

for (int x=1, nrc=0; x<=n; x++)
  if (!viz[x])
    {cout<<"Componenta conexă "<<++nrc<<endl;
     Parcurg(x);
     cout<<endl;}
}

```

Observații

În acest caz complexitatea algoritmului de descompunere în componente conexe este de $O(n+m)$, unde n este numărul de vârfuri, iar m numărul de muchii din graf.

În cazul în care graful este reprezentat prin lista muchiilor, nici unul din algoritmii precedenți nu este eficient.

```
#define NMaxVf 101
#define NMaxM (NMaxVf*(NMaxVf-1)/2)
struct Muchie {int x, y;};
typedef Muchie Graf[NMaxM];
Graf G;
int n, m;
```

În acest caz vom construi componentele conexe ale grafului analizând succesiv muchiile grafului. Vom reține evidența componentelor conexe formate într-un vector $c[]$, cu semnificația $c[i] =$ reprezentantul componentei conexe din care face parte vârful i . Inițial, când nu am considerat nici o muchie, toate vâfurile sunt izolate, deci formează o componentă conexă distinctă:

```
for (i=1; i<=n; i++) c[i]=i;
```

Pentru fiecare nouă muchie introdusă în graf, dacă extremitățile muchiei nu aparțin deja aceleiași componente conexe, unific componentele conexe corespunzătoare extremităților muchiei nou introduse:

```
for (i=0; i<m; i++)
    if (c[G[i].x]!=c[G[i].y])
        /*unific componentele conexe corespunzătoare
         extremităților muchiei i */
        nou=c[G[i].x];  vechi=c[G[i].y];
        for (j=1; j<=n; j++)
            if (c[j]==vechi) c[j]=nou;
```

Pentru a afișa componentele conexe, parcurg vectorul $c[]$:

```
for (i=1; i<=n; i++)
    if (c[i])
        { cout<<"Componenta conexă "<<+nrc<<endl;
          for (j=i+1; j<=n; j++)
              if (c[j]==c[i]) {cout<<j<<' '; c[j]=0;}
          cout<<i<<endl; }
```

Observație

Pentru grafuri reprezentate prin lista muchiilor, acest algoritm de descompunere în componente conexe are complexitatea $O(n*m)$.

4. Descompunerea unui digraf în componente tare conexe

Fie G un digraf. Să se descompună digraful în componente tare conexe.

Soluție

O primă idee ar fi să construim matricea G a drumurilor grafului prin algoritmul *Roy-Warshall*, apoi să determinăm componentele tare conexe observând că x și y aparțin aceleiași componente tare conexe dacă și numai dacă $G[x][y]=G[y][x]=1$. Acest algoritm ar avea complexitatea $O(n^3)$, datorită determinării matricei drumurilor.

O altă idee ar fi să utilizăm parcurgeri. Pentru a determina componenta tare conexă a vârfului x vom realiza mai întâi o parcuregere începând din x . Astfel determinăm multimea vâfurilor accesibile din x . Pentru a determina multimea vâfurilor din care se poate ajunge în x vom utiliza graful transpus (graful în care toate sensurile arcelor sunt inversate). Realizând o parcuregere din x în graful transpus, obținem multimea tuturor vâfurilor y , astfel încât există drum de la y la x . Intersecția celor două multimi determinate prin parcurgeri reprezintă componenta tare conexă a vârfului x . Evident, procedeul se repetă pentru fiecare componentă tare conexă.

5. Transformarea unui graf în graf conex

Fie G un graf neconex. Scrieți un program care să adauge un număr minim de muchii astfel încât graful să devină conex.

Soluție

În primul rând se determină componentele conexe ale grafului C_1, C_2, \dots, C_p . Numărul minim de muchii necesare pentru ca graful să devină conex este egal cu $p-1$ (ne imaginăm că fiecare componentă conexă reprezintă un nod într-un graf denumit graf condensat; numărul minim de muchii dintr-un graf conex este egal cu numărul de vârfuri din graf minus 1). Vom alege căte un reprezentant pentru fiecare componentă conexă și vom adăuga $p-1$ muchii în graf unind succesiv reprezentantul componentei conexe C_1 cu fiecare din reprezentanții componenteelor conexe C_2, C_3, \dots, C_p .

Vom aplica algoritmul de descompunere în componente conexe în urma căruia reținem componentele conexe într-un vector c , cu semnificația $c[i] =$ reprezentantul componentei conexe din care face parte vârful i . Muchiile adăugate se vor afișa astfel:

```
for (i=2; i<=n; i++)
    cc=c[i];
    if (cc && cc!=c[1])
        /*componentă conexă a vârfului i nu a fost unită încă
         printr-o muchie de componentă conexă a vârfului 1 */
        {cout<<c[1]<<' '<<cc<<endl; //scriu muchia
         for (j=2; j<=n; j++)
             if (c[j]==cc) c[j]=0;
         /*marchez cu 0 în vectorul c vâfurile ale caror
          componente conexe au fost deja unite printr-o
          muchie de componentă conexă a vârfului 1 */
        }
    }
```

6. Organizație profitabilă

Un personaj inventiv, pe care îl vom denumi codificat 1, a înființat o nouă organizație profitabilă. Regulile pe care se bazează funcționarea organizației sunt:

1. Orice persoană intră în organizație cu un fond inițial.
2. Orice persoană, după intrarea în organizație, are dreptul de a introduce în organizație noi membri. Aceștia vor fi subordonări direcții ai persoanei care i-a introdus în organizație.
3. Banii organizației vor fi investiți în afaceri profitabile.

4. La sfârșitul anului se calculează venitul fiecărui membru. Venitul unei persoane din organizație este egal cu fondul său inițial plus suma fondurilor tuturor subordonaților săi.

Scrieți un program care să calculeze venitul fiecărei persoane din organizație la sfârșitul anului.

Date de intrare

Din fișierul de intrare ORG.IN se citesc:

n – numărul total de persoane din organizație
 $k_1 \ i_{11} \ i_{12} \dots i_{1k_1}$ – numărul de subordonați direcți și subordonați direcți ai persoanei 1
 $k_2 \ i_{21} \ i_{22} \dots i_{2k_2}$ – numărul de subordonați direcți și subordonați direcți ai persoanei 2
 \dots
 $k_n \ i_{n1} \ i_{n2} \dots i_{nk_n}$ – numărul de subordonați direcți și subordonați direcți ai persoanei n
 $f_1 \ f_2 \dots f_n$ – fondurile inițiale ale persoanelor din organizație.

Date de ieșire

În fișierul de ieșire ORG.OUT se afișează în ordine venitul fiecărei persoane din organizație, căte o valoare pe o linie..

Restriții

$$1 \leq n \leq 1000$$

$$0 \leq k_i \leq 25, \forall i \in \{1, 2, \dots, n\}$$

$$f_i \in N, 0 < f_i \leq 1000, \forall i \in \{1, 2, \dots, n\}$$

Exemplu

ORG.IN	ORG.OUT
6	85
2 2 3	15
1 5	60
2 4 6	20
0	10
0	30
10 5 10 20 10 30	

Soluție

Structura organizației este un digraf aciclic, pe care îl vom reprezenta prin liste de adiacență:

```

#define NMaxVf 1001
#define NMaxM 26
typedef unsigned ListaAd[NMaxM];
typedef ListaAd Graf[NMaxVf];
Graf G;
unsigned int n;
unsigned long f[NMaxVf];
  
```

Pentru a calcula venitul fiecărei persoane la sfârșitul anului este suficient să realizăm o parcurgere în adâncime începând cu vârful 1, rădăcina acestei arborescențe:

```

void DFS(unsigned i)
{
  for (unsigned j=1; j<=G[i][0]; j++)
    {DFS(G[i][j]);
     f[i]+=f[G[i][j]];}
}
  
```

Exercițiu

Descrieți un algoritm iterativ de rezolvare a acestei probleme, bazat pe o parcurgere pe niveluri.

7. Trafic

Un sistem de transport public al orașului constă din m linii. În oraș există n stații, pe care le considerăm numerotate de la 1 la n . Fiecare linie constă dintr-o succesiune de stații și poate fi parcursă în ambele sensuri. Printr-o stație pot trece, eventual, mai multe linii. Scrieți un program care să determine numărul minim de schimbări de pe o linie pe alta necesar pentru a ajunge de la o stație i la o stație j (i, j date). O schimbare de linie este posibilă numai într-o stație prin care trec ambele linii.

Date de intrare

Datele de intrare se citesc din fișierul TRAFIC.IN cu următoarea structură:

n m i j	
$s_{11} \ s_{12} \dots s_{1k_1}$	– stațiile de pe linia 1
$s_{21} \ s_{22} \dots s_{2k_2}$	– stațiile de pe linia 2
\dots	
$s_{m1} \ s_{m2} \dots s_{mk_m}$	– stațiile de pe linia m

Date de ieșire

Fișierul de ieșire se numește TRAFIC.OUT și conține o singură linie cu numărul minim de schimbări necesar pentru a ajunge din stația i în stația j sau mesajul IMPOSSIBIL în cazul în care din stația i nu se poate ajunge în stația j prin astfel de schimbări de linii.

Restriții

$$n, m \text{ numere naturale, } n, m \leq 255.$$

Exemplu

TRAFIC.IN	TRAFIC.OUT
11 3 2 10	2
1 2 3 4	
5 3 6 7 8	
11 10 7 9	

Soluție

Putem asocia problemei un graf neorientat astfel:

- vârfurile grafului corespund celor m liniilor;
- dacă intersecția dintre mulțimea stațiilor de pe linia x și mulțimea stațiilor de pe linia y este nevidă, atunci în graful asociat există muchie de la vârful x la vârful y.

Pentru a ajunge din stația i în stația j cu număr minim de schimbări de linii trebuie să determinăm lanțurile de lungime minimă de la fiecare vârf din graf care corespunde unei liniilor căreia îl aparține stația i către vârfurile din graf care corespund liniilor ce conțin stația j. Dintre toate lanțurile determinate îl vom alege pe cel de lungime minimă.

Reprezentarea informațiilor

Vom reține o linie ca mulțimea stațiilor sale, reprezentată prin vector caracteristic. Datorită dimensiunii relativ mari a datelor de intrare, este necesar să implementăm vectorul caracteristic pe biți.

```
#define Max 256
#define Lg (Max/8)
typedef int Linie[Lg];
Linie T[Max];
int n, m, min, i, j, masca_i, masca_j;
```

Variabilele masca_i și masca_j sunt două variabile auxiliare, utile pentru a testa valoarea bitului corespondător stației i, respectiv al stației j:

```
masca_i = 1<<i%8; masca_j=1<<j%8;
```

În funcția main() vom apela funcția BFS() din fiecare linie care conține stația i.

```
int min=Max, nrsch;
for (int k=0; k<m; k++)
    if (T[k][i/8] & masca_i) //stația i este pe linia k
        nrsch=BFS(k);
    if (min>nrsch) min=nrsch;
```

În apelul BFS(k) determinăm numărul minim de schimbări de linii, necesare de la linia k la o linie care conține stația finală j. Evident, numărul minim absolut de schimbări de linii necesare pentru a ajunge de la stația i la stația j se obține alegând minimul dintre toate valorile calculate de apelurile funcției BFS() din main().

```
int BFS (int k)
{
    //parcure graful BFS incepand din k
    int viz[Max]; //marchez varfurile vizitate
    int C[Max]; //coada
    int ns[Max]; //retin numarul de schimbari de linii
    for (int p=0; p<m; p++) {viz[p]=0; ns[p]=Max;}
    int ic, sc;
    ic=sc=0; C[ic]=k; viz[k]=1; ns[k]=0;
    while (ic<=sc)
        {k=C[ic++];
         for (p=0; p<m; p++)
```

```
        if (există(k,p) && !viz[p])
            {ns[p]=ns[k]+1;
             viz[p]=1; C[++sc]=p;}
        }
    int nr=Max; //in nr determin minimul din vectorul ns
    for (p=0; p<m; p++)
        if (T[p][j/8] & masca_j) //stația j este pe linia p
            if (nr>ns[p]) nr=ns[p];
    return nr; }
```

Funcția există(k, p) utilizată în funcția BFS() testează dacă pot schimba linia de la k la p. Mai exact dacă intersecția dintre mulțimea stațiilor de pe linia k și mulțimea stațiilor de pe linia p este nevidă:

```
int există (int x, int y)
{for (int i=0; i<Lg; i++)
    if (T[x][i]&T[y][i]) return 1;
return 0; }
```

În funcția Afisare() vom testa dacă am obținut o soluție și în caz afirmativ vom afișa numărul minim de schimbări determinat:

```
void Afisare()
{ofstream fout("trafic.out");
if (min==Max) fout<<"IMPOSSIBIL\n";
else fout<<min;
fout.close();}
```

Exercițiu

Putem asocia problemei un graf considerând mulțimea vârfurilor ca fiind mulțimea stațiilor; între două stații există muchie dacă și numai dacă cele două stații aparțin aceleiași linii. Rezolvăți problema utilizând această reprezentare!

8. Drumuri scurte

Numim *graf turneu* un graf orientat în care între oricare două vârfuri se află un arc și numai unul. Dat fiind un număr natural n (n≤250), să se determine un graf turneu cu n vârfuri astfel încât între oricare două noduri să existe un drum de lungime 1 sau 2.

Date de intrare

Fișierul de intrare GRAF.IN conține pe prima linie valoarea lui n.

Datele de ieșire

Fișierul GRAF.OUT va conține matricea de adiacență a grafului care îndeplinește cerințele problemei (pe linia i a fișierului se va afla linia i a matricei) în cazul în care există soluție sau textul *fara solutie* (cu litere mici!), dacă nu există nici o modalitate de dispunere a arcelor.

Exemplu

GRAF.IN

6

GRAF.OUT poate conține

```

0 1 1 1 0 0
0 0 1 1 1 0
0 0 0 1 0 1
0 0 0 0 1 1
1 0 1 0 0 1
1 1 0 0 0 0

```

Timp maxim de execuție: 3 secunde/test.

(ONI, Constanța, 2000, clasa a X-a)

Soluție

Să rezolvăm mai întâi problema (dacă este posibil), pentru niște cazuri particulare (le vom numi cazuri elementare).

Observăm în primul rând că pentru $n=2$ și $n=4$ nu există soluție. Prin urmare, cel mai mic număr natural par pentru care există soluție este $n=6$. O soluție posibilă conține arcele $(1,2)$, $(1,3)$, $(1,4)$, $(2,3)$, $(2,4)$, $(2,5)$, $(3,4)$, $(3,6)$, $(4,5)$, $(4,6)$, $(5,1)$, $(5,3)$, $(5,6)$, $(6,1)$, $(6,2)$.

Pentru $n=3$ o soluție este $(1,2)$, $(2,3)$, $(3,1)$.

Să demonstrăm acum prin inducție că problema are soluție pentru orice număr natural impar ≥ 3 și orice număr natural par ≥ 6 . Presupunem că afirmația este adevărată pentru $\forall k \leq n-1$. Demonstrăm că este adevărată și pentru n vârfuri.

Să considerăm un graf turneu cu $n-2$ vârfuri care are proprietatea din enunțul problemei și să adăugăm două vârfuri suplimentare (n și $n-1$). Vom dispune arcele incidente cu vârfurile n și $n-1$ astfel:

- dispunem $n-2$ arce de tipul (n,i) , $\forall i \in \{1,2,\dots,n-2\}$. Aceste arce au ca extremitate inițială vârful n și asigură un drum de lungime 1 de la n la primele $n-2$ vârfuri ale grafului.
- dispunem $n-2$ arce de tipul $(i,n-1)$, $\forall i \in \{1,2,\dots,n-2\}$. Aceste arce au ca extremitate finală vârful $n-1$ și asigură un drum de lungime 1 de la primele $n-2$ vârfuri ale grafului la vârful $n-1$.
- dispunem un arc de tipul $(n-1,n)$. Acest arc care are ca extremitate inițială vârful $n-1$ și ca extremitate finală vârful n . Prin urmare el asigură drumuri de lungime 2 de la $n-1$ la primele $n-2$ vârfuri ale grafului și drum de lungime 1 de la $n-1$ la n . De asemenea, el asigură drumuri de lungime 2 de la primele $n-2$ vârfuri ale grafului la vârful n . De la vârful n la vârful $n-1$ există un drum de lungime 2, de exemplu $(n,1)$, $(1,n-1)$.

Prin urmare putem genera recursiv graful turneu cerut astfel:

```

#define NMaxVf 251
unsigned n;
unsigned char a[NMaxVf][NMaxVf];
void Gen(unsigned n)
{if (n==3) Init3();

```

```

    else
        if (n==6) Init6();
        else
            {for (unsigned i=1; i<n-1; i++) a[n][i]=a[i][n-1]=1;
             a[n-1][n]=1;
             Gen(n-1); }
}

```

Funcția `Init3()` initializează cu 1 componentele matricei de adiacență corespunzătoare arcelor din exemplul pentru $n=3$, iar `Init6()` realizează aceeași operație pentru arcele din exemplul pentru $n=6$.

9. Descompunere pe niveluri

Fie un digraf fără circuite. Descompuneti graful pe niveluri astfel încât vârfurile situate pe același nivel să nu fie adjacente, iar dacă există arcul (x, y) , atunci vârful x să fie situat pe un nivel superior (cu număr mai mic) față de vârful y .

De exemplu, pentru $n=7$ vârfuri și arcele $(7,3)$, $(7,1)$, $(2,1)$, $(2,4)$, $(2,5)$, $(1,4)$, $(3,6)$, programul va afișa:

```

Nivelul nr. 1: 2 7
Nivelul nr. 2: 1 3 5
Nivelul nr. 3: 4 6

```

*Soluție**Reprezentarea informațiilor*

Reprezentăm graful prin liste de adiacență, deci pentru fiecare vârf din graf reținem lista succesorilor săi direcți.

De asemenea, vom utiliza un vector `np`, în care pentru fiecare vârf din graf reținem -1, dacă vârful a fost deja plasat pe un nivel sau numărul predecesorilor săi care nu au fost încă plasați pe niveluri.

```

#define NMaxVf 101
typedef int Lista[NMaxVf];
typedef Lista Graf[NMaxVf];
Graf G;
Lista np, nivel;
int n, nr_nivel, total;

```

La fiecare pas plasăm pe nivelul curent (în vectorul `Nivel`) toate vârfurile i care nu mai au predecesori neplasați pe niveluri ($np[i]=0$). Marcăm apoi vârfurile de pe nivelul curent cu -1 în vectorul `np` și decrementăm numărul predecesorilor neplasați pe niveluri ai succesorilor lor direcți. Algoritmul se repetă până când toate vârfurile grafului sunt plasate pe niveluri.

```

int i, j;
total=0; //numarul de varfuri plasate deja pe niveluri
while (total < n)
{nr_nivel=0; //numarul de varfuri de pe nivelul curent
 for (i=1; i<=n; i++)

```

```

if (!np[i])           //varful i nu mai are predecesori
{ nr_nivel++;
  //incrementam nr. de varfuri de pe nivelul curent
  nivel[nr_nivel]=i;
  //retinem varful i pe nivelul curent
  total++;}           //numaram varful i
afisare_nivel();    //afisam varfurile de pe nivelul curent
for (i=1; i<=nr_nivel; i++)
{ np[nivel[i]]=-1;
  //marcam varful i de pe nivelul curent
  for (j=1; j<=G[nivel[i]][0]; j++)
    np[G[nivel[i]][j]]--;
/*decrementam numarul predecesorilor succesorilor
varfului i de pe nivelul curent */
}

```

Exerciții

- La pasul următor singurele vârfuri al căror grad interior poate deveni 0 (deci nu mai au predecesori neplasați pe niveluri) sunt succesorii ai vârfurilor plasate pe nivelul curent. Optimizați algoritmul utilizând această observație.
- Modificați programul astfel încât în cazul în care digraful conține circuite să se afișeze un mesaj de eroare.

10. Problema traducătorului

Un traducător găsește o carte scrisă cu litere latine, în care însă ordinea literelor din alfabet este schimbată. La sfârșitul cărții se află un index de cuvinte necontradictoriu. Se cere să se determine o ordine posibilă a literelor din noul alfabet.

(ONI, Cluj, 1991)

Soluție

Se parcurge indexul și pentru fiecare pereche de cuvinte consecutive în index se determină perechea de litere care induce ordinea lor (de exemplu, dacă după qwerty urmează qedfg, deducem că în noul alfabet litera r precedă litera d). Obținem astfel o relație de ordine parțială între literele alfabetului.

Oricarei relații de ordine i se poate asocia un digraf antisimetru în care mulțimea vârfurilor este mulțimea elementelor pe care este definită relația de ordine (în cazul nostru mulțimea literelor din alfabetul latin), iar mulțimea arcelor corespunde perechilor de vârfuri (x, y) cu semnificația „x precedă pe y conform relației de ordine”.

În problema noastră, din faptul că indexul este necontradictoriu deducem că digraful asociat problemei nu conține circuite, deci că problema are soluție.

Asociind problemei un digraf în acest mod, am redus problema la o problemă asemănătoare cu descompunerea pe niveluri a unui digraf fără circuite: la fiecare pas vom selecta un vârf care nu a mai fost selectat și care nu are predecesori care nu au fost deja ordonați. Evident, după selectarea unui vârf, se decrementează numărul predecesorilor neordonati ai tuturor succesorilor săi.

Observație

Problema ordonării unor elemente între care există o relație de ordine parțială se numește o *sortare topologică*.

11. Comentarii

Elevii de clasa a XII-a au un sistem propriu de a-și transmite comentariile la română. În urma îndelungatei lor colaborări, și-au format un sistem de relații astfel încât oricare ar fi doi elevi unul primește comentarii de la celălalt. Evident, orice comentariu primit poate fi transmis mai departe.

Presupunând că în clasa a XII-a sunt N elevi, numerotați distinct de la 1 la N și că sistemul de relații dintre elevi este cunoscut, scrieți un program care să găsească o modalitate prin care un comentariu, transmis de unul dintre elevi să ajungă pe la toți elevii, o singură dată.

(OLI, Iași, 2002, clasa a XI-a)

Date de intrare

Fișierul de intrare COMENT.IN conține

N	– numărul de elevi
$x_1 \ y_1$	– elevul x_1 transmite comentarii elevului y_1
$x_2 \ y_2$	– elevul x_2 transmite comentarii elevului y_2
...	
$x_m \ y_m$	– elevul x_m transmite comentarii elevului y_m

Date de ieșire

Fișierul de ieșire COMENT.OUT conține pe o singură linie ordinea în care elevii primesc comentariul. Elevul z_1 este cel care transmite comentariul.

 $z_1 \ z_2 \dots z_N$ **Restricții și precizări**

- $2 \leq N \leq 100$
- $x_i, y_i \in \{1, \dots, N\}, \forall i \in \{1, 2, \dots, m\}$
- $m = N * (N - 1) / 2$
- Valorile scrise pe aceeași linie în fișierele de intrare și ieșire sunt separate prin spații.

Exemplu

COMENT.IN

4

1 2

1 4

3 1

2 4

3 2

4 3

COMENT.OUT poate conține

2 4 3 1

Timp maxim de execuție: 1 secundă/test.

Soluție

Problema solicită determinarea unui drum hamiltonian într-un graf turneu și admite întotdeauna soluție.

Vom reprezenta graful prin matrice de adiacență:

```
unsigned G[NMaxVf][NMaxVf], n;
void Citire()
{
    ifstream fin("coment.in");
    int x, y;
    fin >> n;
    int m = n * (n - 1) / 2;
    for (int i = 1; i <= m; i++)
        {fin >> x >> y;
         G[x][y] = 1;}
    fin.close();
}
```

Vom reprezenta drumul hamiltonian ca o listă simplu înlățuită, în care vom insera succesiv vârfurile grafului:

```
struct Nod {
    int v;
    Nod * urm;
};

typedef Nod* Lista;
Lista d; //inceputul listei care reprezinta drumul
Lista sf; //sf = sfarsitul listei care reprezinta drumul

Pentru a construi un drum hamiltonian vom începe cu un vârf oarecare al grafului (de exemplu vârful 1).

int vf = 1; //vârful curent
d = new Nod; d->v = 1; d->urm = NULL; uz[1] = 1;
```

Cât timp este posibil vom insera vârfuri în lista care reprezintă drumul înaintea primului nod (putem insera numai un vârf pentru care există arc către primul vârf din lista care reprezintă drumul):

```
while (vf)
    {vf = cautpred(vf);
     if (vf)
         {p = new Nod; p->v = vf; p->urm = d; d = p; uz[vf] = 1;
          }
    }
```

Funcția cautpred() determină un predecesor al vârfului specificat ca parametru care nu aparține deja drumului construit:

```
int cautpred(int vf)
{
    for (int i = 1; i <= n; i++)
        if (!uz[i] && G[i][vf]) return i;
    return 0;
}
```

Drumul nu mai poate fi prelungit în stânga, vom încerca să-l prelungim în dreapta. Prin urmare, cât timp este posibil vom insera vârfuri în lista care reprezintă drumul după ultimul nod (putem insera numai un vârf către care există arc de la ultimul vârf din lista care reprezintă drumul):

```
vf = 1;
while (vf)
    {vf = cautsucc(vf);
     if (vf)
         {p = new Nod; p->v = vf; p->urm = NULL;
          sf->urm = p; sf = p; uz[vf] = 1;
          }
    }
```

Evident, funcția cautsucc() determină un succesor al vârfului specificat ca parametru care nu aparține deja drumului construit:

```
int cautsucc(int vf)
{
    for (int i = 1; i <= n; i++)
        if (!uz[i] && G[vf][i]) return i;
    return 0;
}
```

Drumul nu mai poate fi prelungit nici în stânga, nici în dreapta. Este posibil totuși să mai existe vârfuri ale grafului care nu aparțin drumului construit. Fie vf un astfel de vârf. Vârful vf nu poate fi plasat la începutul drumului, deci există arc de la primul vârf către vf. Vârful vf nu poate fi plasat nici la sfârșitul drumului, deci există arc de la vf către ultimul vârf. Prin urmare, de-a lungul drumului trebuie să existe o „schimbare de sens”, adică două vârfuri consecutive (să le notăm x și y) astfel încât să existe arcul (x, vf) și arcul (vf, y). Prin urmare vârful vf poate fi inserat pe drum între vârfurile x și y:

```
for (vf = 2; vf <= n; vf++)
    if (!uz[vf]) //inserez vf pe pozitia corecta pe drum
        {pq = d; q = d->urm;
         while (G[q->v][vf]) {pq = q; q = q->urm;}
         p = new Nod; p->v = vf; p->urm = pq; pq->urm = p; uz[vf] = 1;
        }
```

Pentru a afișa soluția vom parcurge lista astfel construită:

```
void Scrie()
{
    ofstream fout("coment.out");
    Lista p = d;
    while (p)
        {fout << p->v << ' ';
         p = p->urm;
        }
    fout.close();
}
```

12. Poșta

Un poștaș trebuie să treacă zilnic pe toate străzile din cartierul său. În fiecare zi el pleacă de la poșta și trebuie să se întoarcă la poșta. Cum în cartier sunt m străzi, după o săptămână deja își pune problema să își optimizeze traseul. Pentru aceasta a început prin a numerota intersecțiile, începând cu 1 pentru intersecția în care se găsește poșta. Pentru a se

feri de complicații, poștașul a considerat că într-o intersecție nu este obligatoriu să se întâlnească două străzi, este suficient ca locul respectiv să fie un capăt de stradă. Apoi și-a făcut o hartă, pe care a marcat toate intersecțiile și toate străzile. A observat că în cartierul său pot exista străzi distincte care au ambele capete respectiv în aceleși intersecții și că de la poșta poate ajunge pe orice stradă doar, trecând doar pe străzi din cartierul său. Poștașul ar prefera un traseu pe care să nu fie nevoie să treacă de două ori pe o același stradă.

Scrieți un program care să testeze dacă poștașul are șanse să își îndeplinească visul și dacă da, să determine un traseu convenabil.

Date de intrare

Fișierul de intrare POSTAS.IN conține:

n	- numărul de intersecții ($1 \leq n \leq 1000$)
m	- numărul de străzi din cartier ($1 \leq m \leq 1000$)
$j_1 \ j_1$	- extremitățile primei străzi
$j_2 \ j_2$	- extremitățile celei de-a doua străzi
...	
$j_m \ j_m$	- extremitățile străzii cu numărul m

Date de ieșire

Rezultatele vor fi afișate în fișierul POSTAS.OUT.

Prima linie din fișier conține mesajul DA, dacă problema are soluție, respectiv NU, în caz contrar. Dacă problema are soluție, în fișierul de ieșire vor fi afișate intersecțiile în ordinea în care sunt parcuse pe un traseu optim (unul în care fiecare stradă este parcursă o singură dată), începând cu 1, intersecția de plecare și terminând de asemenea cu 1.

Exemplu

POSTAS.IN	POSTAS.OUT
4	DA
5	1 4 1 2 3 1
1 4	
1 2	
1 3	
1 4	
2 3	

Timp maxim de execuție: 1 secundă/test.

(Tabără de pregătire, Poiana Pinului, 1998)

Soluție

Asociem problemei un graf neorientat, în care vîrfurile sunt intersecțiile, iar străzile reprezintă muchiile grafului (mai exact, acesta este un multigraf; deoarece pot exista mai multe muchii cu același extremități). Problema solicită determinarea unui ciclu eulerian în graful asociat.

Conform teoremei de caracterizare a grafurilor euleriene, condiția necesară și suficientă ca un graf fără vîrfuri izolate să fie eulerian este ca el să fie conex și toate vîrfurile sale să

aibă grad par. Din enunțul problemei se deduce faptul că graful este conex. Prin urmare trebuie doar să verificăm paritatea gradelor vîrfurilor grafului.

Vom reprezenta graful prin liste de adiacență, calculând pe parcursul creării reprezentării gradele vîrfurilor:

```
#define DimMax 1000
struct NodLista {int v;
                 NodLista * urm;};
typedef NodLista* Lista;
typedef Lista Graf[DimMax];
Graf G;
int n; //n - numarul de varfuri din graf
int m; //m - numarul de muchii din graf
int d[DimMax]; //gradele varfurilor grafului

void Citire()
{
    ifstream fin("postas.in");
    int x, y;
    Lista p;
    fin>>n>>m;
    for (int i=1; i<=m; i++)
    {
        fin>>x>>y;
        d[x]++; d[y]++;
        p=new NodLista; p->v=x; p->urm=G[y]; G[y]=p;
        p=new NodLista; p->v=y; p->urm=G[x]; G[x]=p;
    }
    fin.close();
}
```

Testarea condiției de existență a unui ciclu eulerian se reduce la verificarea parității tuturor gradelor vîrfurilor grafului:

```
int EsteEulerian()
{
    for (int i=1; i<=n; i++)
        if (d[i] % 2) return 0;
    return 1;
}
```

Pentru a determina un ciclu eulerian vom utiliza algoritmul constructiv din demonstrația teoremei de caracterizare a grafurilor euleriene.

Construim un ciclu în graf, plecând de la un vîrf oarecare al grafului, notat x_1 ; deoarece x_1 nu poate fi izolat, deducem că există o muchie incidentă cu x_1 , să o notăm $[x_1, x_2]$. Vîrful x_2 are grad par, prin urmare dacă am utilizat muchia $[x_1, x_2]$ pentru a „intră” în vîrful x_2 , există cel puțin o muchie, să o notăm $[x_2, x_3]$, care ne permite să „ieșim” din vîrful x_2 . Multimea muchiilor grafului fiind finită, după un număr oarecare de deplasări de-a lungul muchiilor grafului după acest procedeu, vom reveni în vîrful de plecare x_1 , obținând astfel un ciclu.

Funcția următoare determină un ciclu plecând din vîrful de start x. Ciclul este reprezentat ca o listă simplu înlănțuită; primul vîrf al listei este referit de pointerul c1, iar ultimul vîrf al listei este referit de pointerul sf.

Muchiile grafului care au fost utilizate pentru construirea ciclului sunt eliminate din graf, pentru a evita utilizarea unei muchii de mai multe ori.

```

Lista c, cl, sf;

void DeterminaCiclu (int x)
(Lista p; Lista q, aux; int y, xs=x;
cl=new NodLista; cl->v=x; cl->urm=NULL; sf=cl;
do { y=G[x]->v;
//plasez in ciclul cl muchia [x,y]
p=new NodLista; p->v=y; p->urm=NULL; sf->urm=p; sf=p;
//elimin din graf muchia [x,y]
q=G[x]; G[x]=G[x]->urm; delete q;
if (G[y]->v == x)
{q=G[y]; G[y]=G[y]->urm; delete q;}
else
{q=G[y];
while (q->urm->v!=x) q=q->urm;
aux=q->urm; q->urm=aux->urm; delete aux; }
x=y;
while (y!=xs); }

```

Este posibil ca ciclul C astfel determinat să fie eulerian, caz în care problema este rezolvată. Dacă ciclul C nu este eulerian, deducem că există mulțimea M a muchiilor din graf care nu aparțin acestui ciclu este nevidă. Graful dat fiind conex, deducem că există cel puțin o muchie în mulțimea M incidentă cu un vârf de pe ciclul C (în caz contrar, ciclul C ar forma o componentă conexă distinctă a grafului, ceea ce contrazice ipoteza).

Să considerăm o astfel de muchie din mulțimea M notată $[y_1, y_2]$, vârful y_1 aparținând ciclului C. Vom construi un nou ciclu, plecând din vârful y_1 , deplasându-ne pe muchiile din mulțimea M, fără a trece de două ori prin aceeași muchie (acest lucru este posibil datorită parității gradelor vârfurilor). Numărul de muchii din M fiind finit, după un număr oarecare de deplasări vom reveni în vârful de plecare y_1 .

Am obținut astfel un nou ciclu C_1 , care are un vârf comun cu ciclul C și mulțimele muchiilor disjuncte. Reunind ciclurile C și C_1 obținem un ciclu cu număr mai mare de muchii decât ciclul initial.

Procedeul se repetă până când numărul de muchii de pe ciclul astfel obținut este maxim (mulțimea M a muchiilor din graf care nu aparțin ciclului este vidă), deci ciclul obținut este eulerian.

Funcția CautMuchie() determină o muchie incidentă cu un vârf de pe ciclul C, muchie care nu aparține ciclului:

```

Lista CautMuchie()
{Lista p=c;
do
{if (G[p->v]) return p;
p=p->urm; }
while (p!=c);
return NULL; }

```

Funcția DeterminăCicluEulerian() determină un ciclu eulerian în graf prin reuniri succesive de cicluri disjuncte, conform procedeului prezentat anterior:

```

void DeterminaCicluEulerian()
(Lista p;
DeterminaCiclu(l);
sf->urm=cl; c=cl;
do {
p=CautaMuchie();
if (p)
{DeterminaCiclu(p->v);
Adauga(p);}
}
while (p); }

```

13. SICN

Serviciul de Informații al Comisiei Naționale (SICN) este constituit din n agenți ($1 \leq n \leq 150$), cu numere de cod distințe de la 1 la n și un agent șef codificat 0. Din motive de securitate, nu oricare doi agenți au contacte informaționale directe, dar prin contactele directe existente, oricare doi agenți își pot comunica informații. Un serviciu de informații este considerat *forte* dacă și numai dacă nu conține nici un agent prin suprimarea căruia se compromite comunicarea (și ca urmare, să existe agenți care să nu își mai poată transmite informații).

Scrieți un program, care verifică dacă SICN este *forte*. În plus, dacă SICN nu este *forte*, programul să determine:

- verigile slabe ale SICN (agenții prin a căror suprimare se compromite comunicarea);
- toate grupurile de agenți care sunt *forte* în cadrul SICN, maximale cu această proprietate.

Date de intrare

Fișierul de intrare se numește SICN.ÎN și conține pe prima linie n , numărul de agenți din SICN (exclusiv șeful), iar pe fiecare din următoarele linii câte o pereche de numere x și y unde $x, y \in \{0, 1, 2, \dots, n\}$, sunt separate prin spații și au semnificația „ x și y au contact informațional direct”.

Date de ieșire

Fișierul de ieșire, numit SICN.OUT, va conține mesajul SICN este forte sau:

- pe prima linie, mesajul SICN nu este forte;
- pe a doua linie, numerele de cod ale agenților care constituie verigile slabe ale serviciului, în ordine crescătoare, separate prin spații;
- pe a treia linie, m – numărul de grupuri de agenți *forte* în cadrul SICN;
- pe fiecare din următoarele m linii, numerele de cod ale agenților fiecarui grup *forte*, în ordine crescătoare, separate prin spații.

Exemplul 1

SOICN.IN
2
0 2
0 1
2 1

SOICN.OUT
SOICN este forte

Exemplul 2

SOICN.IN
3
0 1
0 3
2 1

SOICN.OUT
SOICN nu este forte
0 1
3
0 3
1 2
0 1

Timp maxim de execuție: 1 secundă/test.

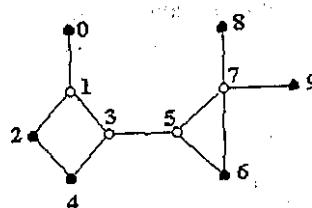
(ONI, Constanța, 2000, clasele a XI-a - a XII-a)

Soluție

Puteam asocia problemei un graf astfel: mulțimea vârfurilor este mulțimea agenților; între două vârfuri există muchie dacă și numai dacă agenții corespunzători celor două vârfuri au contact informațional direct. Din enunțul problemei deducem că graful asociat vârfuri este conex. Problema solicită de fapt să verificăm dacă graful este biconex, iar dacă nu să determinăm punctele de articulație din graf și componentele biconexe ale grafului.

Este graf neorientat conex. Vârfurile care sunt numere pare sunt puncte de articulație, dacă sub graful obținut prin eliminarea vârfului x, multinișor incidente cu acesta nu mai este conex.

De exemplu, pentru graful din figură punctele de articulație sunt 1, 3, 5, 7.

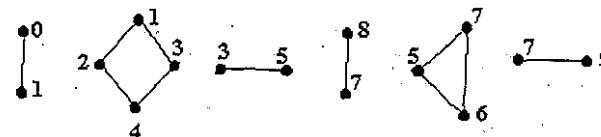


Un graf se numește biconex dacă nu are puncte de articulație.

În multe aplicații practice, ce se pot modela cu ajutorul grafurilor, nu sunt de dorit punctele de articulație. De exemplu, într-o rețea de telecomunicații, dacă o centrală dintr-un punct de articulație se defectează rezultatul este nu numai întreruperea comunicării cu centrala respectivă, ci și cu alte centrale.

O componentă biconexă dintr-un graf este un subgraf biconex maxim cu această proprietate.

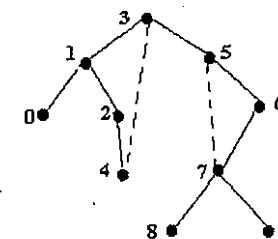
De exemplu, pentru graful din figură componentele biconexe sunt:



Pentru a descompune graful în componente biconexe vom utiliza proprietățile parcurgerii DFS. Parcugând graful DFS putem clasifica muchiile grafului în:

- muchii care aparțin arborelui parțial DFS (*tree edges*);
- muchii $[u, v]$ care nu aparțin arborelui și care unește vârful u cu un strămoș al său v în arborele parțial DFS numite muchii de întoarcere (*back edges*). Acestea sunt marcate în exemplu punctat.

De exemplu graful de mai sus poate fi redesenat, clasificând muchiile ținând cont de arborele parțial DFS cu rădăcina 3:



Observăm că rădăcina arborelui parțial DFS este punct de articulație dacă și numai dacă are cel puțin doi descendenți, între vârfuri din subarbore diferenți ai rădăcinii neexistând muchii. Mai mult, un vârf x oarecare nu este punct de articulație dacă și numai dacă din orice fiu y al lui x poate fi atins un strămoș al lui x pe un lanț format din descendenți ai lui x și o muchie de întoarcere (un drum „de siguranță” între x și y).

Pentru fiecare vârf x din graf definim $dfn(x) =$ numărul de ordine al vârfului x în parcugerea DFS a grafului (depth first search number).

De exemplu

x	0	1	2	3	4	5	6	7	8	9
dfn(x)	2	1	3	0	4	5	6	7	8	9

Observăm că dacă x este un strămoș al lui y în arborele parțial DFS atunci $dfn(x) < dfn(y)$.

Pentru fiecare vârf x din graf definim $low(x) =$ numărul de ordine al primului vârf din parcugerea DFS ce poate fi atins din x pe un alt lanț decât lanțul unic din arborele parțial DFS.

$low(x) = \min\{dfn(x), \min\{low(y) | y \text{ fiu al lui } x\}, \min\{dfn(y) | [x, y] \text{ muchie de întoarcere}\}\}$.

De exemplu

x	0	1	2	3	4	5	6	7	8	9
dfn(x)	2	1	3	0	4	5	6	7	8	9
low(x)	2	1	1	0	1	5	5	5	8	9

Caracterizam punctele de articulație dintr-un graf astfel: x este punct de articulație dacă și numai dacă este rădăcina unui arbore parțial DFS (cu celelalte noduri descendente) și dacă nu este adiacență unui alt nod în care $low(x) > dfn(u)$.

Pentru exemplul din figură nodul 3 este punct de articulație deoarece este rădăcina arborelui parțial DFS și are doi descendenți, nodul 7 este punct de articulație deoarece $low(8)=8 \geq dfn(7)=7$, nodul 5 este punct de articulație deoarece $low(6)=5 \geq dfn(5)=5$, iar nodul 1 este punct de articulație deoarece $low(0)=2 \geq dfn(1)=1$.

Reprezentarea informațiilor

Vom reprezenta graful prin liste de adiacență, alocate dinamic:

```
#define NMaxVf 151
struct NodLista
{
    int v;
    NodLista * leg;
};
typedef NodLista * Lista;
typedef Lista Graf[NMaxVf];
int n; //numarul de varfuri din graf
Graf G;

În funcția Citire() vom crea reprezentarea grafului prin liste de adiacență:

void Citire()
{
    ifstream fin("sicn.in");
    Lista p;
    int x,y;
    fin>>n;
    while (!fin.eof())
    {
        fin>>x>>y; //citesc muchia [x,y]
        if (fin.good())
            {p=new NodLista; p->v=x; p->leg=G[y]; G[y]=p;
             //inserez x in lista de adiacenta a lui y
             p=new NodLista; p->v=y; p->leg=G[x]; G[x]=p;
             //inserez y in lista de adiacenta a lui x
            }
    }
    fin.close();
}
```

Vectorii $dfn[]$ și $low[]$ conțin valorile calculated în timpul parcurgerii în adâncime, variabila globală num este utilizată pentru a calcula numărul de ordine al vârfului curent în parcurgerea în adâncime, iar nrfii0 reține numărul de fii ai vârfului 0 în arborele parțial DFS.

```
int dfn[NMaxVf], low[NMaxVf];
int num, nrfii0;
```

Vom utiliza o stivă S în care vom reține muchiile din graf (atât cele care aparțin arborelui, cât și cele de întoarcere) în ordinea în care sunt întâlnite în timpul parcurgerii. Atunci când identificăm o componentă biconexă, mai exact când identificăm un nod u care are un fiu x astfel încât $low(x) \geq dfn(u)$, eliminăm din stivă toate muchiile din componenta biconexă respectivă.

```
struct NodStiva
{
    int t; //tatal nodului curent
    int f; //nodul curent
    NodStiva *urm;
};
typedef NodStiva* Stiva;
Stiva S;
```

Funcția Initializare() initializează stiva cu o muchie fictivă (cu extremitatea finală 0 și extremitatea inițială -1, un nod inexistent) și vectorii $dfn[]$ și $low[]$:

```
void Initializare()
{
    for (int i=0; i<=n; i++) dfn[i]=low[i]=-1;
    S=new NodStiva; S->f=0; S->t=-1; S->urm=NULL; }
```

Vom reprezenta o componentă biconexă ca mulțimea vârfurilor sale. Descompunerea grafului în componente biconexe va fi un vector D în care reținem pentru fiecare componentă biconexă determinată mulțimea sa de vârfuri.

```
typedef int MultimeVf[NMaxVf];
typedef MultimeVf Descompunere[NMaxVf];
Descompunere D; //descompunerea în componente biconexe
MultimeVf A; //multimea punctelor de articulație
int nr; // numarul de componente biconexe
```

În funcția main(), după citire și initializare, vom apela funcția Biconex(0,-1) care realizează descompunerea grafului în componente biconexe, apoi afișăm rezultatele.

```
ofstream fout("sicn.out"); //fisierul de ieșire
void main()
{
```

```
Citire();
Initializare();
Biconex(0,-1);
if (nr==1) //graful este biconex
    {fout<<"FORTE"<<endl; fout.close(); return;}
if (nrfii0>1) //0 este punct de articulație
    A[0]=1;
```

```
Afisare(A); //afisez multimea punctelor de articulație
fout<<nr<<endl; //afisez numărul de componente biconexe
for (int i=1; i<=nr; i++) //afisez componente biconexe
    Afisare(D[i]);
fout.close(); }
```

Afișarea unei multimi de vîrfuri (reprezentată prin vector caracteristic) se realizează astfel:

```
void Afisare (MultimeVf X)
{for (int i=0; i<=n; i++)
    if (X[i]) fout<<i<<' ';
fout<<endl;
```

Functia Biconex() calculează dfn[] și low[] și determină componentele biconexe:

```
void Biconex(int u,int tu)
//u este nodul curent; tu este nodul parinte al lui u
{Stiva q;
int x;
dfn[u]=low[u]=num++;
Lista p=G[u]; //parcure lista de adiacenta a nodului u
while (p)
    {x=p->v; //x este un nod adjacente cu u
     if (x!=tu & dfn[x]<dfn[u])
        //insereaza in stiva S muchia [u,x]
        {q=new NodStiva; q->f=x; q->t=u; q->urm=S; S=q;}
     if (dfn[x]==-1) //x nu a mai fost vizitat
        {if (u==0) //am gasit un descendant al lui 0
            {if (u==0) //am gasit un descendant al lui 0
                nrfl0++;
                Biconex(x,u);
                low[u]=min(low[u],low[x]);
                if (low[x]>=dfn[u]) //u este punct de articulatie
                    //am identificat o componenta biconexa,
                    //formata din muchiile din stiva S pana la
                    //intalnirea muchiei [u,x]
                    {if (u) A[u]=1;
                     //retin punctul de articulatie u
                     Constructie_Comp_Biconexa(x,u);}
                Constructie_Comp_Biconexa(x,u); //x a mai fost vizitat
            }
        else
            if (x!=tu)
                //x nu este tatal lui u,
                //deci [u,x] e muchie de intoarcere la u la x
                low[u]=min(low[u],dfn[x]);
            p=p->leg; }
    }
}
```

Functia Constructie_Comp_Biconexa(x,u) determină componenta biconexă din care face parte muchia [x,u]:

```
void Constructie_Comp_Biconexa(int x, int u)
{int a,b;
Stiva p; //incrementez numarul de componente Biconexe
nr++;
```

```
do
{p=S; S=S->urm; //extrag un nod din stiva
 a=p->t; b=p->f; delete(p);
 D[nr][a]=1; D[nr][b]=1;
 while (a!=u || b!=x); //nu am ajuns la muchia [x,u]
}
```

Observație

Algoritmul de descompunere în componente biconexe a unui graf reprezentat prin liste de adiacență este liniar ($O(n+m)$), unde n este numărul de vîrfuri, iar m numărul de muchii din graf.

14. Digraf cu număr minim de arce

Fie G un digraf cu n vîrfuri, numerotate 1, 2, ..., n ($n \leq 50$). Să se construiască un alt digraf, având același număr de vîrfuri, aceeași matrice a drumurilor și număr minim de arce.

(Baraj, IOI, 1992)

Observație

Un arc din graful inițial este eliminat sau nu și este permisă și introducerea de noi arce.

Date de intrare

Fișierul de intrare GRAF.IN conține pe prima linie n , numărul de vîrfuri, iar pe următoarele n linii matricea de adiacență a digrafului G .

Date de ieșire

Fișierul de ieșire GRAF.OUT va conține matricea de adiacență a digrafului cu număr minim de arce obținut.

Soluție

Vom reprezenta digrafurile prin matrice de adiacență:

```
#define NMaxVf 51
typedef unsigned Graf[NMaxVf][NMaxVf];
Graf G, Gc;
int n;
```

După citire, vom transforma matricea de adiacență a digrafului G în matricea drumurilor prin algoritmul Roy-Warshall, apoi vom descompune digraful în componente tare-conexe, utilizând matricea drumurilor. Componentele tare-conexe le vom reține în matricea C (linia i corespunde componentei tare-conexe i , pe poziția 0 fiind memorat numărul de vîrfuri din componenta tare-conexă i).

```
int C[NMaxVf][NMaxVf], Nrc;
void Descompunere_in_Componente_Tare_Conexe()
{
    unsigned viz[NMaxVf];
```

```

for (int i=0; i<=n; i++) viz[i]=0;
for (i=Nrc=1; i<=n; i++)
    if (!viz[i])
        {for (int j=1; j<=n; j++)
         if (G[i][j] && G[j][i])
            {C[Nrc][0]++;
             C[Nrc][C[Nrc][0]]=j;
             viz[j]=1;}
         Nrc++;}
Nrc--; //numarul de componente tare-conexe
}

```

Asociem digrafului G digraful condensat G_c , în care fiecare vîrf corespunde unei componente tare-conexe. Dacă există legătură de la componenta tare-conexă i la componenta tare-conexă j , atunci va exista un drum de la vîrful corespunzător componentei i la vîrful corespunzător componentei j în digraful condensat.

```

void Constructie_Matrice_Drumuri_Graf_Condensat()
{
    for (int i=1; i<=Nrc; i++)
        for (int j=1; j<=Nrc; j++)
            Gc[i][j]=G[C[i][1]][C[j][1]];
}

```

Să observăm că graful condensat nu mai conține circuite. Eliminăm arcele inutile din graful condensat

```

void Eliminare_Arce_Inutile_in_Graf_Condensat()
//graful condensat nu contine circuite
{
    int i, j, k;
    Graf Gaux;
    for (i=1; i<=Nrc; i++) Gc[i][i]=0;
    for (i=1; i<=Nrc; i++)
        for (j=1; j<=Nrc; j++)
            Gaux[i][j]=Gc[i][j];
    for (i=1; i<=Nrc; i++)
        for (j=1; j<=Nrc; j++)
            for (k=1; k<=Nrc && Gc[i][j]; k++)
                if (Gaux[i][k]*Gaux[k][j])
                    Gc[i][j]=0;
}

```

Construim digraful cu număr minim de arce considerând că fiecare componentă tare-conexă este formată dintr-un singur circuit hamiltonian și adăugând pentru fiecare arc rămas în graful condensat un arc între reprezentanții componentelor tare-conexe corespunzătoare extremităților.

```

void Constructie_Graf()
{
    int i, j;
    for (i=1; i<=n; i++)
        for (j=1; j<=n; j++) G[i][j]=0;
}

```

```

for (i=1; i<=Nrc; i++)
    {for(j=1;j<C[i][0];j++)//construiesc ciclul componentei i
     G[C[i][j]][C[i][j+1]]=1;
     if (C[i][0]>1) //inchid ciclul
         G[C[i][C[i][0]]][C[i][1]]=1;
    }
    for (i=1;i<=Nrc;i++) //creez legăturile dintre componente
        for (j=1; j<=Nrc; j++)
            if (Gc[i][j]) G[C[i][1]][C[j][1]]=1;
}

```

Păși algoritmului sunt ilustrați în succesiunea apelurilor funcțiilor în main():

```

void main()
{
    Initializare();
    Descompunere_in_Componente_Tare_Conexe();
    Constructie_Matrice_Drumuri_Graf_Condensat();
    Eliminare_Arce_Inutile_in_Graf_Condensat();
    Constructie_Graf();
    Afisare();
}

```

15. Broscuța ergonomică

Broscuța Oac stătea pe lac pe o frunză de nufăr. Fiind hidrofobă, broscuță ar vrea să ajungă la mal fără să se ude, sărind de pe o frunză de nufăr pe alta.

La fiecare săritură broscuță depune efort și consumă calorii, atât pentru a sări, cât și pentru a-și restabili echilibrul. Numărul de calorii consumate pentru un salt este proporțional cu distanța pe care sare broscuță, factorul de proporționalitate fiind k . Numărul de calorii consumate pentru restabilirea echilibrului este întotdeauna același, p. De asemenea, broscuță și-a autoevaluat condiția fizică și a ajuns la concluzia că nu poate sări deodată mai mult de h ori.

Broscuță a fixat un sistem de coordonate cartezian cu centrul în pomul de pe mal și axa OX de-a lungul malului rectiliniu, apoi a calculat coordonatele frunzelor de nufăr, considerate punctiforme, relativ la acest sistem de coordonate.

scrieți un program care să determine, dacă este posibil, un traseu pentru broscuță, de la frunza pe care stă până la mal, cu consum minim de calorii.

Date de intrare

Datele de intrare se citesc din fișierul LAC.IN care are următoarea structură:

- n – numărul de frunze de nufăr ($n \in \mathbb{N}^*$, $n \leq 500$)
- $x_1 \ y_1$ – coordonatele primei frunze de nufăr ($x_1, y_1 \in \mathbb{R}_+$), cea pe care stă broscuță
- $x_2 \ y_2$ – coordonatele celei de a doua frunze de nufăr ($x_2, y_2 \in \mathbb{R}_+$)
- ...
- $x_n \ y_n$ – coordonatele celei de a n-a frunze de nufăr ($x_n, y_n \in \mathbb{R}_+$)
- h – distanță maximă pe care o poate sări broscuță ($h \in \mathbb{R}_+$)
- k – factorul de proporționalitate ($k \in \mathbb{R}_+$)
- p – consumul de calorii pentru restabilirea echilibrului ($p \in \mathbb{R}_+$).

Date de ieșire

În fișierul de ieșire LAC.OUT veți afișa mesajul IMPOSSIBIL dacă broșuță nu poate ajunge la mal fără să se ude sau veți afișa pe prima linie numărul total de calorii consumate pentru a ajunge la mal.

Pe fiecare dintre următoarele linii veți afișa câte două numere separate prin spațiu, reprezentând pozițiile frunzelor de nufăr (cu două zecimale cu rotunjire) în ordinea în care au fost atinse pentru a ajunge la mal:

- c – număr total de calorii consumate
- x1 y1 – coordonatele primei frunze de nufăr de pe traseu, cea pe care stătea broșuță
- x2 y2 – coordonatele celei de a doua frunze de nufăr de pe traseu
- ...
- xm ym – coordonatele ultimei frunze de nufăr de pe traseu.

Exemplu

LAC.IN	LAC.OUT
2	211.00
0 10	0.00
3 6	3.00
7	10.00
1	6.00
100	

Soluție

Puteam asocia problemei un graf neorientat cu $n+1$ vârfuri (cele n frunze de nufăr și un vârf suplimentar, notat 0, corespunzător malului). Considerăm că între două vârfuri există muchie în graf dacă și numai dacă distanța dintre ele nu depășește h , lungimea maximă a saltului pe care îl poate face broșuță.

Costul asociat unei muchii este egal cu consumul de calorii corespunzător unui salt având lungimea muchiei respectivă ($k * \text{lungime_muchie} + p$).

Problema se reduce astfel la a determina un drum de cost minim între două vârfuri din graf (vârful 1, corespunzător poziției initiale a broșuței și vârful 0, corespunzător malului). În acest scop vom utiliza algoritmul lui Dijkstra.

Vom reține pozițiile frunzelor de nufăr într-un vector L, cu componente de tip structură:

```
#define NMaxFrunze 500
#define Infinit 1E15
struct Pozitie {float x, y;};
Pozitie L[NMaxFrunze]; //pozițiile frunzelor
int n; //numarul de frunze
float h, k, p;
```

Pentru a determina drumurile de cost minim, vom utiliza doi vectori suplimentari, d și inainte având următoarea semnificație:

d[i] – costul drumului minim de la vârful 1 la vârful i;
inainte[i] – vârful care precedă imediat vârful i, pe drumul minim de la vârful 1 la vârful i (sau -1 dacă i=1).

```
float d[NMaxFrunze];
int inainte[NMaxFrunze];
```

Funcția dist() determină consumul de calorii pentru un salt între pozițiile corespunzătoare vârfurilor i și j, transmise ca parametru. Dacă saltul nu este posibil, funcția returnează valoarea Infinit.

```
#define sqr(x) (x*x)
float dist (int i, int j);
{float d;
if (i && j)
    d=sqrt (sqr(L[i].x-L[j].x)+sqr(L[i].y-L[j].y));
else if (!i) d=L[j].y;
else d=L[i].y;
if (d<=h) return k * d + p;
return Infinit;}
```

După citirea datelor de intrare, vom inițializa vectorii d și inainte:

```
void Citire()
{ifstream fin("lac.in");
fin>>n;
for (int i=1; i<=n; i++) fin>>L[i].x>>L[i].y;
fin.close();
for (i=0; i<=n; i++)
{inainte[i]=-1;
d[i]=dist (1,i);}
d[1]=0; inainte[1]=-1;}
```

Funcția Dijkstra() implementează algoritmul lui Dijkstra. Funcția returnează valoarea 1 dacă există drum de la vârful 1 la vârful 0, respectiv valoarea 0 în caz contrar.

```
int Dijkstra()
{int M[NMaxFrunze];
//M[i] este multimea varfurilor selectate, pentru care am calculat deja drumul minim de la varful 1 */
int fmin;
float min;
for (int i=0; i<=n; i++) M[i]=0; M[1]=1;
while (!M[0])
/*cat timp drumul minim de la varful 1 la varful 0 nu este calculat */
{min=Infinit;
/* determin un varf pentru care nu am calculat deja drumul minim, situat la distanța minima de multimea varfurilor selectate */
for (i=0; i<=n; i++)
    if (!M[i] && d[i]<min)
        {min=d[i]; fmin=i; }
if (min==Infinit) return 0;
M[fmin]=1;
//actualizez distantele pana la varfurile neselectate
```

```

for (i=0; i<=n; i++)
    if (!M[i] && d[i] > d[fmin] + dist (fmin, i))
        {d[i]=d[fmin]+dist (fmin, i);
         inainte[i]=fmin;}
    }
return 1;
}

```

Afișarea drumului minim se va realiza utilizând informațiile memorate în vectorul inainte:

```

void Afisare()
{
ofstream fout("lac.out");
int m, i; int t[NMaxFrunze];
if (!Dijkstra()) fout<<"IMPOSSIBIL\n";
else
{fout<<d[0]<<endl;
m=1; i=0;
while (inainte[i]!=-1)
{t[m++]=i; i=inainte[i];}
t[m]=1;
for (i=m; i>1; i--) fout<<L[t[i]].x<<' '<<L[t[i]].y;
fout.close();
}

```

16. Litoral

Pe litoralul unei țări există două plaje particulare denumite A și B, între care se poate circula numai cu șalupa. Orarul de funcționare a șalupelor este cunoscut și conține ora de plecare și ora de sosire de pe o plajă pe cealaltă. Evident, orice turist care se deplasează cu o șalupă trebuie să cumpere un bilet de călătorie, al cărui preț poate să difere de la o șalupă la alta.

Proprietarii plajelor au pretenția la o taxă de sedere pe plajă lor, taxă care va fi încasată la anumite ore de la persoanele existente la aceea oră pe plajă respectivă. În cazul în care o persoană sosește pe o plajă după momentul de încasare, nu plătește până la momentul de încasare imediat următor. Aceste taxe pot fi de valori diferite în funcție de ora perceperei lor. În cazul în care o șalupă pleacă sau ajunge la un anumit moment pe o plajă și exact în acel moment se percepe taxă de sedere pe acea plajă, atunci se va plăti și taxa de sedere.

Un Tânăr se află pe plaja A la ora x, cunoscută. El are întâlnire cu prietena lui pe plaja B, la o anumită oră y. El va încerca să-și planifice „staționările” pe plaje și „plimbările” cu șalupa astfel încât la ora y să fie la destinație, plătind taxe de sedere pe plaje și taxe de transport cu șalupa care însuțează să fie de valoare minimă.

Determinați planul pe care el trebuie să-l realizeze specificând și costul acestuia.

Date de intrare

Fișierul text INPUT.TXT are următoarea structură:

- pe prima linie sunt scrise două numere: oA ($0 \leq oA \leq 23$) și mA ($0 \leq mA \leq 59$) reprezentând ora și minutul la care Tânărul se află inițial pe plaja A;
- pe a doua linie sunt scrise două numere: oB ($0 \leq oB \leq 23$) și mB ($0 \leq mB \leq 59$) reprezentând ora și minutul la care se vor întâlni cei doi tineri pe plaja B;

- pe a treia linie sunt scrise două numere ntA și ntB reprezentând numărul de taxe percepute pe plaja A ($0 \leq ntA \leq 100$), respectiv numărul de taxe percepute pe plaja B ($0 \leq ntB \leq 100$) între momentul inițial și momentul întâlnirii;
- pe următoarele ntA linii sunt scrise câte trei numere sub forma: h m c unde h și m reprezintă ora și minutele perceperei taxei pe plaja A, iar c reprezintă valoarea taxei ($c \leq 150$);
- pe următoarele ntB linii sunt scrise câte trei numere sub forma: h m c unde h și m reprezintă ora și minutele perceperei taxei pe plaja B, iar c reprezintă valoarea taxei ($c \leq 150$);
- următoarea linie conține numărul natural ns ≤ 80 reprezentând numărul total de șalupe care se deplasează între cele două plaje;
- pe următoarele ns linii este descris orarul salupelor folosind câte șase numere scrise sub forma: d hA mA hB mB c. Valoarea d poate fi doar 0 sau 1; dacă d este 0, atunci hA și mA reprezintă ora și minutele plecării șalupei din A, iar hB și mB reprezintă ora și minutele sosirii în B; dacă d este 1, atunci hA și mA reprezintă ora și minutele sosirii șalupei în A, iar hB și mB reprezintă ora și minutele plecării din B. Valoarea c reprezintă costul biletului de călătorie ($c \leq 150$).

Datele sunt despărțite pe linii prin căte un spațiu. Datele de intrare se consideră corecte și admit întotdeauna soluție.

Date de ieșire

Fișierul text OUTPUT.TXT va conține pe prima linie numărul ctm, reprezentând costul minim al deplasării (compus din toate taxele plătite de Tânăr).

Exemplu

INPUT.TXT

```

1 0
9 0
3 3
2 0 10
4 0 2
6 0 10
3 0 10
5 0 10
7 0 2
7
0 1 0 2 0 3
0 3 0 4 0 6
0 5 0 6 0 8
0 7 0 8 0 9
1 3 0 2 0 5
1 5 0 4 0 7
1 7 0 6 0 9

```

OUTPUT.TXT

20.

(rezolvare în document)

Interpretarea exemplului

Ora	Şalupe	Plaja iniţială	Plaja finală	Taxa de şedere	Cost transport
1h 0m-2h 0m	1	A	B	-	3
2h 0m-3h 0m	5	B	A	-	5
3h 0m-5h 0m	-	A	-	2	-
5h 0m-6h 0m	3	A	B	-	8
6h 0m-9h 0m	-	B	-	2	-
				Total: 4	Total: 16

Soluție

Pentru simplificare, vom converti toți timpii care intervin în problemă în minute.

Puteam asocia problemei un graf orientat cu 2^*ns+2 vârfuri, unde ns reprezintă numărul de șalupe. Vârfurile grafului au următoarea semnificație:

- vârful 1 reprezintă timpul inițial;
- vârful 2^*ns+2 reprezintă timpul final;
- vârfurile de la 2 până la $ns+1$ reprezintă timpul specific sosirii sau plecării unei șalupe de pe plaja A;
- vârfurile de la $ns+1$ până la 2^*ns+1 reprezintă timpul specific sosirii sau plecării unei șalupe de pe plaja B.

Vom considera în graf următoarele arce:

- arce de la vârful 1 la acele vârfuri din mulțimea $\{2, \dots, ns+1\}$ care au asociat un timp mai mare decât timpul inițial; costul acestor arce este egal cu suma taxelor achităte pe plaja A în intervalul dintre timpul inițial și timpul asociat vârfului care reprezintă extremitatea finală a arcului;
- arce între oricare două vârfuri din mulțimea $\{2, \dots, ns+1\}$, cu condiția ca timpul asociat vârfului care reprezintă extremitatea inițială să fie mai mic decât timpul asociat vârfului care reprezintă extremitatea finală; costul acestor arce este egal cu suma taxelor achităte pe plaja A în intervalul dintre timpul asociat extremității inițiale și timpul asociat extremității finale a arcului;
- arce între oricare două vârfuri din mulțimea $\{ns+2, \dots, 2^*ns+1\}$, cu condiția ca timpul asociat vârfului care reprezintă extremitatea inițială să fie mai mic decât timpul asociat vârfului care reprezintă extremitatea finală; costul acestor arce este egal cu suma taxelor achităte pe plaja B în intervalul dintre timpul asociat extremității inițiale și timpul asociat extremității finale a arcului;
- arce de la oricare vârf din mulțimea $\{ns+2, \dots, 2^*ns+1\}$ la vârful 2^*ns+2 , cu condiția ca timpul asociat vârfului care reprezintă extremitatea inițială să fie mai mic decât timpul final; costul acestor arce este egal cu suma taxelor achităte pe plaja B în intervalul dintre timpul asociat extremității inițiale și timpul final;
- arce corespunzătoare traseelor șalupei; un astfel de arc are ca extremitate inițială vârful care reprezintă timpul de plecare al șalupei, ca extremitate finală vârful care reprezintă timpul de sosire al șalupei și costul egal cu costul biletului de călătorie cu șalupă respectivă.

Problema se reduce la a determina un drum de cost minim de la vârful 1 la vârful 2^*ns+2 , prin urmare se poate aplica algoritmul lui Dijkstra (vezi problema precedentă).

17. Color

Considerăm un graf complet cu N vârfuri în care muchiile sunt colorate fie în roșu, fie în negru. În acest graf se pot forma triunghiuri monocromatice (trei vârfuri conectate prin muchii colorate cu aceeași culoare).

Scrieti un program care să determine pentru un graf colorat dat numărul de triunghiuri monocromatice.

Date de intrare

Fișierul de intrare COLOR.IN conține:

COLOR.IN

N S

$x_1 \ y_1$

$x_2 \ y_2$

...

$x_s \ y_s$

Semnificație

N -- numărul de vârfuri din graf, S -- numărul de muchii roșii

$x_i \ y_i$ -- extremitățile celei de a i-a muchii roșii

Date de ieșire

Fișierul de ieșire COLOR.OUT conține pe prima linie numărul de triunghiuri monocromatice.

Exemplu

COLOR.IN

4 4

1 2

2 3

4 3

4 2

COLOR.OUT

1

Timp maxim de execuție: 1 secundă/test.

Soluție

Numărul total de triunghiuri care se pot construi cu cele N vârfuri ale grafului complet este egal cu numărul de submulțimi de 3 elemente ale mulțimii $\{1, 2, \dots, N\}$, deci combinații de N luate câte 3:

```
double total=N*(N-1)*(N-2)/6.0;
```

Numărul de triunghiuri monocromatice se poate obține din numărul total de triunghiuri, scăzând numărul de triunghiuri care nu sunt monocromatice.

Pentru a calcula câte triunghiuri bicromatice există, vom calcula la citirea datelor de intrare „gradul negru” al fiecărui vârf (numărul de muchii negre incidente cu vârful respectiv):

```
void Citire()
{
    ifstream input("color.in");
    input>>n>>m;
    for (int i=1; i<=n; i++) negru[i]=n-1;
```

```

for (long k=1; k<=m; k++)
    {input>>i>>j;
     negru[i]--; negru[j]--;
    }
input.close();
    
```

Numărul de triunghiuri bicromatice în care poate interveni un vârf i este egal cu numărul total de perechi formate dintr-o muchie roșie și o muchie neagră, ambele incidente cu vârful i : $(\text{negru}[i]) * (\text{n}-1-\text{negru}[i])$.

Numărul total de triunghiuri bicromatice se obține însumând numerele triunghiurilor bicromatice pentru fiecare vârf și împărțind suma obținută la 2 (fiecare muchie este luată în considerare de două ori, o dată pentru fiecare extremitate a ei):

```

double ntri=0;
for (i=1; i<=n; i++)
    ntri+=(n-1-negru[i])*negru[i];
ntri=total-(ntri/2);
    
```

Probleme propuse

1. Electorale

În Utopia au avut loc alegeri pentru președinte. Conform legii din Utopia, președintele trebuie validat de cei N membri ai Adunării Generale. Pentru aceasta, președintele trebuie să primească cel puțin $2/3$ din voturile membrilor Adunării Generale. Între anumiți membri ai Adunării Generale există conflicte de interes. Doi membri aflați în conflict de interes votează diferit.

Cunoscând relațiile dintre membrii Adunării Generale, verificați dacă este posibilă Alegerea președintelui, conform legii.

Date de intrare/ieșire

În fișierul de intrare UTOPIA.IN se găsesc:

- pe prima linie N , numărul de membri din Adunarea Generală;
- pe următoarele linii, până la sfârșitul fișierului, câte două numere x și y , cuprinse între 1 și N , separate prin spațiu, având semnificația „persoanele x și y se află în conflict de interes”.

Pe ecran se va afișa răspunsul (DA sau NU).

2. Aciclic

Scrieți un program care testează dacă un graf neorientat este aciclic.

3. Bazin

Se consideră n bazin dispuse circular și lipite unul de altul în ordinea numerelor de ordine. Se dă lista celor m perechi de bazin care trebuie unite prin canale. Un număr de bazin poate să apară în mai multe perechi. Canalele pot fi realizate în interiorul sau în exteriorul cercului, cu condiția ca ele să nu se intersecteze. Dacă problema are soluție, vor fi tipărite două liste: cea a perechilor de bazin unite prin canale exterioare și cea a

perechilor de bazin unite prin canale interioare. Dacă problema nu are soluție va fi tipărit un mesaj corespunzător.

(ONI, Iași, 1993)

4. Localități

Se consideră localitățile L_1, L_2, \dots, L_n , cu $n > 0$ dat. Se citește de la intrare o mulțime de perechi (i, j) cu $i, j \in \{1, 2, \dots, n\}$, având semnificația: „din localitatea de plecare i se poate ajunge direct în localitatea de destinație j ”.

1. Să se verifice dacă din oricare localitate se poate ajunge în oricare localitate.
2. Fiind precizată o submulțime S de localități, să se verifice dacă ea este „inaccesibilă”, adică dacă din afara submulțimii nu se poate ajunge în nici o localitate a ei.
3. Să se determine grupurile de localități cu proprietățile:
 - a. din oricare localitate a unui grup se poate ajunge în oricare localitate a aceluiasi grup;
 - b. grupurile de localități sunt maximale cu proprietatea a.
4. Fie G_1, G_2, \dots, G_k grupurile de localități determinate la punctul 3. Se cere să se listeze localitățile într-o ordine respectând restricția: dacă din L_i se poate ajunge în L_j și cele două localități fac parte din grupuri diferite, atunci L_i să apară în listă înaintea lui L_j .
5. Să se adauge un număr minim de perechi la cele inițiale astfel încât ordinea de listare a grupurilor de la punctul 4 să fie unică.

(ONI, Arad, 1992)

5. Sateliți

Se preconizează lansarea pe orbită a unui grup de sateliți. Doi sateliți pot transmite date de la unul la altul fie în mod direct, fie indirect (adică prin intermediul altor sateliți intermediari); de asemenea este posibil să nu aibă nici o legătură. Spunem că doi sateliți sunt vecini numai dacă ei pot transmite unul altuia date în mod direct.

Fiind date un număr n și un set de n numere întregi, se cere:

- a. Să se stabilească dacă există o configurație de n sateliți astfel încât setul de n numere citite la intrare să corespundă numerelor de vecini ai sateliților;
- b. Dacă răspunsul la punctul a este YES, atunci descrieți o astfel de configurație;
- c. Fiind dată configurația din b, decideți dacă se pot trimite date de la oricare satelit la oricare altul.

Date de intrare

Fișierul de intrare INPUT.TXT conține o singură linie pe care se află o secvență de numere întregi separate prin câte un spațiu. Primul număr, $n \leq 20$ reprezintă numărul de sateliți de pe orbită. Următoarele n numere întregi reprezintă numărul de vecini ai fiecărui satelit. Atenție! Ordinea în care sunt date numerele vecinilor direcți nu are nici o importanță.

Date de ieșire

Prima linie a fișierului de ieșire OUTPUT.TXT conține răspunsul la punctul a: YES sau NO. Dacă răspunsul este NO, nu se mai cere altceva la ieșire; dacă răspunsul este YES,

atunci urmează n linii, fiecare cu câte n numere 0/1, separate prin câte un spațiu, formând o matrice $n \times n$. Sateliți i și j sunt vecini dacă și numai dacă al (i, j) -lea element al matricei este 1. A $(n+2)$ -a linie a fișierului de ieșire corespunde punctului c și este un YES sau NO.

Exemplul 1

INPUT.TXT
6 4 3 1 4 2 0

OUTPUT.TXT
NO

Exemplul 2

INPUT.TXT
7 4 3 1 5 4 2 1

OUTPUT.TXT
YES ...
0 1 1 1 1 1 0
1 0 1 0 0 0 0
1 1 0 1 0 1 0
1 0 1 0 0 1 0
1 0 0 0 0 0 0
1 0 1 1 0 0 1
0 0 0 0 0 1 0
YES

(BOI, Cipru, 1996)

6. Competiție dificilă

La o competiție au participat N concurenți. Fiecare dintre ei a primit un număr de concurs astfel încât să nu existe concurenți cu același număr. Numerele de concurs aparțin mulțimii $\{1, 2, \dots, N\}$. Din păcate, clasamentul final a fost pierdut, iar comisia își poate aduce aminte doar câteva relații între unii participanți (de genul „participantul cu numărul 3 a ieșit înaintea celui cu numărul 5”).

Şeful comisiei are nevoie de un clasament final și vă cere să-l ajutați determinând primul clasament în ordine lexicografică ce respectă relațiile pe care și le amintește comisia.

Date de intrare

Fișierul de intrare COMPET.IN conține:

- N – numărul concurenților ($N \leq 1000$)
- M – numărul relațiilor pe care și le amintește comisia
- $i_1 j_1$ – concurențul i_1 a fost în clasament înaintea concurențului j_1
- $i_2 j_2$ – concurențul i_2 a fost în clasament înaintea concurențului j_2
- ...
- $i_M j_M$ – concurențul i_M a fost în clasament înaintea concurențului j_M

Date de ieșire

Fișierul de ieșire COMPET.OUT conține pe o singură linie clasamentul sub forma unui șir de numere naturale nenule, separate prin câte un spațiu, reprezentând numerele de concurs ale concurenților, în ordine de la primul clasat la ultimul.

$nr_1 \ nr_2 \ \dots \ nr_N$

Exemplu

COMPET.IN	COMPET.OUT
4	2 1 3 4
2	
2 1	
3 4	

Timp maxim de execuție: 1 secundă/test.

(ONI, Bacău, 2001, clasa a IX-a)

7. Turism

Pe versantul unui munte există n cabane ($n \leq 100$) aflate la altitudini diferite, cabane identificate prin numere naturale între 1 și n . Cabanele sunt legate între ele prin poteci, între două cabane existând cel mult o potecă. Inspectorii fisicali ai ministrului Șpagaton sunt parașutați la câte o cabană, urmând ca fiecare să viziteze una sau mai multe cabane pentru a aduna taxa turistică, urmând numai potecile existente între cabane și numai în sensul coborâtor al acestora.

Știind că pe la fiecare cabană trebuie să treacă *exact* un inspector (nici mai mult nici mai puțin), stabiliți numărul minim necesar de inspectori și traseul parcurs de fiecare astfel încât să se poată aduna taxele de la toate cabanele.

(Lugoj, 2002, clasele a XI-a – a XII-a)

Date de intrare

Fișierul de intrare TURISM.IN conține:

TURISM.IN	Semnificație
$n \ m$	– n numărul de cabane și m numărul de poteci existente între cabane.
$s_1 \ j_1$	– următoarele m linii conțin m perechi de numere despărțite prin câte un spațiu, numerele dintr-o pereche reprezentând două cabane între care există potecă; primul număr din pereche identifică o cabană aflată la altitudine strict mai mare decât cabana identificată de al doilea număr.
$s_2 \ j_2$	
...	
$s_m \ j_m$	

Date de ieșire

Fișierul de ieșire TURISM.OUT va conține:

TURISM.OUT	Semnificație
k	– numărul minim de inspectori necesar
$v_1 \ c_{11} \ c_{12} \ \dots \ c_{1v_1}$	– următoarele k linii conțin două sau mai multe numere naturale, pe fiecare linie i existând un prim număr v_i ce reprezintă numărul de cabane vizitate de inspectorul i și apoi cele v_i cabane vizitate de inspectorul i , în ordinea în care au fost vizitate.
$v_2 \ c_{21} \ c_{22} \ \dots \ c_{2v_2}$	
...	
$v_k \ c_{k1} \ c_{k2} \ \dots \ c_{kv_k}$	

Restricții

- $1 \leq n \leq 100$
- $1 \leq m \leq 5000$
- $1 \leq s_i, j_i \leq n$

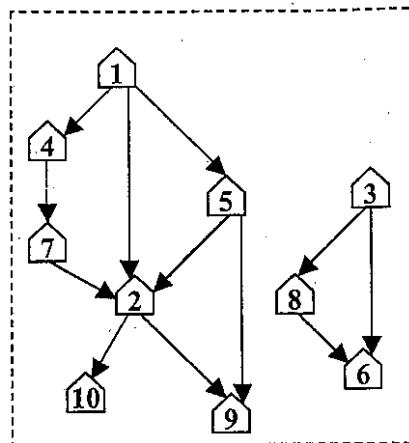
Exemplu

TURISM.IN

10 12
1 4
4 7
1 2
7 2
1 5
5 2
5 9
2 10
2 9
3 8
3 6
8 6

TURISM.OUT

3
5 1 4 7 2 10
2 5 9
3 3 8 6

*8. Castel*

Unui muncitor î se încreză o grea misiune. El are de lăcut toate coridoarele unui castel. Pentru aceasta el primește o hartă în care sunt reprezentate coridoarele și cele N puncte de intersecție ale acestora (numerotate de la 1 la N). Unele puncte de intersecție sunt considerate de muncitor ca fiind „puncte speciale” deoarece ele sunt singurele puncte care îi permit ieșirea și intrarea în castel, prin intermediul unor uși. Deoarece operația de lăcuire trebuie să fie efectuată în cel mai scurt timp, muncitorului î s-a impus ca orice corridor să nu fie traversat decât o singură dată, adică doar atunci când este lăcut. În aceste condiții, muncitorul trebuie să lăcuiască toate coridoarele, folosindu-se eventual de „punctele speciale” ale castelului, plecând din orice intersecție de coridoare dorește și terminând operația în același punct.

Determinați un traseu ciclic prin care operația este satisfăcută.

Date de intrare

Fișierul de intrare CASTEL.IN conține pe prima linie numărul N, iar pe cea de-a doua linie numărul de coridoare M. Pe următoarele M linii perechi de două numere reprezentând perechi de puncte între care există un corridor ce trebuie lăcut. Pe următoarea linie numărul P reprezentând numărul de „puncte speciale”, iar pe ultima linie sirul „punctelor speciale” separate prin câte un spațiu.

Datele de ieșire

Ieșirea se va face în fișierul CASTEL.OUT în formatul următor:

Pe prima linie se va scrie mesajul DA sau NU, după cum există sau nu un traseu ciclic care respectă cerințele. În caz afirmativ fișierul va conține pe a doua linie un sir de numere reprezentând punctele în ordinea de apariție pe traseu, despărțite după caz de secvență de caractere „ - ” (spațiu minus spațiu) sau „ * ” (spațiu asterisc spațiu). O secvență de tipul i - j semnifică faptul că muncitorul a traversat și lăcut corridorul dintre punctele i și j. O secvență de tipul i * j semnifică faptul că muncitorul a părăsit castelul prin „punctul special” i și a revenit în castel prin „punctul special” j.

Restriții

- $1 \leq N \leq 5000$
- $0 \leq M \leq 10000$

Exemplu

CASTEL.IN

6
6
1 2
2 3
3 4
4 1
2 5
4 6
5
1 2 5 4 6

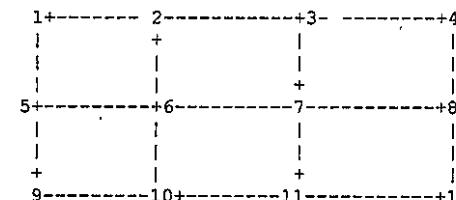
CASTEL.OUT poate conține:

DA
1 - 2 - 5 * 6 - 4 - 3 - 2 * 4 - 1

(ONI, Constanța, 2000, clasa a X-a)

9. Bariere

Un șoricel se află situat într-un nod al unei rețele dreptunghiulare de dimensiune $m \times n$, având forma și numerotarea nodurilor conform figurii de mai jos (în care $m=4$, $n=3$):



Fiecare nod V al rețelei are *exact* o barieră pe o muchie VW, care blochează trecerea șoricelului de la V la W, dar și de la W la V.

Pentru exemplul din figura anterioară, în nodul 2 avem o barieră către nodul 6, care împiedică trecerea șoricelului de la nodul 2 la nodul 6, dar și de la nodul 6 la nodul 2.

Șoricelul trebuie să ajungă la o bucătică de cașcaval, situată într-un alt nod al rețelei, parcurgând un drum de cost minim, respectând următoarele reguli:

– șoricelul, aflat în nodul V, poate trece la nodul W, dacă nu există nici o barieră pe muchia VW; această trecere îl costă 1\$.

– șoricelul poate schimba poziția barierei din nodul curent, ceea ce îl costă tot 1\$.

Pentru exemplul din figura anterioară, șoricelul (presupus a fi inițial în nodul 2) poate ajunge în nodul 7, în mai multe moduri, de pildă:

a. mută bariera din 2 (așezând-o către nodul 1), se deplasează apoi în nodul 6, apoi în 7 (costul: 3\$);

b. mută bariera din 2 (așezând-o către nodul 1), se deplasează apoi în nodul 6, apoi în nodul 10, unde pune bariera către nodul 9, apoi se duce în 11, pune bariera de aici către nodul 10 și, în sfârșit, se deplasează în nodul 7 (costul: 7\$).

Se cere să se determine drumul de cost minim al șoricelului către cașcaval.

Date de intrare

Fișierul de intrare BARIERE.IN are forma următoare:

$n \ m$ – dimensiunile matricei ($1 \leq n, m \leq 50$)

s – nodul de unde pleacă șoriceul

c – nodul unde se află cășcavalul

$d_{11} \ d_{12} \ d_{13} \dots \ d_{1m}$

$d_{21} \ d_{22} \ d_{23} \dots \ d_{2m}$

\dots

$d_{n1} \ d_{n2} \ d_{n3} \dots \ d_{nm}$

unde d_{ij} este poziția inițială a barierelor din nodul de pe linia i și coloana j , codificată prin unul din caracterele N, S, E, V, reprezentând punctul cardinal către care este pusă bariera.

Date de ieșire

Fișierul de ieșire BARIERE.OUT conține pe prima linie costul drumului minim al șoriceului, iar pe cea de a doua linie, drumul minim, codificat astfel:

$v_1 \ p_1 \ v_2 \ p_2 \dots \ v_k \ p_k$

unde v_1, v_2, \dots, v_k sunt vârfurile prin care trece șoriceul, inclusiv extremitățile; p_i ($1 \leq i \leq k$) este punctul cardinal în care se află bariera din nodul v_i , la părăsirea acestuia.

Exemplu

Pentru figura anterioară, $m=4, n=3, s=1, c=7$:

BARIERE.IN	BARIERE.OUT
4 3	3
2	2 E 6 V 7 N
7	
E S V V	
E V N V	
N E N V	

(ONI, Suceava, 1996, clasa a XI-a)

10. Statui

Primarul unui oraș dorește să amplaseze cât mai multe statui, cel mult una într-o piață. Orașul este vizitat de diverse oficialități care sosesc la aeroport și trebuie transportate la primărie pe un drum de lungime minimă. Primarul dorește să poată transporta oaspeții astfel încât fie să vadă toate statuile, fie să nu vadă nici o statuie, cele două trasee fiind disjuncte.

Cunoscând rețeaua stradală și pozițiile aeroportului și primăriei în această rețea, să se determine unde trebuie plasate statuile și care sunt cele două trasee, astfel încât numărul statuilor să fie maxim.

(Baraj, Cluj, 1996)

Date de intrare

Datele de intrare se citesc din fișierul STATUI.IN care conține:

- pe prima linie numărul n de piețe din oraș; primăria se află în piață cu numărul 1, aeroportul în piață cu numărul n ($n \leq 150$);
- pe următoarele n linii se găsesc câte n numere întregi nenegative; al j -lea număr de pe a i -a linie reprezintă lungimea străzii de la piață având numărul i la piață cu

numărul j , sau 0 dacă $i=j$ sau dacă nu există stradă. Datorită sensurilor unice, se poate că distanța de la i la j să nu fie egală cu distanța de la j la i .

Datele din fișierul de intrare asigură existența cel puțin a unui drum între aeroport și primărie.

Date de ieșire

Fișierul de ieșire STATUI.OUT va conține:

- pe prima linie numărul s de statui urmat de lista piețelor în care urmează a fi amplasate;
- pe a doua linie numărul de piețe de pe drumul fără statui, urmat de lista piețelor, în ordine, de la aeroport la primărie (inclusiv capetele);
- pe a treia linie numărul de piețe de pe drumul ce vizitează toate statuile, urmat de lista piețelor, în ordine, de la aeroport la primărie (inclusiv capetele).

Se cere o singură soluție.

Exemplu

STATUI.IN
5
0 2 0 3 0
2 0 2 2 0
0 2 0 1 2
3 1 0 0 3
0 0 2 3 0

STATUI.OUT
2 2 3
3 5 4 1
4 5 3 2 1

11. James Bond

Un grup de teroriști s-au ascuns într-un sistem de canalizare. James Bond, având la dispoziție o hartă care descrie configurația sistemului de canalizare și pozițiile în care teroriștii au amplasat bombe, vrea să anihileze grupul de teroriști.

Conducetele din acest sistem sunt desenate pe hartă sub formă unor segmente de dreaptă în plan, astfel încât oricare două segmente au cel mult un punct comun, care este capăt pentru amândouă segmentele.

Explozia unei bombe afectează o zonă circulară (interior și frontieră) și este instantanea pe întreaga zonă afectată. Bombele pot să difere una de cealaltă prin raza de acțiune sau prin timpul la care explodează. În urma unei explozii, canalele care trec prin zona de acțiune a bombei devin impracticabile pe segmentul afectat, dar, dacă Bond se află într-un astfel de canal și a trecut deja de zona afectată de explozie, poate să-și continue drumul. Dacă James Bond se află în raza de acțiune a unei bombe la momentul exploziei, el moare!

La momentul inițial (momentul 0), James Bond pornește dintr-un punct al sistemului de canalizare (capăt al unui segment) și se deplasează cu viteză constantă, egală cu 1. James Bond trebuie să ajungă în punctul în care se află teroriștii (capăt al unui segment) în cel mai scurt timp posibil.

Determinați calea pe care trebuie să-o urmeze James Bond astfel încât să ajungă viu la teroriști în cel mai scurt timp.

Date de intrare

Fișierul de intrare BOND.IN conține:

BOND.IN

M N
 X₁₁ Y₁₁ X₁₂ Y₁₂
 X₂₁ Y₂₁ X₂₂ Y₂₂
 ...
 X_{M1} Y_{M1} X_{M2} Y_{M2}
 X_{1c} Y_{1c} R₁ T₁

X_{2c} Y_{2c} R₂ T₂

...

X_{NC} Y_{NC} R_N T_N

XP YP

XD YD

Date de ieșire

Fișierul de ieșire BOND.OUT va conține:

BOND.OUT

TMIN

C₁₁ C₁₂C₂₁ C₂₂

...

Semnificație

tempul minim în care James Bond ajunge la teroriști;
 pe următoarele linii, până la sfârșitul fișierului se vor scrie câte două numere întregi, separate printr-un spațiu, reprezentând coordonatele capetelor de segmente prin care va trece James Bond.

Restriții

- 1 ≤ M ≤ 100
- 0 ≤ N ≤ 50
- 1 ≤ R ≤ 30000
- 0 ≤ T ≤ 30000
- Coordonatele capetelor segmentelor și ale pozițiilor bombelor aparțin intervalului [0, 30000, 30000].

Timp maxim de execuție: 1 secundă/test.

Exemplu

BOND.IN

3 1
 0 0 10 0
 0 0 10 10
 10 0 10 10
 6 4 2 1
 0 0
 10 10

BOND.OUT

20
 0 0
 10 0
 10 10

(Baraj, Bacău, 2001)

Observații

În procesul de verificare se iau în considerare partea sănătreagă și primele 2 cifre zecimale. Pentru datele de test se garantează existența soluției.

Semnificație

numărul de conducte și numărul de bombe;

M linii ce conțin fiecare câte patru numere întregi, separate prin câte un spațiu, reprezentând descrierea segmentelor prin coordonatele extremităților;

N linii ce conțin fiecare câte patru numere întregi, separate prin câte un spațiu, reprezentând descrierea bombelor prin coordonatele punctului unde sunt amplasate, raza cercului pe care acționează și momentul de timp la care acționează;

coordonatele punctului de plecare, capăt al unui segment;
 coordonatele punctului în care se află teroriștii, capăt de segment.

Structuri arborescente**I. Secvența gradelor unui arbore**

Fie $d = (0 < d_1 \leq d_2 \leq \dots \leq d_n)$ o secvență de numere naturale. Verificați dacă secvența dată poate constitui secvența gradelor vârfurilor unui arbore și dacă da, determinați un astfel de arbore.

Soluție

Vom demonstra următorul rezultat:

Numerele întregi $0 < d_1 \leq d_2 \leq \dots \leq d_n$ sunt gradele vârfurilor unui arbore dacă și numai dacă $d_1 + d_2 + \dots + d_n = 2n - 2$.

Condiția este necesară, deoarece orice arbore cu n vârfuri are $n-1$ muchii, iar suma gradelor vârfurilor oricărui graf este dublul numărului de muchii. Deci $d_1 + d_2 + \dots + d_n = 2n - 2$.

Să demonstrăm că această condiție este suficientă. Fie $0 < d_1 \leq d_2 \leq \dots \leq d_n$ astfel încât $d_1 + d_2 + \dots + d_n = 2n - 2$. Demonstrăm inducitiv că există un arbore cu gradele vârfurilor d_1, d_2, \dots, d_n .

Pentru $n=1$, $d_1=0$ (arborele format dintr-un vârf izolat), iar pentru $n=2$, $d_1+d_2=2$, deci $d_1=d_2=1$ (arborele format din două vârfuri adiacente).

Presupunem acum că proprietatea este adevărată pentru orice secvență de n numere naturale $0 < d_1 \leq d_2 \leq \dots \leq d_n$, astfel încât $d_1 + d_2 + \dots + d_n = 2n - 2$ și demonstrăm că pentru orice secvență $0 < d'_1 \leq d'_2 \leq \dots \leq d'_{n+1} \leq d'_n$ astfel încât $d'_1 + d'_2 + \dots + d'_{n+1} = 2n$, există un arbore cu $n+1$ vârfuri cu secvența gradelor $d'_1, d'_2, \dots, d'_{n+1}$.

Observăm că există măcar un nod terminal x_1 cu gradul $d'_1=1$, altfel dacă $d_i \geq 2, \forall i \in \{1, 2, \dots, n+1\} \Rightarrow d'_1 + d'_2 + \dots + d'_{n+1} \geq 2(n+1)$, ceea ce contrazice ipoteza. În mod analog, observăm că există măcar un nod neterminal x_{n+1} , cu gradul $d'_{n+1} > 1$, altfel dacă $d'_{n+1}=1, \forall i \in \{1, 2, \dots, n+1\} \Rightarrow d'_1 + d'_2 + \dots + d'_{n+1} = n+1 < 2n$.

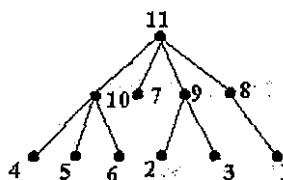
Să considerăm următoarea secvență de n numere întregi $d'_2, \dots, d'_{n+1}, d'_{n+1}-1$ cu proprietatea că $d'_2 + \dots + d'_{n+1} + d'_{n+1}-1 = 2n - 2$. Din ipoteza inductivă există un arbore A_n cu n vârfuri și secvența gradelor $d'_2, \dots, d'_{n+1}, d'_{n+1}-1$. Adăugăm la arborele A_n un vârf pe care îl unim printr-o muchie cu vârful având gradul $d'_{n+1}-1$. Obținem un arbore A_{n+1} cu gradele vârfurilor $d'_1, d'_2, \dots, d'_{n+1}$.

Demonstrația acestei teoreme oferă și o soluție constructivă pentru obținerea unui arbore cu secvența gradelor dată.

De exemplu, pentru $n=11$ și sirul gradelor:

1	2	3	4	5	6	7	8	9	10	11
1	1	1	1	1	1	1	2	3	4	4

obținem arborele din figură.



Vom reține gradele vârfurilor într-un vector d , ordonat crescător, cu suma componentelor egală cu $2n-2$. Pentru a reține arborele, este suficient să reținem pentru fiecare vârf terminal „eliminat” din arbore, vârful neterminal cu care este adiacent.

```

#define NMaxVf 100
int a[NMaxVf], d[NMaxVf], n;
  
```

Spre deosebire de ideea din demonstrație, pentru a conserva ordinea în vectorul d , la fiecare pas al algoritmului gradul care va fi decrementat va fi primul grad mai mare ca 1 și nu ultimul.

```

int s=0, i, VfT, VfNt;
for (i=1; i<=n; i++) s+=d[i];
if (s!=2*(n-1))
    fout<<" Suma gradelor trebuie sa fie 2(n-1)!\n";
else
    for (VfT=1; VfT<n; VfT++)
        //determin primul varf neterminal
        for (VfNt=VfT+1; VfNt<n && d[VfNt]==1; VfNt++);
        a[VfT]=VfNt;
        d[VfNt]--;
  
```

Observație

Acest algoritm constructiv sugerează o metodă foarte eficientă de reprezentare a arborilor. Dacă A_n este un arbore cu vârfurile x_1, x_2, \dots, x_n , suprimăm vârful terminal cu cel mai mic indice și muchia incidentă cu acesta și reținem a_1 , vârful adiacent cu vârful terminal suprimat. Am obținut astfel un subgraf cu $n-2$ muchii conex și aciclic, deci un arbore A_{n-1} . Repetăm procedeul pentru arboarele A_{n-1} , determinând un alt doilea vârf, a_2 , adiacent cu vârful terminal de indice minim ce va fi eliminat din A_{n-1} împreună cu muchia incidentă cu el și a.m.d., până când se obține un arbore A_2 cu două vârfuri adiacente.

Am obținut astfel un sistem $\{a_1, a_2, \dots, a_{n-2}\}$ de $n-2$ numere ($1 \leq a_i \leq n$, $\forall i \in \{1, 2, \dots, n-2\}$) asociat arborelui A_n numit *codul Prüffer* al lui A_n .

Se poate demonstra că există o corespondență biunivocă între mulțimea arborilor A cu n vârfuri și mulțimea sistemelor $\{a_1, a_2, \dots, a_{n-2}\}$, $a_i \in \{1, 2, \dots, n\}$, $\forall i \in \{1, 2, \dots, n-2\}$.

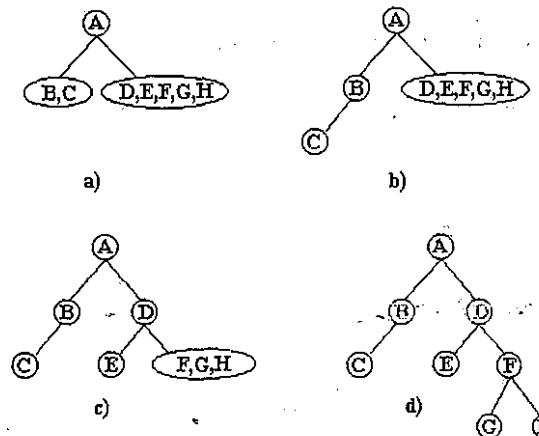
Folosind acest rezultat, deducem că numărul arborilor ce se pot construi cu n vârfuri date este egal cu n^{n-2} (numărul funcțiilor definite pe o mulțime cu $n-2$ elemente, cu valori într-o mulțime cu n elemente). Această formulă poartă numele de *formula lui Cayley*.

2. Creare arbore binar

Se dau secvențele obținute prin parcurgerile în preordine și în inordine ale unui arbore binar. Scrieți un program care să determine arborele binar corespunzător.

Soluție

De exemplu, fie parcurgerea în preordine A, B, C, D, E, F, G, H și parcurgerea în inordine C, B, A, E, D, G, F, H . Analizând parcurgerea în preordine, deducem că nodul A este rădăcina. De asemenea, analizând parcurgerea în inordine, deducem că nodurile C, B vor constitui arborele stâng, iar D, E, F, G, H subarborele drept. Pentru a determina structura subarborelui stâng, respectiv a subarborelui drept, procedăm analog: din parcurgerea în preordine deducem că B este rădăcina subarborelui stâng, iar din parcurgerea în inordine deducem că C este fiul stâng al lui B . În mod similar, pentru subarborele drept deducem din parcurgerea în preordine că D este rădăcină, iar din parcurgerea în inordine că subarborele stâng al lui D este format numai din nodul E , iar subarborele drept al lui D din nodurile F, G, H . Procedeul se repetă până când obținem întreg arborele. Succesiunea operațiilor este ilustrată în figura următoare:



Propoziție

Succesiunile de noduri obținute prin parcurgerile în inordine și în preordine ale unui arbore binar definesc în mod unic structura arborelui.

Demonstrație

Vom proceda prin inducție completă după numărul de noduri.

$P(1)$: Dacă arboarele are un singur nod, rădăcina, afirmația este evidentă.

$P(n)$: Presupunem că pentru $\forall k \in \{1, 2, \dots, n\}$ afirmația este adeverită, adică pentru orice pereche de secvențe inordine – preordine de lungime k , arborele binar corespunzător este unic.

$P(n+1)$: Să demonstrăm că orice pereche de secvențe preordine-inordine de lungime $n+1$ determină în mod unic un arbore binar.

Să considerăm o pereche de secvențe preordine-inordine de lungime $n+1$. Primul nod din parcurgerea în preordine este în mod necesar rădăcina arborelui, celelalte n noduri fiind distribuite în subarbore: toate nodurile situate în stânga rădăcinii în parcurgerea în ordine vor constitui subarborele stâng, nodurile situate în dreapta rădăcinii în parcurgerea inordine vor constitui subarborele drept.

Obținem două perechi de secvențe preordine-inordine de lungime cel mult n , care din ipoteza inductivă, determină în mod unic subarborele stâng, respectiv cel drept și, în consecință, cum rădăcina este în mod unic determinată, deducem că perechea de secvențe preordine-inordine de lungime $n+1$ determină în mod unic un arbore binar cu n noduri.

Vom reprezenta arborele binar prin referințe descendente. Mai exact, un nod din arbore va fi o structură în care vom reține informația asociată nodului (în acest caz numărul de ordine al nodului) și doi pointeri st , dr , care reprezintă adresa rădăcinii subarborelui stâng al nodului respectiv și dr , care reprezintă adresa rădăcinii subarborelui drept.

```
struct NodArbore
{
    int inf;
    NodArbore * st, *dr;
} typedef NodArbore * ArboreBinar;
```

Vom descrie o funcție recursivă `ConstrArb()`, care determină arborele binar corespunzător unei perechi de secvențe preordine-inordine date. Pentru simplificare, vom considera că nodurile arborelui sunt numerotate în preordine de la 1 la n . Astfel, este suficient să reținem într-un vector global `Ino` indicii vârfurilor în ordinea în care au fost atinse în ordine. Inițial, apelăm `ConstrArb(1, 1, n)`.

```
ArboreBinar ConstrArb (int rad, int st, int dr)
/* functia intoarce radacina arborelui unic determinat de
parcurgerile inordine-preordine;
rad este indicele radacinii arborelui; st si dr sunt
limitele intre care se gaseste parcurgerea inordine a
arborelui in vectorul Ino */
if (st>dr) return NULL;
ArboreBinar r;
r=new NodArbore; //aloc memorie pentru radacina arborelui
r->inf=rad; //retin informatia -> numarul asociat nodului
// determin pozitia radacinii in parcurgerea inordine
for (int IPozRad=st; Ino[IPozRad]!=rad; IPozRad++);
if (IPozRad == st) //subarborele stang este vid
    r->st = NULL;
else
    // Ino[ st.. IPozRad-1] contine subarborele stang
    r->st = ConstrArb(rad+1, st, IPozRad-1);
if (IPozRad == dr) //subarborele drept este vid
    r->dr=NULL;
else
    // Ino[ IPozRad+1.. dr] contine subarborele drept
    r->dr = ConstrArb(rad+IPozRad-st+1, IPozRad+1, dr);
    // in subarborele stang au fost IPozRad-st+1 varfuri
return r; }
```

3. Arbori binari m-ponderați

Un arbore binar strict se numește *m-ponderat* dacă fiecare nod i are asociată o pondere $p[i]$, cu valori între 1 și $m-1$, astfel încât pentru orice nod terminal t suma ponderilor din nodurile aflate pe drumul de la rădăcină la nodul t este egală cu m . Definim $P_{n,m}(T)$ ponderea unui arbore binar strict cu n noduri terminale m -ponderat ca fiind suma ponderilor atașate nodurilor din T . Să se scrie un program care pentru n și m să determine un arbore binar strict m -ponderat cu n noduri terminale T^* astfel încât:

$$P_{n,m}(T^*) = \max\{P_{n,m}(T) \mid T \text{ arbore binar strict } m\text{-ponderat cu } n \text{ noduri terminale}\}$$

(OJI, Iași, 1995, clasa a XI-a)

Soluție

Pentru ca problema să admită soluție trebuie ca pentru orice drum de la rădăcină la un nod terminal să putem asocia nodurilor ponderi ≥ 1 . Înălțimea minimă a unui arbore binar cu n vârfuri terminale este $h=[\log_2 n]$. Pentru a asocia ponderi este necesar ca m , suma ponderilor vârfurilor de pe orice drum de la rădăcină la un vârf terminal, să fie cel puțin egală cu $[\log_2 n]+1$. Deci condiția necesară pentru ca problema să admită soluție este $m \geq [\log_2 n]+1$.

Observații

A. Fie T un arbore binar strict cu n vârfuri terminale. Construim o m -ponderare p_o a arborelui T astfel:

- asociem fiecărui nod interior ponderea 1;
- deoarece suma ponderilor de pe orice drum de la rădăcină la un vârf terminal trebuie să fie egală cu m , asociem fiecărui nod terminal o pondere egală cu m -lungimea drumului de la rădăcină la nodul terminal respectiv.

Demonstrăm că p_o este o m -ponderare maximală pentru arborele T .

Să considerăm p_o o m -ponderare oarecare pentru arborele T și x un vârf interior astfel încât $p[x]>1$. Dacă există mai multe astfel de noduri, considerăm un vârf de pe un nivel de rang minim. Fie y și z cei doi fi ai lui x . Construim o altă m -ponderare p' a lui T astfel:

$$p'[x]=1; p'[y]=p[y]+p[x]-1; p'[z]=p[z]+p[x]-1.$$

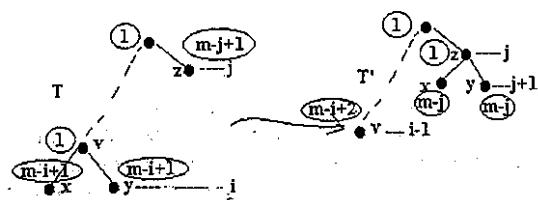
$$\text{Atunci } P'(T)=P(T)-p[x]-p[y]-p[z]+p'[x]+p'[y]+p'[z] \Rightarrow$$

$$P'(T)=P(T)-p[x]-p[y]-p[z]+1+ p[y]+p[x]-1+ p[z]+p[x]-1 \Rightarrow$$

$$P'(T)=P(T)+p[x]-1 \Rightarrow P'(T) > P(T).$$

Modificând succesiv ponderile nodurilor interioare de sus în jos, obținem după un număr finit de pași m -ponderarea p_o , în care toate vârfurile interioare au ponderea 1. Deci $P_o(T) \geq P(T), \forall p$ o m -ponderare a arborelui T .

B. Observația A. oferă o modalitate de m -ponderare optimă a unui arbore binar strict dat. Rămâne să determinăm, pentru n și m dați, arborele binar strict cu n vârfuri terminale care maximizează $P_{n,m}(T)$. Demonstrăm că arborele căutat T^* este arborele binar complet. Fie T un arbore binar strict cu n vârfuri terminale care nu este complet și fie x și y două vârfuri terminale fii ai aceluiași nod interior, situate pe nivelul i , iar z un nod terminal situat pe nivelul j , astfel încât $i-j>1$.



Mutăm nodurile x și y de pe nivelul i pe nivelul $j+1$, ca fi să fie nodului z și reponderăm nodurile.

$$P(T') = P(T) - 2(m-i+1) - 1 - (m-j+1) + 2(m-j) + m-i+2+1 \Rightarrow \\ P(T') = P(T) + i - j - 1 > P(T).$$

Aplicăm succesiv această transformare până când obținem un arbore binar complet. Deci $P(T^*) > P(T)$, $\forall T$ arbore binar strict.

Tinând cont de observațiile A și B, algoritmul se reduce la determinarea unui arbore complet cu n noduri și ponderarea acestora conform regulii A:

```
#include <math.h>
#define NMaxVf 100
void main()
{int p[NMaxVf], n, m, nivel;
const float ln2=log(2);
cin>>n>>m;
if (m<log(n)/ln2) cout<<"Nu există soluție";
else
    //nodurile interioare au ponderea 1
    for (int i=1; i<n; i++) p[i]=1;
    for (i=n; i<2*n; i++)
        {nivel=log(i)/ln2;
        p[i]=m-nivel;}
}
for (int i=1; i<2*n; i++) cout<<p[i]<<' ';
```

4. Reprezentarea mulțimilor disjuncte

Definiție

Fie mulțimea $U=\{1, 2, \dots, n\}$, $n \geq 1$. Sistemul $S=\{S_1, S_2, \dots, S_k\}$, $k \geq 1$, constituie o partitie a mulțimii U dacă sunt îndeplinite următoarele condiții:

1. $S_i \subseteq U$, $\forall i \in \{1, 2, \dots, k\}$
2. $S_i \neq \emptyset$, $\forall i \in \{1, 2, \dots, k\}$
3. $S_i \cap S_j = \emptyset$, $\forall i \neq j$, $i, j \in \{1, 2, \dots, k\}$
4. $S_1 \cup S_2 \cup \dots \cup S_k = U$.

Dată fiind mulțimea U și $S=\{S_1, S_2, \dots, S_k\}$ o partitie a mulțimii U , problema constă în a proiecta o structură de date care să permită executarea eficientă a următoarelor două operații fundamentale:

- $\text{find}(x)$: determină mulțimea $S_i \subseteq S$ căreia îl aparține elementul x , $x \in U$;
- $\text{union}(S_i, S_j)$: unește mulțimea S_i cu mulțimea S_j , obținându-se un nou element al mulțimii S , ce va înlocui S_i și S_j .

Observație

Această problemă este cunoscută și sub denumirea de *problema claselor de echivalență* (x, y se vor numi echivalente dacă aparțin aceleiași mulțimi $S_i \subseteq S$, sau, altfel spus, $\text{find}(x)=\text{find}(y)$), concepție de relație de echivalență și partiție reprezentând două abordări diferite ale aceleiași structuri matematice.

De exemplu, dat fiind un graf neorientat, componentele conexe ale grafului constituie o partiție a mulțimii vârfurilor grafului. Un algoritm de determinare a componentelor conexe ale unui graf neorientat poate fi formulat cu ajutorul operațiilor *union-find* astfel:

Pentru $\forall i \in \{1, 2, \dots, n\}$, $S_i = \{i\}$;

Pentru $\forall \{i, j\}$ muchie în graf

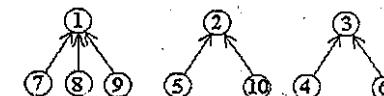
$A = \text{find}(i);$

$B = \text{find}(j);$

dacă $A \neq B$ atunci $\text{union}(A, B)$;

O soluție eficientă este reprezentarea mulțimilor disjuncte cu ajutorul arborilor cu rădăcină. Fiecare arbore reprezintă o mulțime, iar fiecare nod din arbore un element al mulțimii. Rădăcina arborelui va fi elementul reprezentativ al mulțimii.

De exemplu, să considerăm $n=10$ și o partiție a mulțimii $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ formată din $S_1=\{1, 7, 8, 9\}$, $S_2=\{2, 5, 10\}$, $S_3=\{3, 4, 6\}$. Partitia $S=\{S_1, S_2, S_3\}$ poate fi reprezentată ca o pădure formată din trei arbori:

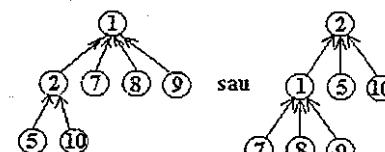


Reprezentăm arborii prin referințe ascendente, deci pentru fiecare nod din arbore, cu excepția rădăcinii, reținem legătura spre părintele său: $\text{tata}(x) = \text{nodul} \text{ părinte} \text{ al lui } x$, sau 0, dacă x este rădăcina arborelui.

Operația $\text{find}(x)$ constă în a determina rădăcina arborelui al căruia nod este x .

```
int find(int x)
{ while (tata[x]) x=tata[x];
  return x; }
```

Operația *union* se poate realiza transformând unul dintre arbori în subarborele celuilalt. Reuniunea $S_1 \cup S_2$ poate fi reprezentată în două moduri:



Deci rădăcina unuia dintre arbori devine părintele rădăcinii celuilalt arbore.

```
void union(int i, int j)
{tata[i]=j;}
```

Acești algoritmi sunt foarte simpli, dar nu sunt eficienți.

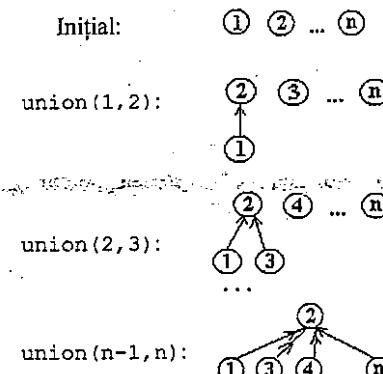
Să considerăm, de exemplu partitia $S=\{1, \{2\}, \dots, \{n\}\}$. Deci configurația inițială constă dintr-o pădure în care fiecare arbore este format doar din rădăcină: $tata(i)=0$, $\forall i \in \{1, 2, \dots, n\}$. Să considerăm acum următoarea secvență de operații union și find : $\text{union}(1, 2)$, $\text{find}(1)$, $\text{union}(2, 3)$, $\text{find}(1)$, $\text{union}(3, 4)$, $\text{find}(1)$, ..., $\text{find}(1)$, $\text{union}(n-1, n)$.

Această secvență conduce la arborele degenerat din figura alăturată.

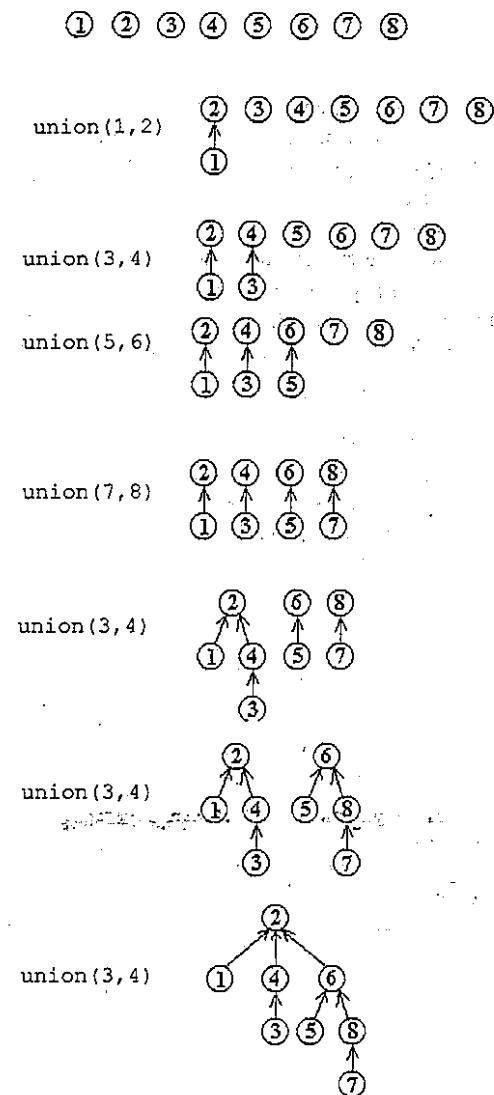
Cum timpul necesar operației union este constant, cele $n-1$ operații union au timpul de execuție de $O(n)$. Dar fiecare operație $\text{find}(1)$ parcurge tot drumul de la nodul 1 până la rădăcină. Cum timpul de execuție necesar operației find pentru un nod de pe nivelul i este $O(i)$, obținem un timp de execuție de $O(n^2)$ pentru cele $n-2$ operații $\text{find}(1)$.

Puteam îmbunătăți eficiența operațiilor union și find , aplicând următoarea *regulă de ponderare* pentru $\text{union}(i, j)$: dacă numărul de niveluri din arborele cu rădăcina i este mai mic decât numărul de niveluri din arborele cu rădăcina j , atunci j va deveni părintele lui i , altfel i va deveni părintele lui j .

Utilizând regula de ponderare secvența de operații union și find de mai sus va conduce la următorii arbori:



În acest caz timpul de execuție necesar operațiilor find este $O(n)$, deoarece arborii obținuți au înălțimea cel mult egală cu 1. Totuși, există cazuri mai defavorabile. De exemplu, fie $n=8$. Inițial vom avea pădurea:



Înălțimea arborelui obținut este $3 = \log_2 8$.

Lemă

Dacă A este un arbore cu n noduri obținut în urma aplicării operației union folosind regula de ponderare, înălțimea arborelui A este cel mult egală cu $\lceil \log_2 n \rceil$.

Demonstrație

Vom proceda prin inducție după n , numărul de noduri. Pentru $n=1$, rezultatul este evident.

Presupunem afirmația adevărată pentru arbori cu i noduri ($1 \leq i \leq n-1$), obținuți în urma aplicării algoritmului union. Să demonstrăm că înălțimea oricărui arbore cu n noduri A_n , obținut în urma aplicării algoritmului union este cel mult egală cu $\lceil \log n \rceil$.

Fie $\text{union}(j, k)$ ultima operație union executată pentru obținerea arborelui A_n . Notăm cu m numărul de noduri din arborele cu rădăcina j . Arborele cu rădăcina k are $m-n$ noduri. Putem presupune, fără a restrângere generalitatea, că $1 \leq m \leq n/2$. Deci înălțimea h a arborelui A_n va fi egală cu înălțimea arborelui cu rădăcina k sau cu o unitate mai mare decât înălțimea arborelui cu rădăcina j .

În primul caz: $h \leq \lceil \log(n-m) \rceil \leq \lceil \log n \rceil$.

În cel de-al doilea caz: $h \leq \lceil \log m \rceil \leq \lceil \log(n/2) \rceil + 1 \leq \lceil \log n \rceil$.

Pentru a aplica regula de ponderare este necesar ca pentru fiecare arbore să cunoaștem numărul de niveluri. Fie h , un vector în care $h[i]$ reprezintă numărul de niveluri din arborele cu rădăcina i . Acest vector va fi actualizat eventual la operații union.

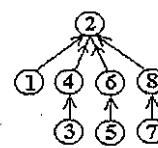
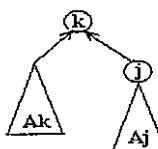
```
void union_ponderare(int x, int y)
{
    if (h[x] > h[y]) tata[y] = x;
    else
        {tata[x] = y;
         if (h[x] == h[y]) h[y]++;
        }
}
```

Lema 1 garantează un timp de execuție de $O(\log n)$ pentru operația $\text{find}(x)$, pentru orice nod x dintr-un arbore obținut prin operații union folosind regula de ponderare.

Pentru a îmbunătăji în continuare timpul de execuție a operației $\text{find}(x)$, vom utiliza următoarea *regulă de compresie*: orice nod y de pe drumul de la rădăcina arborelui la nodul x va deveni fiu al rădăcinii arborelui.

```
int find_compresie(int x)
{
    int r=x;
    while (tata[r]) r=tata[r];
    // r este acum radacina arborelui
    int y=x;
    while (y!=r)           //aplica regula de compresie
        {t=tata[y];
         tata[y]=r;
         y=t;}
    return r;
}
```

Funcția $\text{find}_\text{compresie}()$ parcurge drumul de la nodul x la rădăcină de două ori, o dată pentru determinarea rădăcinii, și două oară pentru comprimarea drumului de la x la rădăcină. Totuși, această modificare reprezintă o îmbunătățire, deoarece într-o secvență de operații union-find, timpul total de execuție va fi mai mic.



Să considerăm arborele obținut prin secvență de operații union din exemplul precedent Executând prima oară operația $\text{find}(8)$ obținem arborele alăturat:

Au fost necesare trei deplasări pe legături ascendentă pentru a identifica rădăcina arborelui și apoi pentru compresia drumului de la rădăcină la nodul 8 altă două deplasări. Dar următoarele apeluri ale funcției $\text{find}(8)$ vor necesita o singură deplasare până la rădăcina arborelui. Astfel costul total al unei secvențe de operații $\text{find}(8)$ va fi mai mic.

5. Cozi cu prioritate

O coadă cu prioritate este o structură de date abstractă formată din elemente care au asociată o valoare numită cheie sau prioritate și care suportă următoarele operații:

- $\text{Insert}(Q, x)$: inserează elementul x în coada cu prioritate Q ;
- $\text{ExtractMax}(Q)$: extrage din coada cu prioritate Q elementul de valoare maximă.

Observație

În mod analog se poate defini o coadă cu min-prioritate, pentru care interesează operația de extragere din coadă a elementului de prioritate minimă.

Există multe aplicații ale cozilor cu prioritate. De exemplu, planificarea execuției programelor pe un calculator: coada cu prioritate reține programele ce trebuie executate în funcție de prioritățile lor relative. Când se încheie sau se întrerupe execuția unui program, programul cu cea mai mare prioritate din coadă este selectat și lansat în execuție (operația ExtractMax). Adăugarea unui nou program în coadă se realizează cu operația Insert .

Există diverse modalități de a implementa o coadă cu prioritate: ca un vector, ca o listă înlinățuită, ordonată sau nu etc. Fiecare din aceste variante optimizează una din cele două operații, cealaltă realizându-se în timp liniar.

O structură de date ce permite implementarea ambelor operații în timp logaritmice este *heap-ul*.

Min-heap, max-heap

Un *max-heap* este un arbore binar complet în care valoarea memorată în orice nod al său este mai mare sau egală decât valorile memorate în nodurile fiilor ai acestuia. Corespunzător se poate defini *min-heap*-ul, ca un arbore binar complet în care valoarea memorată în orice nod este mai mică sau egală decât valorile memorate în nodurile fiilor ai acestuia.

În cele ce urmează, un *max-heap* va fi numit *heap*, urmând ca atunci când este vorba despre un *min-heap*, aceasta să se preciseze în mod explicit.

Un *heap* fiind un arbore binar complet, reprezentarea cea mai adecvată este reprezentarea secvențială. Deci este suficient să reținem într-un vector informațiile asociate nodurilor, relațiile dintre noduri fiind implicate:

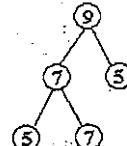
- fiul stâng al nodului x este $2x$, dacă $2x \leq n$ (n fiind numărul de noduri din arbore) și, evident, nu există dacă $2x > n$;
- fiul drept al nodului x este $2x+1$, dacă $2x+1 \leq n$ și nu există dacă $2x+1 > n$;
- tatăl nodului x este $[x/2]$, dacă $x > 1$ și nu există dacă $x = 1$.

Utilizând reprezentarea secvențială, putem reformula definiția unui *heap* astfel:

Tabloul $H[n]$ formează un *max-heap* dacă $H[i] \geq H[i/2]$, $\forall i \in \{2, 3, \dots, n\}$.

Tabloul $H[n]$ formează un *min-heap* dacă $H[i] \leq H[i/2]$, $\forall i \in \{2, 3, \dots, n\}$.

De exemplu, arborele binar complet din figură este un *max-heap*.



și poate fi reprezentat sevențial astfel:

1	2	3	4	5
9	7	5	5	7

Crearea unui heap

I. O primă soluție de creare a unui *heap* este de a insera succesiv elementele în *heap*, plecând inițial de la *heap*-ul format dintr-un singur element.

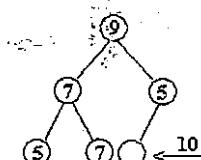
```

void CreareHeap()
//creeaza un heap inserand succesiv elemente
{for (int i=2; i<=n; i++) InsertHeap(i-1, H[i]);}
  
```

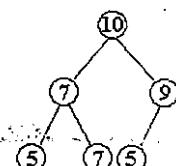
Inserarea unui nod într-un heap

Fie $H[n]$ un *heap* cu n elemente și x valoarea ce trebuie inserată. Inițial, valoarea x va fi memorată în *heap* pe pozitia $n+1$. Apoi se restaurează eventual *heap*-ul, „promovând” valoarea x spre rădăcină, până când proprietatea de *heap* este restabilită.

De exemplu, să inserăm valoarea 10 în *heap*-ul din figură:



Valoarea inserată va ocupa poziția 6. Pentru a conserva proprietatea de *heap*, valoarea 10 trebuie promovată până în rădăcină, valorile de pe drumul parcurs spre rădăcină fiind retrogradate succesiv, cu un nivel. Rezultă arborele:



Modificările sunt aplicate de fapt vectorului ce constituie reprezentarea implicită:

1	2	3	4	5	6
10	7	9	5	7	5

```

void InsertHeap (int x)
//insereaza valoarea x in heap-ul H cu n elemente
int fiu, tata;
fiu=++n; tata=n/2;
//fiu indica pozitia pe care trebuie plasata valoarea x
while (tata && H[tata]<x)
    //valoarea x trebuie promovata
    H[fiu]=H[tata];
    fiu=tata; tata=fiu/2; }
H[fiu]=x;
  
```

Analizând *complexitatea* procedurii de inserare deducem că, în cazul cel mai defavorabil, valoarea nou inserată urcă până în rădăcină arborelui, deci necesită $O(n)$ operații elementare, unde n este înălțimea *heap*-ului. Un *heap* fiind un arbore binar complet, înălțimea sa $h=[\log_2 n]$, deci inserarea unui nod într-un *heap* cu n elemente este de $O(\log n)$.

Să estimăm complexitatea în cazul cel mai defavorabil a procedurii de construcție a *heap*-ului prin inserări succesive. Dacă toate cele 2^i valori de pe nivelul i urcă până în rădăcină, obținem:

$$\sum_{i=1}^{[\log n]} i \cdot 2^i < \log n \sum_{i=1}^{[\log n]} 2^i = O(n \log n)$$

II. O strategie mai eficientă de construcție a unui *heap* se bazează pe ideea de echilibrare. Apelul procedurii *InsertHeap(x)* poate fi interpretat ca o combinare a două *heap*-uri: un *heap* cu n elemente și un *heap* format numai din elementul x . Putem construi *heap*-ul cu rădăcina $H[i]$ combinând la fiecare pas îi două *heap*uri de dimensiuni apropiate, *heap*-ul cu rădăcina $2i$ cu *heap*-ul cu rădăcina $2i+1$ și cu elementul $H[i]$.

```

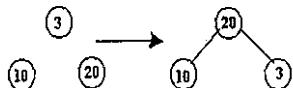
void CreareHeap()
{for (int i=n/2; i>0; i--) CombHeap(i, n);}
  
```

```

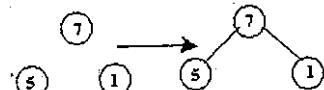
void CombHeap (int i, int n)
//combină elementul H[i] cu heap-ul cu radacina 2i
//si heap-ul cu radacina 2i+1
int fiu, tata, v;
v=H[i]; tata=i;
//tata indica pozitia pe care va fi plasata valoarea v
fiu=2*i; //fiul stang
while (fiu<=n)
    {if (fiu<n) //există fiu drept
        if (H[fiu]<H[fiu+1]) fiu++;
        //fiu indica fiul cu valoarea cea mai mare
        if (v<H[fiu])
            H[tata]=H[fiu];
    }
  
```

```
//valoarea celui mai mare dintre fiți urca în arbore
tata=fiu;
fiu=fiu*2;
else
break; //am determinat pozitia corecta pentru v
H[tata]=v;
```

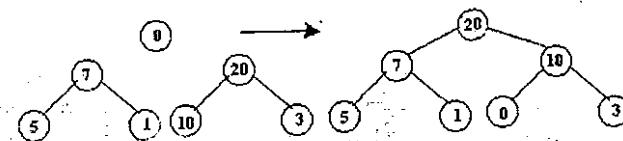
De exemplu, organizarea ca heap a vectorului $H = \{0, 7, 3, 5, 1, 10, 20\}$ se face în trei pași. La pasul $i=3$ se apelează $\text{CombHeap}(3, 7)$:



După acest pas $H = \{0, 7, 20, 5, 1, 10, 3\}$. La pasul $i=2$ se apelează $\text{CombHeap}(2, 7)$:



La pasul $i=1$, se apelează $\text{CombHeap}(1, 7)$:



Heap-ul obținut în reprezentare implicită este $H = \{20, 7, 10, 5, 1, 0, 3\}$.

Complexitatea algoritmului

Notăm $h = \lceil \log_2 n \rceil$ înălțimea heap-ului. Pentru fiecare nod x de pe nivelul i , $i \in \{0, 1, \dots, h-1\}$, pentru a construi heap-ul cu rădăcina x se fac cel mult $h-i$ coborări ale valorii x în arbore. În cazul cel mai defavorabil, numărul de operații elementare efectuate de algoritm pentru a construi un heap cu n elemente este:

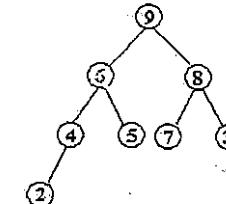
$$T_n \leq \sum_{i=0}^{h-1} 2^i (h-i) = \sum_{j=1}^h j \cdot 2^{h-j} = \sum_{j=1}^h 2^h \cdot \frac{j}{2^j} \leq n \sum_{j=1}^h \frac{j}{2^j} < 2n = O(n)$$

Algoritmul de construcție a unui heap a devenit astfel liniar.

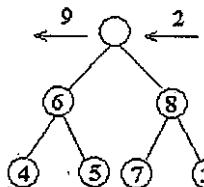
Am prezentat și analizat deja procedura InsertHeap , care realizează operația de inserare. Extragerea elementului de prioritate maximă presupune stergerea din heap a valorii din rădăcină. Pentru aceasta se plasează în rădăcină (prima poziție din vector) ultimul element din heap. Evident, numărul de elemente din heap scade, iar heap-ul trebuie

restaurat. Cum subarborele rădăcinii și-au păstrat structura de heap, restaurarea se face combinând valoarea din rădăcină cu heap-urile fiilor.

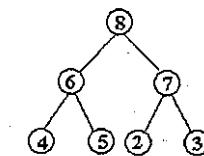
De exemplu, din heap-ul din figură se extrage nodul cu valoarea 9,



locul lui fiind ocupat de nodul cu valoarea 2.



Cum 8 este maximul dintre succesorii săi, el îl înlocuiește pe 2. Apoi 2 este înlocuit de 7. Rezultă heap-ul următor:



În reprezentare implicită, heap-ul

1	2	3	4	5	6	7	8
9	6	8	4	5	7	3	2

devine

1	2	3	4	5	6	7
8	6	7	4	5	2	3

```
int ExtractMax()
{//extrage elementul maxim din heap-ul H de dimensiune n
int Max=H[1]; //extrage valoarea din radacina
H[1]=H[n--];
CombHeap(1, n); //restaureaza heap-ul
return Max;}
```

Complexitatea operației de extragere a elementului de cheie maximă se reduce la complexitatea algoritmului de combinare a două heap-uri, care este $O(\log n)$.

6. Heapsort

O aplicație importantă a *heap*-urilor este sortarea unui vector, metodă cunoscută sub denumirea de *heapsort*.

Primul pas este de a organiza vectorul ca un *heap*. Deoarece, conform proprietății de *heap*, elementul maxim din vector este plasat în rădăcina *heap*-ului, deci pe poziția 1, el poate fi plasat pe poziția sa corectă, interschimbându-l cu elementul de pe poziția n . Noul element din rădăcina *heap*-ului nu respectă proprietatea de *heap*, dar subarborei rădăcinii rămân *heap*-uri. Prin urmare trebuie restaurat *heap*-ul, apelând *CombHeap*(1, $n-1$), elementul de pe poziția n fiind deja pe poziția sa corectă nu mai este inclus în *heap*. Procedeul se repetă până când toate elementele vectorului sunt plasate pe pozițiile lor corecte.

```
int aux;
for (int i=n; i>=2; i--)
    {aux=H[1]; H[1]=H[i]; H[i]=aux;
     CombHeap (1, i-1);}
```

Complexitatea algoritmului *heapsort* este de $O(n \log n)$, crearea *heap*-ului fiind liniară, iar fiecare dintre cele $n-1$ apeluri ale funcției *CombHeap*() fiind de complexitate $O(\log n)$. Deci algoritmul de sortare cu ajutorul *heap*-urilor este optimal în cazul cel mai favorabil.

7. Beculete

Fie o rețea ortogonală, ale cărei linii sunt la distanță unitară, iar nodurile în puncte de coordonate întregi; pătratele rețelei sunt colorate inițial în alb (culoarea de indice 0). În n noduri ale rețelei ($1 \leq n \leq 255$) se află câte un becul. Unind în linie dreaptă printr-un fir două beculete, toate pătratele traversate efectiv de fir (firul trece prin interiorul păratului) și schimbă culoarea din 0 în 1. Instalația funcționează dacă și numai dacă între oricare două becuri, există (direct sau indirect) legătură.

Determinați o modalitate de conectare a celor n becuri, astfel încât instalația să funcționeze, iar colorațura rețelei să fie de indice minim. Prin indicele unei coloraturi înțelegem suma indicilor de culoare ai tuturor păratelor din rețea.

Date de intrare

Fișierul de intrare INPUT.TXT conține pe prima linie n , numărul de beculete, iar pe următoarele n linii câte două numere întregi din intervalul $[-16000, 16000]$, care reprezintă coordonatele nodurilor rețelei în care sunt plasate cele n beculete.

Date de ieșire

Fișierul de ieșire OUTPUT.TXT conține pe prima linie indicele coloraturii obținute; pe două linii numărul k , de legături directe realizate; iar pe fiecare dintre următoarele k linii, coordonatele a două beculete unite prin fir direct, separate prin câte un spațiu.

(ONI, Timișoara, 1997, clasa a XII-a)

Exemplu

INPUT.TXT
3
0 0
20 5
18 40

OUTPUT.TXT
56
2
0 0 20 5
20 5 18 40

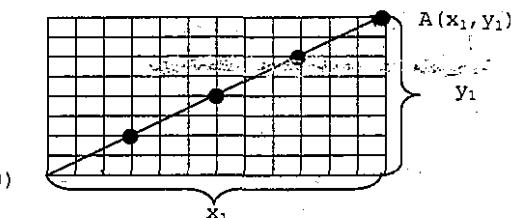
Soluție

Potem asocia problemei un graf ale căruia vârfuri să fie becuile; între oricare două vârfuri există muchie având drept cost numărul de pătrate traversate de segmentul care unește pozițiile becușelor corespunzătoare extremităților muchiei. Pentru ca rețea să funcționeze trebuie ca între oricare două becuri să existe (direct sau indirect) legătură. Prin urmare, putem formula problema în limbaj de grafuri astfel: determinați un graf parțial conex având cost minim.

Mai exact, deoarece arborele este un graf conex minimal, problema cere determinarea unui arbore parțial de cost minim.

O primă subproblemă a problemei date constă în evaluarea costului unei muchii. Să considerăm $A(x_1, y_1)$ un punct de coordonate întregi și $O(0, 0)$ originea sistemului de coordonate. Să numărăm câte pătratele unitare sunt traversate de segmentul (OA) . Pentru a ajunge în A , segmentul (OA) traversează x_1 pătratele pe orizontală și y_1 pătratele pe verticală. Unele dintre aceste pătrate sunt numărate de două ori (acest lucru se întâmplă ori de câte ori segmentul (OA) trece printr-un punct al rețelei, adică un punct de coordonate întregi).

Dacă $\text{cmmdc}(x_1, y_1)=1$, singurul punct de coordonate întregi traversat de segmentul (OA) este punctul A . Dacă $\text{cmmdc}(x_1, y_1)=d > 1$, numărul de puncte de coordonate întregi traversate de segmentul (OA) este d , deoarece segmentul (OA) traversează și $d-1$ puncte interioare:



Dacă dorim să estimăm costul muchiei dintre punctele $A(x_1, y_1)$ și $B(x_2, y_2)$, este suficient să facem o translație a sistemului de coordonate și obținem formula generală: $\text{cost}(AB) = \text{abs}(x_1 - x_2) + \text{abs}(y_1 - y_2) - \text{cmmdc}(\text{abs}(x_1 - x_2), \text{abs}(y_1 - y_2))$

Deoarece calculul costului unei muchii presupune determinarea cmmdc , operație costisitoare, vom determina costurile muchiilor o singură dată și le vom reține într-o matrice de costuri C . Numărul maxim de beculete fiind 255 nu putem aloca această matrice static, o vom aloca dinamic, linie cu linie.

Pentru a determina un arbore parțial de cost minim vom aplica algoritmul lui Prim.

Algoritmul lui Prim utilizează o strategie Greedy. Inițial se pleacă de la un arbore format dintr-un singur vîrf. La fiecare pas se selectează o muchie de cost minim astfel încât multimea muchiilor selectate și multimea vîrfurilor unite de acestea să formeze un arbore (această muchie va avea ca extremități un vîrf deja selectat în arbore și un vîrf neselectat).

Mulțimea vârfurilor deja selectate în arbore o vom nota X și o vom reprezenta prin vector caracteristic.

Pentru a selecta la fiecare pas muchia de cost minim, vom utiliza un vector key cu n componente, în care pentru fiecare vârf neselectat x (deci care nu aparține mulțimii X) vom reține costul minim al muchiilor ce unesc vârful x cu un vârf v deja selectat în arbore: $key[x] = \min\{C([x, v]) \mid v \in X\}$, $\forall x \notin X$. Evident, dacă astfel de muchii nu există $key[x] = +\infty$.

Vom reprezenta arborele parțial de cost minim printr-un vector APM în care reținem vârfurile grafului, în ordinea în care au fost atinse:

$APM[x] =$ vârful din care a fost atins x (cealaltă extremitate a muchiei utilizată pentru selectarea vârfului x).

```
#define NMaxVf 256
#define Infinit 32500
int *C[NMaxVf]; //matricea costurilor
int n; //numarul de becuri
int X[NMaxVf], key[NMaxVf], APM[NMaxVf];
struct Bec {int x, y;} B[NMaxVf]; //pozițiile [becurilor]
```

Vom citi datele de intrare și vom inițializa matricea costurilor:

```
void Citire()
{
    ifstream fin("input.txt");
    int i, j;
    fin >> n;
    for (i=1; i<=n; i++)
        fin >> B[i].x >> B[i].y;
    fin.close();
    //aloc memorie pentru matricea costurilor
    for (i=0; i<=n; i++) C[i]=new int[n];
    //calculez costurile
    for (i=1; i<=n; i++)
        {C[i][i]=0;
        for (j=i+1; j<=n; j++)
            {C[i][j]=abs(B[i].x-B[j].x)+abs(B[i].y-B[j].y);
            cmmdc(abs(B[i].x-B[j].x), abs(B[i].y-B[j].y));
            C[j][i]=C[i][j];}}
```

Apoi vom aplica algoritmul lui Prim:

```
void Prim()
{
    int i, min, vfmin;
    X[1]=1; APM[1]=-1; key[1]=0; //selectam initial varful 1
    int nrw=1; //nr de varfuri selectate
    for (i=2; i<=n; i++) {key[i]=C[1][i]; APM[i]=1;}
    while (nrw<n)
        {min=Infinit;
        for (i=2; i<=n; i++)
```

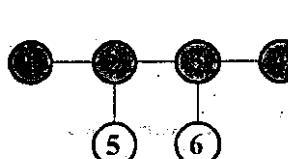
```
        if (!X[i]) //i este varf neselectat
            if (key[i]<min)
                {min=key[i]; vfmin=i;}
        X[vfmin]=1; nrw++;
        for (i=2; i<=n; i++) //actualizez cheile vârfurilor
            if (!X[i] && key[i]>C[i][vfmin])
                {key[i]=C[i][vfmin];
                APM[i]=vfmin;}}
```

Complexitatea algoritmului

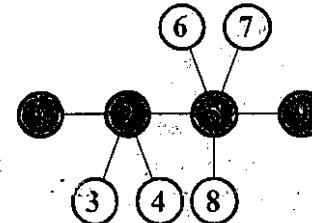
Algoritmul execută $n-1$ pași, la fiecare pas selectând un vârf din graf de cheie minimă și reactualizând cheile vârfurilor neselectate, operație de complexitate $O(n)$. Deci algoritmul este de ordinul $O(n^2)$.

8. Omida

O „omidă” este un arbore cu următoarea proprietate: există un lanț central astfel încât fiecare nod al arborelui fie aparține *lanțului* central fie este adiacent lanțului. Figurile de mai jos ilustrează două „omizi”, nodurile gri indicând lanțul.



„Omida” 1



„Omida” 2

Figura 1

Figura 2

Pot exista mai multe lanțuri. De exemplu, un alt lanț pentru „omida” 2 este 3-2-5-9.

Scrieți un program care, fiind dată o „omidă” cu N noduri, va eticheta nodurile cu numere de la 1 la N astfel încât:

- fiecare etichetă de la 1 la N este folosită o singură dată;
- nu există două muchii cu același modul al diferenței dintre etichetele nodurilor adiacente.

Mai jos este ilustrat un mod de etichetare a „omizii” 2. Modulele diferențelor sunt indicate pe muchiile respective.

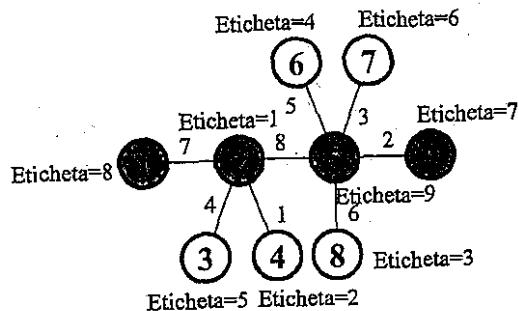


Figura 3

Date de intrare

Fișierul de intrare CP.IN conține

- N – un întreg ce indică numărul de noduri
 - $x_1 \ y_1$ – extremitățile primei muchii
 - $x_2 \ y_2$ – extremitățile celei de a doua muchii
 - ...
 - $x_{N-1} \ y_{N-1}$ – extremitățile muchiei N-1
 - $x_N \ y_N$ – extremitățile muchiei N-1.
- Datele de intrare sunt corecte și arborele dat este o „omidă”.

Date de ieșire

Fișierul de ieșire CP.OUT conține

$L_1 \ L_2 \ \dots \ L_N$
 (N numere întregi, separate prin spații, unde L_i reprezintă eticheta asociată nodului i).
 În cazul când există mai multe soluții, se va prezenta numai una. Dacă nu este posibilă
 nici o etichetare, fișierul de ieșire va conține cuvântul IMPOSSIBIL.

Restricții

- $2 \leq N \leq 10000$

Exemplu

Fișierul de intrare (CP.IN) descrie „omida” din figura 2, iar fișierul de ieșire (CP.OUT) descrie etichetarea din figura 3.

CP.IN	CP.OUT (o soluție posibilă)
9	8 1 5 2 9 4 6 3 7
1 2	
6 5	
5 7	
4 2	
2 3	
8 5	
2 5	
5 9	

(CEOI, România, 2000)

Soluție

Vom reprezenta arborele dat prin liste de adiacență:

```

#define MAX_N 10001
struct NodLista
{
    int vf;
    NodLista *urm;
};
typedef NodLista *Lista;
int n; // numarul de noduri
Lista A[MAX_N]; // listele de adiacenta
  
```

La citirea informațiilor vom crea listele de adiacență și vom calcula gradul fiecărui nod din arbore:

```

int far d[MAX_N]; // gradele nodurilor
void AdMuchie(int x, int y)
/* Adauga muchia [x,y] */
{ Lista l= new NodLista;
  l->vf=y; l->urm=A[x]; A[x]=l;
  d[x]++;
}
void Citire(void)
{ ifstream f("cp.in");
  int i, x, y;
  f>>n;
  for (i=1; i<n; i++)
  { f>>x>>y;
    AdMuchie(x, y); AdMuchie(y, x); }
  f.close(); }
  
```

Nodurile care aparțin lanțului central au gradul > 1 . O extremitate a lanțului central este un nod care are gradul > 1 și este adjacent cu cel mult un nod terminal (nod de grad 1). Vom începe etichetarea nodurilor arborelui cu o extremitate a lanțului central, pe care o vom denumi rădăcină.

```

int DeterminaRad(void)
{ int i, nr;
  Lista l;
  if (n<3) return 1;
  for (i=1; i<=n; i++)
  { if (d[i]>1)
      { nr=0; //numarul de vecini de pe lant ai nodului i
        for (l=A[i]; l; l=l->urm)
          //parcurg lista de adiacenta a nodului i
          if (d[l->vf]>1) nr++;
        if (nr<=1) return i; }
    }
  return 0; }
  
```

Vom eticheta nodurile arborelui astfel încât să obținem în ordine următoarele diferențe dintre nodurile etichetate $n-1, n-2, \dots, 1$.

În acest scop vom utiliza două variabile: *st*, care indică valoarea cu care vom eticheta nodul curent de pe lanțul central, și *dr*, care indică valoarea cu care vom eticheta următorul nod terminal, adiacent cu nodul curent de pe lanțul central.

Inițial variabila *st* primește valoarea 1 (valoare cu care vom eticheta rădăcina), iar *dr* primește valoarea *n*, pentru a obține diferența dintre etichetele *n*-1. Toate nodurile terminale adiacente nodului curent de pe lanțul central vor fi etichetate succesiv cu valoarea variabilei *dr*. Evident, după ce utilizăm o valoare pentru etichetare o actualizăm (valoarea mică va fi incrementată cu 1, iar valoarea mare va fi decrementată cu 1).

După ce etichetez toate nodurile terminale adiacente cu nodul curent de pe lanțul central, vom trece la următorul nod neetichetat de pe lanțul central (acesta este unic, deoarece am început etichetarea dintr-o extremitate a lanțului). Diferența următoare se va obține între etichetele a două noduri adiacente de pe lanțul central. Pentru ca această diferență să aibă valoarea corespunzătoare, la trecerea la un nou nod de pe lanțul central vom inversa valorile variabilelor *st* și *dr*, continuând apoi etichetarea în același mod:

```
void Etichetare(void)
{
    int NodC=Rad, NodUrm;
    int st=1, PasSt=1, dr=n, PasDr=-1, aux;
    Lista l;
    while (NodC)
    /*cat timp mai există noduri pe lantul central
    { NodUrm=0; l=A[NodC];
        Et[NodC]=st; st+=PasSt;
        while (1)
            /*parcureg lista de adiacenta a nodului curent si
            etichetez toate nodurile terminale adiacente cu i */
            if (d[l->vf]==1) //nod terminal
                {Et[l->vf]=dr;
                 dr+=PasDr; }
            else //nod neterminal
                if (!Et[l->vf]) //neetichetat
                    NodUrm=l->vf;//urmatorul nod de pe lant
                    l=l->urm;
                }
        /*inversez valorile de etichetare, pentru a continua
        cu urmatorul nod de pe lantul central*/
        aux=st; st=dr; dr=aux;
        aux=PasSt; PasSt=PasDr; PasDr=aux;
        NodC=NodUrm; }
}
```

9. Centru

O mare companie de calculatoare are *N* ($1 \leq N \leq 16000$) depozite cu componente, legate între ele prin străzi bidirectionale de lungime 1 km. Între oricare două centre de desfacere există exact un singur drum care le unește (un drum este format dintr-o succesiune de străzi, oricare două străzi alăturate având un capăt comun). Directorul companiei dorește să stabilească depozitul central într-unul dintre cele *N* depozite, care are proprietatea că distanța de la el la cel mai îndepărtat depozit este minimă (distanța este calculată ca fiind

lungimea drumului ce unește cele două depozite). Fiind un angajat al acestei mari companii, sarcina dumneavoastră este de a determina în câte dintre cele *N* depozite s-ar putea situa depozitul central și care sunt acestea.

Date de intrare

Pe prima linie a fișierului CENTRU.IN se află numărul întreg *N*, reprezentând numărul de depozite. Pe următoarele *N*-1 linii se află câte două numere întregi *i* și *j* (*i* ≠ *j*), având semnificația că există o stradă între depozitele *i* și *j*.

Date de ieșire

Pe prima linie a fișierului CENTRU.OUT veți afișa numărul depozitelor care ar putea deveni depozit central. Pe a doua linie se vor afișa numerele de ordine ale acestor depozite (în ordine crescătoare), separate prin spații.

Exemplu

CENTRU.IN
6
1 6
2 3
3 5
3 6
4 6

CENTRU.OUT
2
3 6

Soluție

Vom reprezenta arborele prin liste de adiacență, ca în problema precedentă. De asemenea, la citire vom calcula și gradele nodurilor arborelui.

Vom realiza o parcursare a arborelui dinspre „exterior spre interior”. La fiecare pas vom reține și distanța dintre nodul curent și exterior.

Pentru aceasta vom utiliza o coadă. Inițial în coadă vom insera toate nodurile terminale:

```
Lista aux, Inc=NULL, Sf; //indica inceputul si sfarsitul cozii
for (int i=1; i<=n; i++)
    if (d[i]==1)
        (dist[i]=1;
         if (!Inc)
             Inc = new NodLista;
             Inc->vf=i; Inc->urm=NULL;
             Sf=Inc; }
         else
             aux = new NodLista;
             aux->vf=i; aux->urm=NULL;
             Sf->urm=aux; Sf=aux; }
```

Cât timp nu am parcurs tot arborele (deci coada nu este vidă), vom extrage din coadă un nod terminal și-l vom elimina din arbore, împreună cu muchia incidentă cu acesta. Prin eliminarea acestui vârf este posibil ca vârful adiacent cu el să devină terminal, caz în care vârful respectiv trebuie inserat în coadă.

```

while (Inc)                                //coada nu este vida
    {i=Inc->vf;
     aux=A[i];   //parcureg lista de adiacenta a nodului i
     while (aux)
        {int j=aux->vf;
         d[j]--; d[i]--;           //decrementam gradele
         if (d[j]==1)             //j a devenit terminal
            {dist[j]=dist[i]+1;   //retin distanta
             aux = new NodLista; //inserez in coada
             aux->vf=j; aux->urm=NULL;
             Sf->urm=aux; Sf=aux; }
         aux=aux->urm;          }
     //extrag elementul din coada
     aux=Inc; Inc=Inc->urm; delete aux;      }
}

```

Pentru a determina nodurile care pot constitui centrul grafului vom parcurge vectorul dist, reținând maximul:

```

void Afisare()
{
ofstream f("centru.out");
int max=0, NrD=0;
for (int i=1; i<=n; i++)
    if (dist[i]>max)
        {max=dist[i]; NrD=1; }
    else
        if (dist[i]==max) NrD++;
f<<NrD<<endl;
for (i=1; i<=n; i++)
    if (dist[i]==max) f<<i<<' ';
f<<endl;
f.close();
}

```

Probleme propuse

1. Generări arbori

- Generează toți arborii cu n vârfuri.
- Calculăți numărul arborilor cu n vârfuri și secvența gradelor vârfurilor d_1, d_2, \dots, d_n ($d_i \geq 1, \forall i \in \{1, 2, \dots, n\}, d_1+d_2+\dots+d_n=2n-2$).
- scrieți un program de generare a tuturor arborilor cu secvența gradelor dată.
- Calculați numărul arborilor cu n vârfuri, dintre care p vârfuri terminale.

2. Înălțime

- Scrieți o funcție care să determine înălțimea unui arbore reprezentat prin referințe descendente.
- Scrieți o funcție care să determine înălțimea unui arbore reprezentat prin referințe ascențente.

3. Noduri terminale

Scrieți o funcție care să determine numărul de noduri terminale ale unui arbore.

4. Arbore binar strict

Scrieți o funcție care să verifice dacă un arbore binar este strict (fiecare vârf are 2 descendenti sau nici unul).

5. Arbore echilibrat

Un arbore binar este denumit echilibrat dacă pentru orice nod al său x , diferența dintre numărul de noduri din subarborele stâng al lui x și numărul de noduri din subarborele drept al lui x este cel mult 1. Scrieți o funcție care să construiască un arbore binar echilibrat cu n noduri (valoarea lui n fiind transmisă ca parametru).

6. Egalitate

Scrieți o funcție care să testeze dacă doi arbori binari, reprezentați prin referințe descendente, specificați ca parametru, au aceeași structură.

7. Șeful econom

Şeful unei mari firme dorește să proiecteze o rețea de calculatoare astfel încât să poată comunica cu toate filialele pe care le are în țară. Serverul pe care îl are instalat șeful firmei în biroul personal trebuie să poată comunica, direct sau indirect, cu toate serverele din birourile șefilor de filială printr-o linie telefonică dedicată. RomTelecom este de acord să asigure toate legăturile telefonice necesare, stabilind pentru fiecare linie de legătură directă o taxă lunară direct proporțională cu lungimea liniei telefonice dedicate, exprimată în km, factorul de proporționalitate fiind specificat k ($k \in \mathbb{R}$). Șeful firmei ia harta țării, desenează un sistem de coordonate cartezian cu centru în Iași, unde se găsește biroul său personal, calculează coordonatele birourilor șefilor de filială și organizează un concurs de programe care, pe baza coordonatelor calculate (corect!!!) de șef să determine un mod căt mai economicos de interconectare a calculatoarelor din rețea. Participați și voi la concurs!

Date de intrare

Datele de intrare se citesc din fișierul RETEA.IN cu structura:

RETEA.IN
k
n
oras ₁ x ₁ y ₁
oras ₂ x ₂ y ₂
...
oras _n x _n y _n

Semnificație
factorul de proporționalitate
numărul de filiale ($n \leq 250$)
oras _i – numele orașului în care se află filiala i
x _i , y _i – coordonatele filialei i (numere reale)

Numele de orașe nu conțin spații și au maxim 30 de caractere. Numele de orașe și coordonatele orașelor sunt separate printr-un singur spațiu.

Date de ieșire

În fișierul de ieșire RETEA.OUT se va afișa pe prima linie c, cu două zecimale cu rotunjire, c fiind taxa lunară totală plătită de firmă la RomTelecom, iar pe fiecare din următoarele linii căte două nume de orașe, separate printr-un singur spațiu, în care se găsesc filiale conectate direct în rețea.

Exemplu

RETEA.IN	RETEA.OUT
0.5	125.13
3	Pascani Iași
Pascani 0 80	Targu-Neamt Pascani
Targu-Neamt 10 120.5	Suceava Iasi
Suceava 125 30	

(OJI, Iași, 1999)

8. Urgență

Autoritățile dintr-o zonă de munte intenționează să stabilească un plan de urgență, pentru a reacționa mai eficient la frecvențele calamități naturale din zonă. În acest scop au identificat N puncte de interes strategic și le-au numerotat distinct de la 1 la N. Punctele de interes strategic sunt conectate prin M căi de acces având priorități în funcție de importanță. Între oricare două puncte de interes strategic există cel mult o cale de acces ce poate fi parcursă în ambele sensuri și cel puțin un drum (format din una sau mai multe căi de acces) ce le conectează.

În cazul unei calamități, unele căi de acces pot fi temporar întrerupte și astfel între anumite puncte de interes nu mai există legătură. Ca urmare pot rezulta mai multe grupuri de puncte în aşa fel încât între oricare două puncte din același grup să existe măcar un drum și între oricare două puncte din grupuri diferite să nu existe drum.

Autoritățile estimează gravitatea unei calamități ca fiind suma priorităților căilor de acces distruse de această și doresc să determine un scenariu de gravitate maximă, în care punctele de interes strategic să fie împărțite într-un număr de K grupuri.

Date de intrare

Fișierul de intrare URGENTA.IN are următorul format:

URGENTA.IN	<i>Semnificație</i>
N M K	
i ₁ j ₁ p ₁	– între punctele i ₁ și j ₁ există o cale de acces de prioritate p ₁
i ₂ j ₂ p ₂	– între punctele i ₂ și j ₂ există o cale de acces de prioritate p ₂
...	...
i _M j _M p _M	– între punctele i _M și j _M există o cale de acces de prioritate p _M

Date de ieșire

Fișierul de ieșire URGENTA.OUT va avea următorul format:

URGENTA.OUT	<i>Semnificație</i>
gravmax	– gravitatea maximă
C	– numărul de căi de acces întrerupte de calamitate

k₁ h₁
k₂ h₂
...
k_c h_c

- între punctele k₁ și h₁ a fost întreruptă calea de acces
- între punctele k₂ și h₂ a fost întreruptă calea de acces
- ...
- între punctele k_c și h_c a fost întreruptă calea de acces

Restricții și precizări

- 0 < N < 256
- N - 2 < M < 32385
- 0 < K < N + 1
- Prioritățile căilor de acces sunt întregi strict pozitivi mai mici decât 256.
- Un grup de puncte poate conține între 1 și N puncte inclusiv.
- Dacă există mai multe soluții, programul va determina una singură.
- Timp maxim de execuție: 1 secundă / test

Exemplu

URGENTA.IN	URGENTA.OUT
7 11 4	27
1 2 1	8
1 3 2	1 3
1 7 3	1 7
2 4 3	2 4
3 4 2	3 4
3 5 1	3 7
3 6 1	4 5
3 7 5	5 6
4 5 5	6 7
5 6 4	
6 7 3	

(OJI, 2002, clasele a XI-a – a XII-a)

9. Banana

Se consideră o pădure tropicală, reprezentată sub forma unui caroaj dreptunghiular. Celula din colțul stânga sus al caroiajului are coordonatele (1, 1), iar coordonatele celorlalte celule sunt determinate de linia și coloana pe care se află. În anumite celule ale caroiajului sunt păsări bananieri; o celulă conține cel mult un bananier. Mai mulți bananieri care se învecinează pe orizontală sau verticală formează o zonă de bananieri. Într-o astfel de zonă, CEKILI se deplasează ușor, cu agilitatea-i cunoscută, de la un bananier la altul.

Maimuța CEKILI este lacomă și nu îi ajung bananele dintr-o singură zonă. Tarzan vrea să-și ajute prietena. Pentru aceasta, el ar putea conecta exact K zone de bananieri înnodând mai multe liane și astfel CEKILI s-ar putea deplasa de la o zonă la alta utilizând lianele. Evident, Tarzan trebuie să aleagă zonele astfel încât numărul total de bananieri din cele K zone să fie maxim.

Determinați numărul maxim de bananieri care se poate obține prin conectarea a exact K zone.

Date de intrare

Fișierul de intrare BANANA.IN conține:

BANANA.IN
 Nr K
 $x_1 \ y_1$
 $x_2 \ y_2$
 \dots
 $x_{Nr} \ y_{Nr}$

Semnificație
 Nr – numărul de bananieri
 K – numărul de zone ce pot fi conectate
 x_i – linia pe care se află bananierul i
 y_i – coloana pe care se află bananierul i

Date de ieșire

Fișierul de ieșire BANANA.OUT va conține pe prima linie numărul maxim de bananieri care se poate obține prin conectarea zonelor.

Restricții și precizări

- $1 \leq \text{Nr} \leq 16000$
- $1 \leq x_i, y_i \leq 10000, \forall i \in \{1, 2, \dots, \text{Nr}\}$
- În teste utilizate K nu va depăși numărul de zone.
- Două poziții se învecinează pe orizontală dacă sunt pe aceeași linie și pe coloane consecutive, respectiv pe verticală dacă sunt pe aceeași coloană și pe linii consecutive.

Timp maxim de execuție: 1 secundă/test.

Exemplu

BANANA.IN
 10 3
 7 10
 1 1
 101 1
 2 2
 102 1
 7 11
 200 202
 2 1
 3 2
 103 1

BANANA.OUT
 9

(Baraj, Brăila, 2002)

10. Galerii Subterane

Într-un munte există n peșteri subterane numerotate de la 1 la n, conectate prin m tuneluri. Se știe că orice pereche de peșteri este conectată prin cel mult un tunel și orice tunel conectează două peșteri distincte. Tunelurile nu se intersecează în afara peșterilor, iar orice tunel poate fi parcurs de un eventual explorator în ambele direcții. Vom numi galerie o mulțime nevidă de peșteri cu următoarele proprietăți:

1. Un explorator ce se află într-o peșteră arbitrară din galerie poate ajunge în orice altă peșteră a galeriei, parcurgând un drum ce constă din unul sau mai multe tuneluri.
2. Pentru orice două peșteri distincte din galerie acest drum este unic.
3. Nu există peșteri din afara galeriei în care se poate ajunge din peșteri ce fac parte din galerie.

Vom spune că două galerii distincte A și B sunt asemănătoare dacă și numai dacă fiecare peșteră a din galeria A îl corespunde exact o peșteră I(a) din galeria B astfel încât:

1. $I(a_1) = I(a_2)$ numai dacă $a_1 = a_2$.
2. Pentru orice peșteră b din B există o peșteră a din A, astfel încât $I(a) = b$.
3. Peșterile $I(a_1)$ și $I(a_2)$ din B sunt conectate printr-un tunel dacă și numai dacă peșterile a_1 și a_2 din A sunt conectate printr-un tunel.

Fie un munte care conține exact două galerii. Scrieți un program ce determină dacă aceste galerii sunt asemănătoare.

Date de intrare

Fișierul galerii.in conține pe prima linie numerele m și n. Pe următoarele m linii sunt scrise câte două numere i și j având semnificația că peșterile i și j sunt conectate printr-un tunel.

Date de ieșire

Fișierul galerii.out va conține pe prima linie numărul 1 dacă peșterile sunt asemănătoare și 0, dacă nu sunt. În cazul în care peșterile sunt asemănătoare, pe următoarele n linii se vor scrie câte două numere i și j având semnificația că peșterii i din galeria A îl corespunde peștera j din galeria B.

(Olimpiada Republicană de Informatică, Republica Moldova, 2001)

11. Arbore

Se dă un arbore cu N noduri. Se cere ca, prin eliminarea unui număr minim de muchii, să se izoleze un subarbore cu P noduri ($1 \leq P \leq N \leq 150$).

Date de intrare/ieșire

Fișierul de intrare ARBORE.IN conține pe prima linie valorile N și P, despărțite printr-un blanc. Următoarele $N-1$ linii contin fiecare câte două numere I și J, cuprinse între 1 și N separate printr-un blanc, cu semnificația că nodul J este descendenter direct al nodului I.

Fișierul de ieșire ARBORE.OUT conține numărul de muchii eliminate.

Timp maxim de execuție: 1 secundă/test.

Exemplu

ARBORE.IN	ARBORE.OUT
11 6	2
1 2	
2 6	
2 7	
2 8	
1 3	
1 4	
4 9	
4 10	
4 11	
1 5	

(Baraj, București, 2001)

12. Transformare arbore

Să considerăm un arbore cu N vârfuri, numerotate de la 1 la N . Scrieți un program care să adauge, dacă este posibil, un număr minim de muchii astfel încât fiecare vârf să aparțină exact unui singur ciclu.

Date de intrare

Fișierul de intrare ARBORE.IN conține:

ARBORE.IN	<i>Semnificație</i>
N	numărul de vârfuri din arbore
$x_1 \ y_1$	
$x_2 \ y_2$	x_i și y_i sunt extremitățile muchiei i
\dots	
$x_{N-1} \ y_{N-1}$	

Date de ieșire

Fișierul de ieșire ARBORE.OUT va conține pe prima linie valoarea -1 dacă problema nu admite soluție, respectiv numărul de muchii adăugate, dacă problema admite soluție. Dacă problema admite soluție, pe fiecare dintre următoarele linii se vor scrie extremitățile unei muchii adăugate, separate printr-un spațiu, sub forma:

ARBORE.OUT	<i>Semnificație</i>
Nr	numărul de muchii adăugate
$a_1 \ b_1$	
$a_2 \ b_2$	a_i și b_i sunt extremitățile unei muchii adăugate
\dots	
$a_{Nr} \ b_{Nr}$	

Restrictii

- $3 \leq N \leq 100$
- x_i, y_i sunt numere întregi din intervalul $[1; N]$.

Timp maxim de execuție: 1 secundă/test.

Exemple

ARBORE.IN	ARBORE.OUT	ARBORE.IN	ARBORE.OUT
4	-1	7	2
1 2		1 2	
2 3		1 3	
2 4		3 5	
		3 4	
		5 6	
		5 7	

(ONI, Brăila, 2002, clasele a XI-a și a XII-a)

Bibliografie

1. Atanasiu, Adrian; Pintea, Rodica, *Culegere de probleme Pascal*, Editura Petrion, București, 1996.
2. Bostan, Gheorghe, *Culegere de probleme de informatică*, Editura Lumina, Chișinău, 1996.
3. Cerchez, Emanuela; Șerban, Marinel, *Informatica. Manual pentru clasa a X-a*, Editura Polirom, Iași, 2000.
4. Corici, C.; Mânz, D.; Simulescu, A.; Șerban, M., *Limbajul Pascal*, Editura Libris, 1992.
5. Cormen, Thomas; Leiserson, Charles; Rivest, Ronald, *Introduction to Algorithms*, The Massachusetts Institute of Technology, 1990.
6. Croitoru, Cornelius, *Tehnici de optimizare combinatorie*, Editura Universității „Al.I. Cuza”, Iași, 1992.
7. Frâncu, Cătălin, *Psihologia concursurilor de informatică*, Editura L&S Infomat, București, 1997.
8. Horowitz, Ellis; Sahni, Sartaj; Anderson-Freed, Susan, *Fundamentals of Data Structures in C*, Computer Science Press, New York, 1993.
9. Jamsa, Kris; Klander, Lars, *Total despre C și C++*, Editura Teora, București, 2001.
10. Lucanu, Dorel, *Bazele proiectării programelor și algoritmilor*, Editura Universității „Al.I. Cuza”, Iași, 1996.
11. Mateescu (Cerchez), Emanuela; Maxim, Ioan, *Arbore*, Editura Țara Fagilor, Suceava, 1996.
12. Mitrana, Victor, *Provocarea algoritmilor*, Editura Agnii, București, 1994.
13. Niculescu, Șt.; Cerchez, Em. ș.a., *Bacalaureat și atestat*, Editura L&S Infomat, București, 1999.
14. Oltean, Mihai, *Programarea jocurilor matematice*, Editura Microinformatica, Cluj-Napoca, 1996.
15. Tomescu, Ioan, *Probleme de combinatorică și teoria grafurilor*, Editura Didactică și Pedagogică, București, 1981.
16. Tomescu, Ioan; Leu, Adrian, *Matematică aplicată în tehnica de calcul*, Editura Didactică și Pedagogică, București, 1982.