MINISTERUL EDUCAŢIEI
AL REPUBLICII MOLDOVA

МИНИСТЕРСТВО
ПРОСВЕЩЕНИЯ РЕСПУБЛИКИ
МОЛДОВА

Olimpiada Republicană de Informatică

ediţia anului 2015

Chişinău 2015

Olimpiada Republicană la Informatică. Ediţia anului 2015. Chişinău, 75 pag.

Lucrarea conţine problemele propuse la Olimpiada Republicană la Informatică a elevilor, ediţia anului 2015 şi la barajul de selectare a lotului olimpic pentru participare la etapele internaţionale ale olimpiadei de Informatică. Pentru fiecare problemă sunt descrise algoritmul şi soluţia respectivă pentru mediul de programare PASCAL sau limbajele C/C++.
Materialele Olimpiadei pot fi de un real folos atât elevilor, cât şi cadrelor didactice la studierea aprofundată a Informaticii, pregătirea pentru olimpiadele de Informatică.

La elaborarea ediţiei au contribuit:

Beşliu Victor	dr., prof.univ.int., UTM
Prisăcaru Angela	consultant, Ministerul Educaţiei
Bolun Ion	dr.hab., prof.univ., ASEM
Croitor Mihai	lector univ., USM
Globa Angela	lector sup. univ., UST (Chişinău)
Hîncu Boris	dr., conf. univ., USM
Negară Corina	dr., lector sup. univ., US „A. Russo”, m.Bălţi
Bercu Igor	lector sup. univ., ASEM
Dolghier Constantin	Informatician

CUPRINS

Dragi elevi și stimați profesori, care vă dăruiți acestui domeniu miraculos al gîndirii și activității umane pe nume *Informatica*

Participarea la o olimpiadă republicană reprezintă, fără îndoială, un succes, dar e bine să ştim că nu e cel mai mare şi, poate, nici cel mai important din viaţa voastră... Deşi sună oarecum ciudat, motivul pentru care fac aceste afirmaţii este că vreau să înţelegeți cu toţii câteva lecţii importante.

Mai întîi, e bine să ştiți că în viaţă nu avem doar succese... de nenumărate ori pierdem, sau, altfel spus, greşim, şi, în ambele situaţii, trebuie să mergem mai departe. Acest lucru ne face să credem că succesul în viaţă nu se defineşte doar prin cîştigarea unor competiţii, ci, mai ales, prin cum ne raportăm la acele experienţe, dar şi la momentele cînd pierdem sau greşim, pentru că, indiferent de context, cel mai important este să învăţăm ceva care să aibă impact pozitiv asupra vieţii noastre.

În al doilea rând, e bine să privim competiţia dintr-o perspectivă nouă, şi anume ca fiind una cu noi înşine, în primul rînd, şi nu cu cei din jur, în sensul că aceasta trebuie să constituie lupta noastră de a ne depăşi ultima performanţă realizată. Această abordare ne va ajuta să creştem continuu şi, mai ales, să ne raportăm corect şi cu înţelegere la ceilalţi competitori, ceea ce va face lupta frumoasă şi provocatoare pentru toţi.

În al treilea rând, oricât de preţios ar fi premiul pe care îl primim, cel mai important este drumul până la obţinerea lui, toată munca şi eforturile, trăirile şi descoperirile, hotărârea şi încrederea în noi şi în cei care ne-au fost mentori sunt vectorii care ne vor plasa pe o orbită a succesului adevărat şi de durată, care se măsoară în ceea ce faci în viaţă, pentru tine şi pentru cei din jurul tău.

În numele Consiliului Olimpic al Olimpiadei Republicane de Informatică

Victor BEŞLIU
prof. univ. int., dr., şeful catedrei "Automatica şi Tehnologii Informaţionale", UTM
preşedintele Consiliului Olimpic al Olimpiadei Republicane de Informatică, 2015

Problemele pentru elevii claselor 7 – 9

Denumirea problemei	Numărul de puncte alocat problemei	Denumirea fişierului sursă	Denumirea fişierului de intrare	Denumirea fişierului de ieşire
Acoperiș	100	ACOPERIS.PAS ACOPERIS.C ACOPERIS.CPP	ACOPERIS.IN	ACOPERIS.OUT
Album	100	ALBUM.PAS ALBUM.C ALBUM.CPP	ALBUM.IN	ALBUM.OUT
Împărțirea	100	IMPART.PAS IMPART.C IMPART.CPP	IMPART.IN	IMPART.OUT
Acoperire	100	ACOPERIRE.PAS ACOPERIRE.C ACOPERIRE.CPP	ACOPERIRE.IN	ACOPERIRE.OUT
Cărţi	100	CARTE.PAS CARTE.C CARTE.CPP	CARTE.IN	CARTE.OUT
Puncte în plan	100	PUNCTE.PAS PUNCTE.C PUNCTE.CPP	PUNCTE.IN	PUNCTE.OUT
Total	**600**	-	-	-

Acoperiş

În Ecolandia acoperişurile clădirilor sunt formate din baterii solare. În timpul unei furtuni s-a produs un fulger ce a deteriorat parţial acoperişul unei clădiri. Specialiştii de la departamentul Situaţii Excepţionale au scanat acoperişul şi au format harta lui: cu **1** au fost notate sectoarele acoperişului deteriorate de fulger, iar cu **0** – cele rămase întregi. Acoperişul este de formă pătrată $m \times m$. Pentru refacerea acoperişului sunt necesare blocuri de baterii solare de lăţime **1** şi lungime x, $x = 1, 2, ..., m$. Din cauza unor dispozitive speciale care unesc blocurile, ele pot fi plasate pe acoperiş toate doar vertical sau toate doar orizontal.

Sarcină. Alcătuiţi un program, care ar determina numărul minim total de blocuri şi numărul de blocuri de fiecare tip pentru a repara acoperişul.

Date de intrare. Fişierul text de intrare **ACOPERIS.IN** conţine pe prima linie un număr natural, egal cu dimensiunea m a laturii acoperişului, iar pe fiecare din următoarele m linii – câte o secvenţă de m cifre **1** sau **0** separate printr-un spaţiu.

Date de ieşire. Fişierul **ACOPERIS.OUT** va conţine pe prima linie un număr întreg egal cu numărul minim de blocuri, iar pe următoarele linii, ordonate crescător după x, secvenţe din două numere întregi, separate printr-un spaţiu: x şi num, unde x este lungimea (tipul) blocului, iar num este numărul de blocuri de lungime x.

Observaţie: Dacă numărul de blocuri pe verticală şi pe orizontală sunt egale, se va afişa varianta pe verticală.

Restricţii:

- $1 \le m \le 200$
- Nu se vor afişa blocurile de lungime x, a căror număr este zero (0)
- Timpul de execuţie nu va depăşi 1 secundă
- Programul va folosi cel mult 32 MB de memorie operativă.

Fişierul sursă va avea denumirea **ACOPERIS.PAS, ACOPERIS.C** sau **ACOPERIS.CPP.**

Exemplul 1

ACOPERIS.IN	ACOPERIS.OUT	Explicaţie
5 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 0 1 0 1 0	7 1 5 3 1 5 1	Blocurile vor fi plasate pe orizontală, în total 7 blocuri (5 blocuri de lungime 1, 1 bloc de lungime 3 şi 1 bloc de lungime 5); în cazul plasării blocurilor pe verticală vom avea nevoie de 7 blocuri de lungime 1 (1 – 7) şi 3 blocuri de lungime 2 (2 – 3), în total 10 blocuri.

Exemplul 2

ACOPERIS.IN	ACOPERIS.OUT	Explicaţie
10 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 0 0 0 1 1 1	20 1 9 2 5 3 1 4 2 10 3	S-a afişat varianta pe verticală. Numărul de blocuri pe orizontală este tot 20 (1- 2, 2- 4, 3- 8, 4 -4, 5 -2). Dacă numărul de blocuri pe verticală şi pe orizontală sunt egale, se va afişa varianta pe verticală.

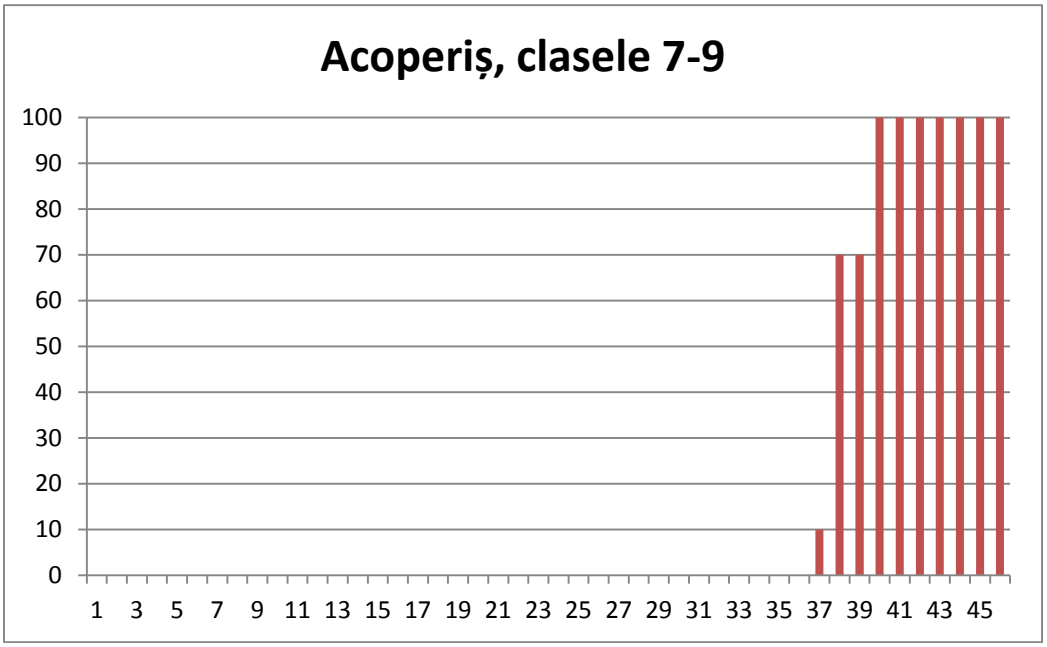
Rezolvare

Rezolvarea problemei necesită competențe de a lucra cu tipul de date tablou (unidimensionale și bidimensionale). Numărul de blocuri de lungimea `m`, care pot fi plasate pe orizontală (verticală) se vor păstra într-un vector cu `m` elemente (`blok` și `blok1`, respectiv), adică `bloc[i]` va reține numărul de blocuri de lungimea `i`, care pot fi plasate pe orizontală, iar `bloc1[i]` va reține numărul de blocuri de lungimea `i`, care pot fi plasate pe verticală. Acest lucru este necesar, deoarece trebuie să se țină cont de observația: dacă numărul de blocuri pe verticală și pe orizontală sunt egale, se va afișa varianta pe verticală. Odată cu incrementarea numărului de blocuri de fiecare lungime, se va introduce câte un contor (`orizontal`, `vertical`), care va număra numărul total de blocuri care pot fi plasate sau numai vertical sau numai orizontal pe acoperiș.

```
Program p1;
{clasele 07-09}

type vector=array[1..200] of longint;
     matrice=array[1..200,1..200] of byte;

var a:matrice;
    blok,blok1:vector;
    i,j,k,m:byte;
    orizontal,vertical:integer;
    f:text;

Begin
 assign(f,'ACOPERIS.IN');
 reset(f);
 readln(f,m);
 for i:=1 to m do
  for j:=1 to m do
   read(f,a[i,j]);
 close(f);

 orizontal:=0;
 for i:=1 to m do
  begin
   j:=1;
   while j<=m do
    begin
     k:=0;
     while (a[i,j]<>0) and (j<=m) do
      begin
       inc(k);inc(j);
      end;
     if k<>0 then begin inc(blok[k]);inc(orizontal);end;
     inc(j);
    end;
  end;

 vertical:=0;
 for j:=1 to m do
  begin
   i:=1;
   while i<=m do
    begin
     k:=0;
     while (a[i,j]<>0) and (i<=m) do
      begin
       inc(k);inc(i);
      end;
     if k<>0 then begin inc(blok1[k]);inc(vertical);end;
     inc(i);
```

```
    end;
  end;

 assign(f,'ACOPERIS.OUT');
 rewrite(f);
 if vertical>orizontal then
  begin
   writeln(f,orizontal);
   for i:=1 to m do
    if blok[i]<>0 then writeln(f,i,' ',blok[i]);
  end                  else
  begin
   writeln(f,vertical);
   for i:=1 to m do
    if blok1[i]<>0 then writeln(f,i,' ',blok1[i]);
  end;

 close(f);

End.
```

Punctajul acumulat de fiecare competitor



Notă: Pe orizontală sunt competitorii, simbolizaţi prin numerele 1, 2, 3, ... ş.a.m.d., iar pe verticală punctajul fiecăruia din ei

Album

Ionel este un fan al muzicii și vrea să descarce un album al interpretului preferat. O piesă poate fi descărcată în A secunde (timpul de descărcare depinde de volumul fișierului). Piesele (fișierele respective) pot fi descărcate consecutiv în seturi – câte una sau, în paralel, câte două sau, cel mult, câte trei. Dacă Ionel descarcă câteva fișiere în paralel, timpul de descărcare crește: pentru două fișiere descărcate în paralel – cu 40%, iar pentru trei fișiere – cu 60%. Dacă se dă la descărcare un set de fișiere, următoarea descărcare Ionel o poate face doar după descărcarea completă a setului precedent. Piesele sunt descărcate în ordinea lor din album.

Sarcină. Alcătuiți un program, care ar determina timpul minim (în secunde), necesar pentru descărcarea albumului.

Date de intrare. Fişierul text **`ALBUM.IN`** conține pe prima linie un număr întreg N – numărul de piese, iar pe fiecare din următoarele N linii timpul de descărcare a piesei respective.

Date de ieșire. Fișierul text **`ALBUM.OUT`** va conține un număr întreg, obţinut prin rotunjirea până la întregi prin adaos a valorii timpului minim (în secunde), necesar pentru descărcarea albumului.

Restricţii:

- $1 \le N \le 100$
- Timpul de execuţie nu va depăşi 1 secundă
- Programul va folosi cel mult 32 MB de memorie operativă.

Fişierul sursă va avea denumirea **`ALBUM.PAS`**, **`ALBUM.C`** sau **`ALBUM.CPP`**.

Exemplu

`ALBUM.IN`	**`ALBUM.OUT`**
4 6 8 3 7	19

Rezolvare

Pentru a determina timpul minimal de descărcare a pieselor trebuie să le grupăm în două sau trei fișiere. Important este de a determina cea mai eficientă modalitate de a le grupa. Vom folosi invarianții.

Fie Z[i] este timpul minim de descărcare a primelor i piese în ordinea în care esle apar în album. Este evident că dacă Z[0]=0. În caz general, însă, Z[i] depinde de Z[i-1], Z[i-2], Z[i-3].
Z[i]:=min(Z[i-1]+X[i-1], Z[i-2]+ max(X[i-2]*1.4, X[i-1]*1.4, 0), Z[i-3]+ max(X[i-3]*1.6, X[i-2]*1.6, X[i-1]*1.6))

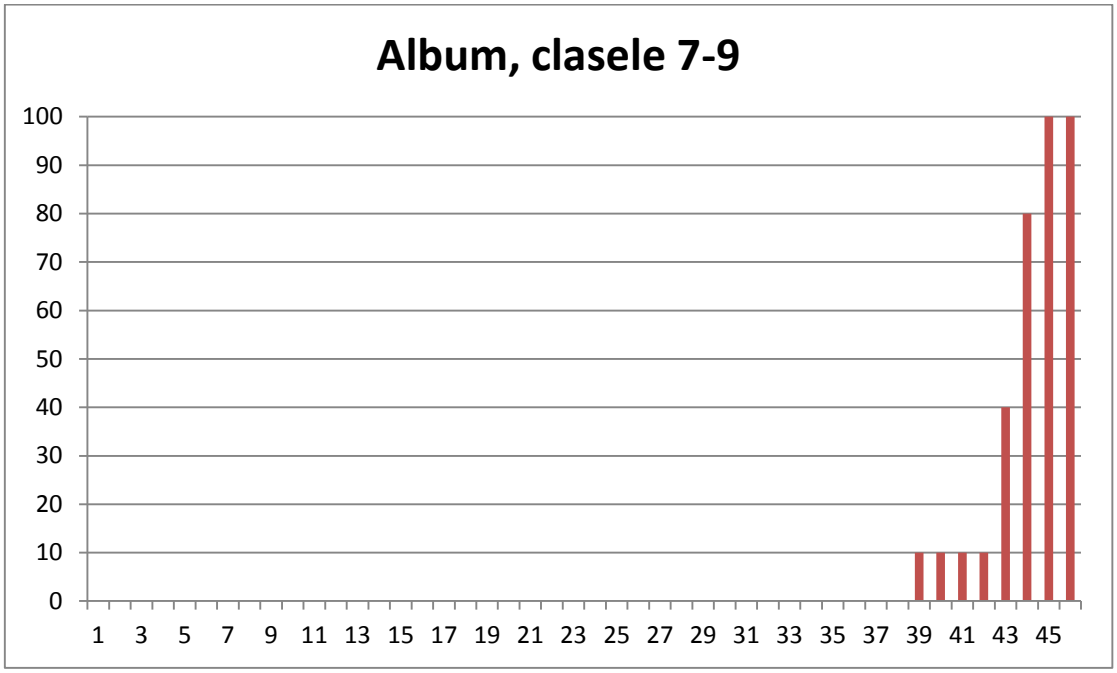
Această formulă precaută trei situații:

1. Z[i-1]+X[i-1] - timpul pentru descărcarea primelor i-1 piese este minim și se va descărca doar piesa i;
2. Z[i-2]+ max(X[i-2]*1.4, X[i-1]*1.4, 0) - timpul pentru descărcarea primelor i-2 piese este minim, deci pentru piesele X[i-2] și X[i-1] va fi determinat timpul minim pentru descărcarea lor paralelă;
3. Z[i-3]+ max(X[i-3]*1.6, X[i-2]*1.6, X[i-1]*1.6) - timpul pentru descărcarea primelor i-3 piese este minim, deci pentru piesele X[i-3], X[i-2] și X[i-1] va fi determinat timpul minim pentru descărcarea lor paralelă.

Aceste trei situații se evaluează și se determină valoare minimă.

```
Program album;
{clasele 07-09}

Uses math;

Function max(a,b,c:real):real;
  var m:real;
  Begin
    if (a>=b) and (a>=c) then m:=a;
    if (b>=a) and (b>=c) then m:=b;
    if (c>=a) and (c>=b) then m:=c;
    max:=m;
  end;

Function min(a,b,c:real):real;
  var t:real;
  Begin
    if (a<=b) and (a<=c) then t:=a;
    if (b<=a) and (b<=c) then t:=b;
    if (c<=a) and (c<=b) then t:=c;
   min:=t;
  end;

  var N:integer;
      X:array[1..5001] of real;
      Z:array[1..5001] of real;
      i, j :integer;
      f,h : real;
      fin:text;
      fout:text;

 Begin
     Assign(fin,'ALBUM.IN');
     Assign(fout,'ALBUM.OUT');
     Reset(fin);
     read(fin,N);
     for i:=1 to N do read (fin, X[i]);
     close(fin);

    Z[1]:=0;
    Z[2]:=X[1];
    Z[3]:=max(X[1]*1.4,X[2]*1.4,0);
    for i:= 4 to N+1 do
    Begin
      f:=max(X[i-2]*1.4, X[i-1]*1.4, 0);
      h:=max(X[i-3]*1.6, X[i-2]*1.6, X[i-1]*1.6);
      Z[i]:=min(Z[i-1]+X[i-1],  Z[i-2]+f, Z[i-3]+h);
       end;

   rewrite(fout);
  writeln (fout, ceil(Z[N+1]));

   close (fout);

end.
```

Punctajul acumulat de fiecare competitor



Notă: Pe orizontală sunt competitorii, simbolizaţi prin numerele 1, 2, 3, ... ş.a.m.d., iar pe verticală punctajul fiecăruia din ei

Împărțirea

Pasiunea Angelicăi este numerologia (numerologia este cea mai ușoară dintre științele oculte; se ocupă de studiul numerelor care determină și reflectă caracteristicile, talentele, motivațiile și drumul în viață ale unei persoane). De aceea pentru ea orice cifră din înscrierea unui număr are mare importanță. Se știe că calculatoarele moderne, la împărțirea a două numere naturale, reprezintă rezultatul cu un număr anumit de cifre după virgulă.

Sarcină. Alcătuiţi un program, care pentru două numere naturale *m* şi *n* ar determina numărul *m/n* în formă zecimală completă (cu perioadă, dacă ea există).

Date de intrare. Fişierul text **`IMPART.IN`** conţine pe o singură linie numerele *m* şi *n* despărţite printr-un spaţiu.

Date de ieşire. Fişierul text **`IMPART.OUT`** va conţine un număr zecimal sub forma: *`0,partea neperiodică`* sau *`0,(parte periodică)`*, sau *`0,partea neperiodică(partea periodică).`*

Restricţii:

- $1 \leq m < 100000$, $1 < n \leq 100000$, $m < n$
- Timpul de execuţie nu va depăşi 1 secundă
- Programul va folosi cel mult 32 MB de memorie operativă.

Fişierul sursă va avea denumirea **`IMPART.PAS`**, **`IMPART.C`** sau **`IMPART.CPP`**.

Exemplul 1

IMPART.IN	IMPART.OUT
1 5	0,2

Exemplul 2

IMPART.IN	IMPART.OUT
3 9	0,(3)

Exemplul 3

IMPART.IN	IMPART.OUT
2 12	0,1(6)

Rezolvare

Fie data fracția ordinară, ireductibilă $\frac{2}{13}$. Să calculăm manual fracția zecimală:

```
2        | 13
0        +----------
-        | 0.153846
20
13
--
 70
 65
 --
  50
  39
  --
  110
   104
   ---
     60
     52
     --
      80
      78
      --
       2
```

Restul obținut, adică numărul 2 este exact numărul cu care am început. Deci putem face concluzia că, fracția zecimală este peiodică, adică $\frac{2}{13} = 0,(153846)$.

Pentru rezolvarea acestei probleme vom aplica următoarele definiții și teoreme:

Definiția 1. Fracția zecimală $0,a_1a_2...a_n$ se numește fracție zecimală finită, unde *n* este numărul de cifre după virgulă a fracției.

Definiția 2. Fracția zecimală $0,a_1a_2...a_n...$ se numește fracție zecimală periodică simplă cu perioada *s*, dacă pentru $\forall a_i$ are loc relația $a_k = a_{k+s}$, unde *s* este cel mai mic număr natural cu așa proprietate. O fracție zecimală periodică simplă se notează $0,(a_1a_2...a_s)$.

Definiția 3. Fracția zecimală $0,a_1a_2...a_n...$ se numește fracție zecimală periodică mixtă cu perioada *s*, dacă $\exists m \in N, m > 0$, încît pentru $\forall k \in N, k > m$ are loc relația $a_k = a_{k+s}$, unde *s* este cel mai mic număr natural cu așa proprietate. O fracție zecimală periodică mixtă se notează $0,a_1a_2...a_m(a_{m+1}a_{m+2}...a_{m+s})$.

Teorema 1. Dacă numitorul unei fracții ordinare ireductibile $\frac{a}{b}$ este reciproc prim cu 10, adică (*b*,10)=1, atunci fracția $\frac{a}{b}$ se transformă într-o fracţie zecimală periodică simplă, perioada având cel mult *b*-1 cifre. Perioada *s* se poate calcula din relația: $10^s \equiv 1 \bmod b$, unde *s* este cel mai mic număr care satisfice această relație.

Teorema 2. Fie $\frac{a}{b}$ o fracție ordinară ireductibilă. Fie că descompunerea în produs de factori primi a numărului *b* are forma: $b = 2^k \cdot 5^t \cdot p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot ... \cdot p_q^{\alpha_q}$, atunci fracția ordinară ireductibilă $\frac{a}{b}$ poate fi reprezentată sub forma unei fracții zecimale periodice mixte $0,a_1a_2...a_m(a_{m+1}a_{m+2}...a_{m+s})$. Perioada *s* a fracției zecimale periodice mixte se va calcula din relația: $10^s \equiv 1 \bmod(p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot ... \cdot p_q^{\alpha_q})$, unde *s* este cel mai mic număr care satisfice această relație, iar numărul de cifre neperiodice *m* este egal cu valoarea maximă dintre exponenții numerelor prime 2 și 5, adică $m = \max(k,t)$.

Algoritmul este:

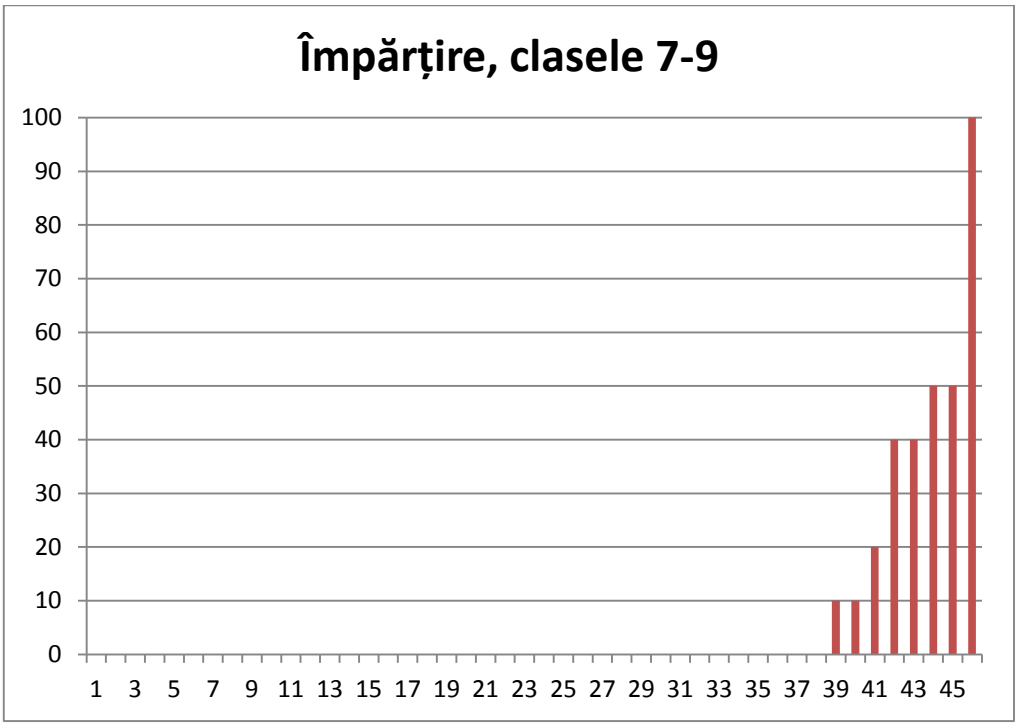
1. scriem 0,
2. calculăm numărul de cifre neperiodice *m*;
3. efectuăm *m* împărțiri, scriem cele *m* cifre obținute și reținem ultimul rest *R*;
4. dacă *R=0,* atunci fracția este neperiodică și finită; stop;
5. dacă $R \neq 0$, atunci:
   5.1. scriem ‘(‘;
   5.2.efectuăm împărțirea și comparăm restul obținut *r* cu *R*;
   5.3. dacă *r=R*, scriem ‘)’;stop.

Memoria este *O*(1) (constantă), timpul de rulare *O*(*n*).

```
Program p1;
{clasele 07-09}

var m,n,c,rest,stop:longint;
    x,y,nep,cif,w:byte;
    f:text;

Function CMMDC(a,b:longint):longint;
begin
 while a<>b do
  if a>b then a:=a-b else b:=b-a;
 CMMDC:=a;
end;

Procedure Puteri(a:longint;var k,t:byte);
var q:boolean;
begin
 k:=0;t:=0;
 repeat
  q:=true;
  if a mod 2=0 then begin inc(k);a:=a div 2;q:=false;end;
  if a mod 5=0 then begin inc(t);a:=a div 5;q:=false;end;
 until q;
end;

Begin
 assign(f,'IMPART.IN');
 reset(f);
 read(f,m,n);
 close(f);

 c:=CMMDC(m,n);
 m:=m div c;n:=n div c;
 Puteri(n,x,y);
 if x>y then nep:=x else nep:=y;

 assign(f,'IMPART.OUT');
 rewrite(f);
 write(f,'0,');

 if nep=0 then
                  begin
                     rest:=m; stop:=rest;
                  end
          else
                begin
                   repeat
                        m:=m*10;
                        cif:=m div n;
                        rest:=m mod n;
                        m:=rest;
                        inc(w);
                        write(f,cif);
                  until w>=nep;
                 stop:=rest;
              end;

 if rest<>0 then
             begin
                 write(f,'(');
                  repeat
                     m:=m*10;
                     cif:=m div n;
                     rest:=m mod n;
```

```
                    m:=rest;
                    write(f,cif);
                until rest=stop;
            end;

 if stop<>0 then write(f,')');

 close(f);

End.
```

Punctajul acumulat de fiecare competitor



Notă: Pe orizontală sunt competitorii, simbolizaţi prin numerele 1, 2, 3, ... ş.a.m.d., iar pe verticală punctajul fiecăruia din ei

Acoperire

Dintr-un dreptunghi cu laturile *m* şi *n* este tăiat un pătrat cu latura 1 şi coordonatele (*i, j*) ale colţului din dreapta-sus al lui.

Sarcină. Alcătuiţi un program, care ar determina dacă figura obţinută poate fi acoperită cu dreptunghiuri 1x2 fără suprapunerea lor.

Date de intrare. Fişierul text de intrare **ACOPERIRE.IN** conţine pe prima linie 2 numere întregi *m* și *n* – dimensiunile laturilor dreptunghiului, separate prin spaţiu, iar pe a doua linie - coordonatele (*i, j*) ale pătratului unitar ce a fost tăiat din dreptunghiul iniţial, de asemenea separate prin spaţiu.

Date de ieşire. Fişierul **ACOPERIRE.OUT** va conţine pe prima linie cuvântul `Yes`, dacă acoperirea figurii este posibilă, sau cuvântul `No`, în caz contrar.

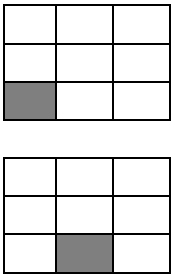
Restricţii:

- $1 \le i,\ j \le m, n \le 100000$
- Timpul de execuţie nu va depăşi 1 secundă
- Programul va folosi cel mult 16 MB de memorie operativă.

Fişierul sursă va avea denumirea **ACOPERIRE.PAS, ACOPERIRE.C** sau **ACOPERIRE.CPP.**

Exemplu

ACOPERIRE.IN	ACOPERIRE.OUT
2 3 3 1 1 3 3 1 2	Yes No

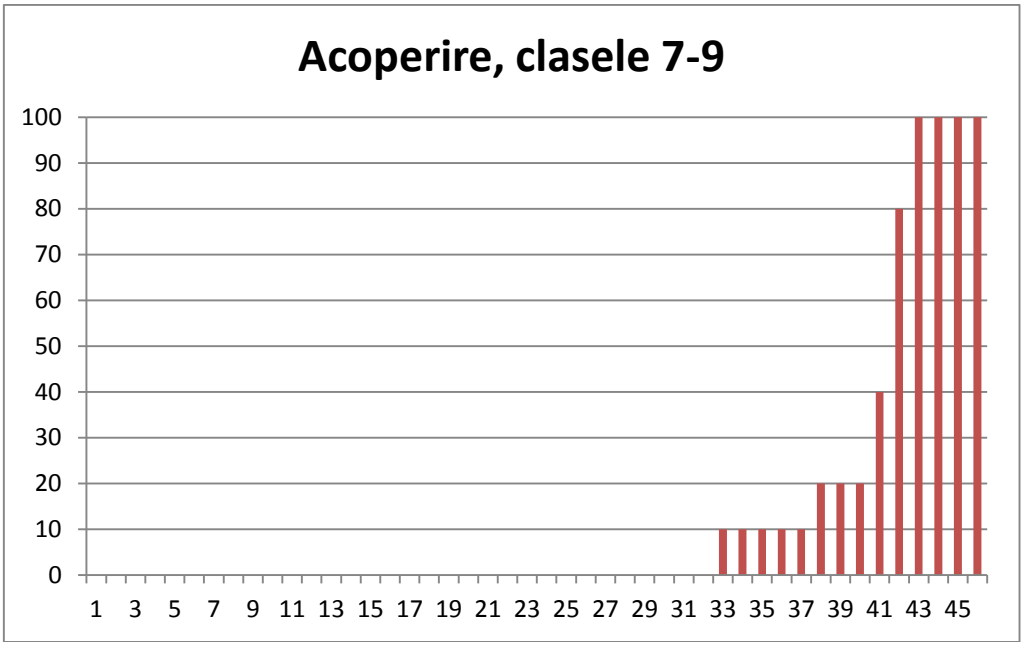


Soluţia

```
{clasele 07-09}

#include <iostream>
#include <cstdio>

int main(int argc, char** argv) {
    int m, n, num_tests;
    int i, j;
      freopen("ACOPERIRE.IN", "r", stdin);
      freopen("ACOPERIRE.OUT", "w", stdout);
    std::cin >> num_tests;
    while (num_tests--) {
        std::cin >> m >> n >> i >> j;

        if ((m % 2 == 0) || (n % 2 == 0)) {
            std::cout << "No";
        } else if ((m == 1) && (j % 2 == 0)) {
            std::cout << "No";
        } else if ((n == 1) && (i % 2 == 0)) {
            std::cout << "No";
        } else if ((i + j) % 2 == 1) {
```

```
            std::cout << "No";
        } else {
            std::cout << "Yes";
        }
        std::cout << std::endl;
    }
    return 0;
}
```

Punctajul acumulat de fiecare competitor



Notă: Pe orizontală sunt competitorii, simbolizaţi prin numerele 1, 2, 3, ... ş.a.m.d., iar pe verticală punctajul fiecăruia din ei

Cărţi

Marele magnat BitMan are în unul din depozitele sale *n* cutii, numerotate de la 1 la *n*, pline cu cărți. Nu există cutii cu același număr de cărți. Cutia *i* conţine x_i cărți ($x_i \neq x_j$, $1 \leq i, j \leq n$). BitMan preconizează să meargă la *n* orfelinate şi să le împartă cărți în mod egal, fără rest, luând cu sine doar *k* cutii ($1 \leq k \leq n$).

Sarcină. Alcătuiţi un program, care ar determina numărul maxim de cărţi, pe care BitMan ar putea să le doneze fiecărui orfelinat.

Date de intrare. Fişierul text de intrare **`CARTE.IN`** conţine pe prima linie un număr natural *n* - numărul de cutii din depozit, iar pe a doua linie - numerele naturale $x_1, x_2, \ldots, x_n$, separate prin spațiu - numărul de cărți din cutiile respective.

Date de ieşire. Fişierul text de ieşire **`CARTE.OUT`** va conține pe prima linie un număr natural - cantitatea de cărți, pe care o va primi fiecare orfelinat.

Restricţii:

- $1 \leq n \leq 255$; $1 \leq x_1, x_2, \ldots, x_n \leq 1000$
- Timpul de execuţie nu va depăşi 1 secundă
- Programul va folosi cel mult 32 MB de memorie operativă.

Fişierul sursă va avea denumirea **`CARTE.PAS`**, **`CARTE.C`** sau **`CARTE.CPP`**.

Exemplu

`CARTE.IN`	**`CARTE.OUT`**	**Explicaţie**
5 12 9 14 17 11	8	BitMan poate lua cutiile 1, 4 şi 5 sau cutiile 2, 3 şi 4, fiecare orfelinat primind câte 8 cărţi. Orice alt set de cutii nu va permite lui BitMan să împartă cărți în mod egal fără rest, distribuind mai mult de 8 cărți fiecărui orfelinat.

Rezolvare

Nu se va cere numărul cutiilor, pe care trebuie să lea ia cu sine BitMan, deoarece problema admite mai multe soluții (de exemplu 1,4,5). Însă numărul maxim de cărți pe cate îl va primi fiecare orfelinat este unic.

Rezolvarea problemei se reduce la rezolvarea problemei suma maximă divizibilă cu *n*. Se citesc *n* - număr natural şi *n* numere naturale. Se cere să se tipărească cea mai mare sumă care se poate forma utilizând cele *n* numere naturale (fiecare număr poate participa o singură dată în calculul sumei) şi care se divide cu *n*.

Observaţie: Problema admite întotdeauna soluţie.

În general, avem 2^n submulţimi ale unei mulţimi cu *n* elemente (acest număr include şi submulţimea vidă). Dacă am analiza fiecare sumă obținută cu scopul de a o alege pe cea cea maximă divizibilă cu *n*, timpul de calcul ar fi foarte mare ($O(2^n)$).

Problema poate fi rezolvată în *n* etape. La prima etapă (se ia în consideraţie numai primul număr citit) se poate forma o singură sumă. La etapa *i* se vor calcula sumele maxime care prin împărţirea la *n* dau resturile *(0,1,...,n-1)*. La ultima etapă se va tipări suma care fiind împărţită la *n* dă restul 0.

Notăm cu $n_1,...,n_n$ numerele date din enunțul problemei. Vom reține în vectorul *Suma* sumele maxime care împărțite la *n* dau resturile *(0,1,...,n-1)*. Deci, *Suma(0)* va reține suma maximă care împărțită la *n* dă restul *0*, *Suma(1)* va reține suma maximă care împărțită la *n* dă restul *1*, și așa mai departe, *Suma(n-1)* va reține suma maximă care împărțită la *n* va da restul *n-1*.

Dacă, în procesul de calcul, la un anumit moment avem nici o sumă care împărțită la *n* dă restul *i*, *Suma(i)* va retine numărul 0.

La pasul *i* se reţin sumele maxime care se pot forma cu primele *i* numere. Pentru a adăuga elementul *i+1* vom reține sumele maxime care se pot forma cu primele *i* numere în vectorul $Suma_i$ - conţinutul vectorului *S* după ce am prelucrat primele *i* numere. Așa dar, în $Suma_i(m)$ se va reține suma maximă care împărțita la *n* dă restul *m*, sumă obţinută cu primele *i* numere naturale.
După prelucrarea primului număr vom avea:

$$Suma_1(m) = \begin{cases} n_1, \ daca \ n_1 \bmod n = m \\ 0, \ in \ rest \end{cases}, \forall m \in \{0,1,...,.n-1\}$$

Din această formulă putem înțelege următoarele: cu un singur număr se poate forma o singură sumă, cea egală cu numărul respectiv. Această sumă va fi memorată în acea componentă a vectorului *Suma*, pentru care indicele este egal cu restul împărţirii numărului la *n*.

La pasul *i+1* vom avea:

$$Suma_{i+1}(m) = \max \begin{cases} n_{i+1}, \quad daca \quad n_{i+1} \bmod n = m \\ Suma_i(m) \\ Suma_i(p) + n_{i+1}, daca \quad (Suma_i(p) + n_{i+1}) \bmod n = m, \\ \forall i \in \{1,2,...,n-1\}, \forall m \in \{0,1,...,n-1\} \end{cases}$$

Formula de mai sus poate fi explicată în felul următor: vectorul Suma se actualizează pentru n_{i+1} astfel: fiecare componentă m a sa va avea ca valoare maximul dintre:

- valoarea n_{i+1} dacă restul împărţirii lui n_{i+1}, la n este m;
- vechea valoare;
- o altă suma reţinută la care dacă adunăm n_{i+1}, obţinem o valoare care împărţită la n dă restul m.

Corectitudinea algoritmului poate fi demonstrată folosind inducţia matematică. Cu primul număr se poate calcula o singură sumă, (care este și maximă). Fie $Suma_1, Suma_2, ..., Suma_i$ şirul primilor *i* vectori generaţi, iar sumele reţinute le presupunem optime. Pentru a calcula elementele vectorului S_{i+1} se va utiliza vectorul $Suma_i$ prin adăugarea elementului n_{i+1}. Deoarece în vectorul $Suma_i$ sunt reținute sumele optime obținute cu primele *i* numere, atunci vectorul $Suma_{i+1}$ va conţine sumele optime calculate cu ajutorul primelor *i+1* numere (nu se pot genera sume cu proprietatea cerută care să fie mai mari).c.t.d.
Algoritmul continuă până la citirea tuturor numerelor.

Elementele (numărul cutiilor) ce formează $Suma_i(m)$ se vor reține într-un vector de tip mulțime $Elem_i(m)$.

Exemplu rezolvat: Fie *n*=5 și conținutul celor *n* cutii 12 9 14 17 11.

Formăm sumele maxime posibile cu elementul **12**:

Rest	0	1	2	3	4
Suma	0	0	12	0	0
Elem	[]	[]	[1]	[]	[]
Suma1	0	0	0	0	0
Elem1	[]	[]	[]	[]	[]

Calcule: 12 mod 5=2.

Formăm sumele maxime posibile cu elementul **9:**

Rest	0	1	2	3	4
Suma	0	21	12	0	9
Elem	[]	[1,2]	[1]	[]	[2]
Suma1	0	0	12	0	0
Elem1	[]	[]	[1]	[]	[]

Calcule: 9 mod 5=4; 12+9=21, 21 mod 5=1;

Formăm sumele maxime posibile cu elementul **14:**

Rest	0	1	2	3	4
Suma	35	26	12	23	14
Elem	[1,2,3]	[1,3]	[1]	[2,3]	[3]
Suma1	0	21	12	0	9
Elem1	[]	[1,2]	[1]	[]	[2]

Calcule: 14 mod 5=4; 14>9 (true);
14+9=23; 23 mod 5=3; 14+12=26; 26 mod 5=1;26>21(true);
14+21=35; 35 mod 5=0;

Formăm sumele maxime posibile cu elementul **17**:

Rest	0	1	2	3	4
Suma	40	31	52	43	29
Elem	[2,3,4]	[3,4]	[1,2,3,4]	[1,3,4]	[1,4]
Suma1	35	26	12	23	14
Elem1	[1,2,3]	[1,3]	[1]	[2,3]	[3]

Calcule: 17 mod 5=2; 17>12 (true);
17+14=31; 31 mod 5=1; 31>26 (true);
17+23=40; 40 mod 5=0; 40>35 (true);
17+12=29; 29 mod 5=4; 29>14 (true);
17+26=43; 43 mod 5=3; 43>23 (true);
17+35=52; 52 mod 5=2; 52>17 (true);

Formăm sumele maxime posibile cu elementul **11**:

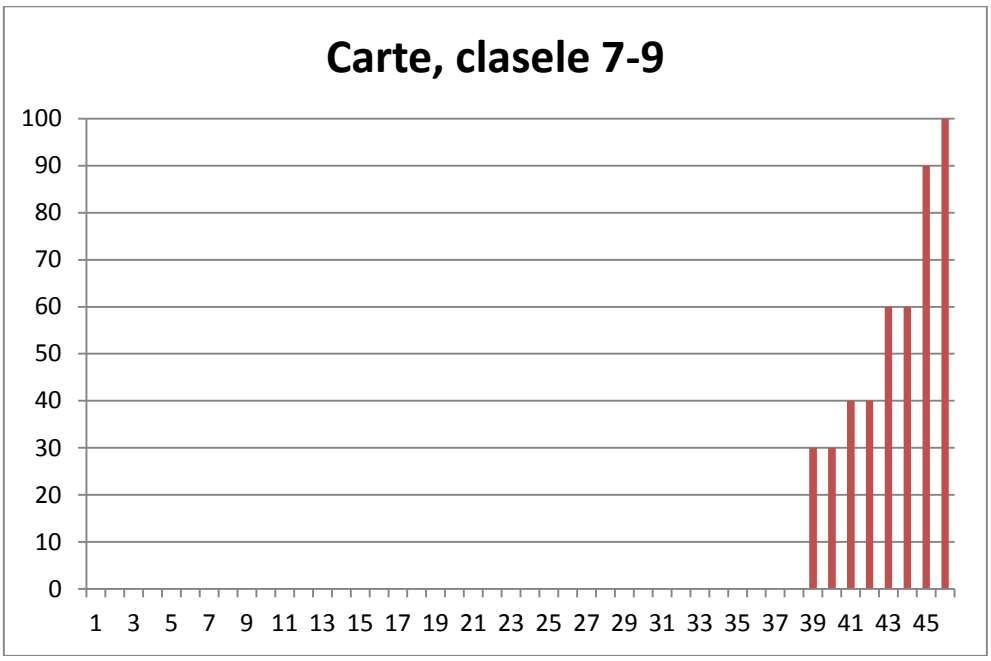
Rest	0	1	2	3	4
Suma	40	51	52	63	54
Elem	[2,3,4]	[2,3,4,5]	[1,2,3,4]	[1,2,3,4,5]	[1,3,4,5]
Suma1	40	31	52	43	29
Elem1	[2,3,4]	[3,4]	[1,2,3,4]	[1,3,4]	[1,4]

Calcule: 11 mod 5=1; 11>31 (false);
11+29=40; 40 mod 5=0; 40>40 (false); (a doua soluție 1,4,5)
11+43=54; 54 mod 5=4; 54>29 (true);
11+52=63; 63 mod 5=3; 63>43 (true);

11+31=42; 42 mod 5=2; 42>52 (false);
11+40=51; 51 mod 5=1; 51>31 (true);
Suma(0)=40; 40 div 5=8. Deci, rezultatul este 8.

```
Program cartile;
{clasele 07-09}

type vector=array[0..255] of longint;
     vector1=array[0..255] of set of byte;
     vector2=array[1..255] of integer;

var  n,i,j:byte;
     num:vector2;
     suma,suma1:vector;
     elem,elem1:vector1;
     f:text;

Begin
 assign(f,'CARTE.IN');
 reset(f);
 readln(f,n);

 for i:=1 to n do
  begin
     read(f,num[i]);
     elem[i-1]:=[];
     suma[i-1]:=0;
  end;

 close(f);

 suma[num[1] mod n]:=num[1];
 elem[num[1] mod n]:=[1];

 for i:=2 to n do
  begin
    suma1:=suma;
    elem1:=elem;
    if suma1[num[i] mod n]<num[j] then
          begin
              suma[num[i] mod n]:=num[i];
              elem[num[i] mod n]:=[i];
          end;
    for j:=0 to n-1 do
     if suma1[j]+num[i]>suma[(suma1[j]+num[i]) mod n] then
       begin
        suma[(suma1[j]+num[i]) mod n]:=suma1[j]+num[i];
        elem[(suma1[j]+num[i]) mod n]:=elem1[suma1[j] mod n]+[i] ;
       end;
  end;

 assign(f,'CARTE.OUT');rewrite(f);
 write(f,suma[0]div n);

 close(f);

End.
```

Punctajul acumulat de fiecare competitor



Notă: Pe orizontală sunt competitorii, simbolizaţi prin numerele 1, 2, 3, ... ş.a.m.d., iar pe verticală punctajul fiecăruia din ei

Puncte în plan

Într-un plan sunt date *n* puncte cu coordonatele x_i, y_i, i = 1, 2, …, *n*. Se va considera *cel mai mare pătrat* pătratul cu cea mai mare arie.

Sarcină. Alcătuiţi un program, care ar determina aria celui mai mare pătrat, care ar putea fi format de 4 dintre cele *n* puncte.

Date de intrare. Fişierul text de intrare **`PUNCTE.IN`** conţine pe prima linie un număr natural *n* – numărul de puncte în plan, iar pe fiecare din următoarele *n* linii - câte două numere întregi x_i şi y_i, separate prin spaţiu, coordonatele punctelor respective.

Date de ieşire. Fişierul text de ieşire **`PUNCTE.OUT`** va conţine pe prima linie un număr, egal cu aria celui mai mare pătrat.

Restricţii:

- $1 \le n \le 100$
- Timpul de execuţie nu va depăşi 1 secundă
- Programul va folosi cel mult 16 MB de memorie operativă.

Fişierul sursă va avea denumirea **`PUNCTE.PAS`**, **`PUNCTE.C`** sau **`PUNCTE.CPP`**.

Exemplu

PUNCTE.IN	**PUNCTE.OUT**
7 0 0 1 1 1 3 3 5 3 1 5 5 5 3	8

Rezolvare

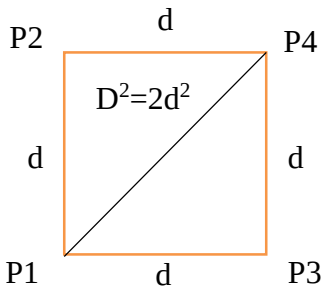
Pentru a rezolva această problemă este necesar de a cunoşte:

a. *Calculul distanţei dintre două puncta după coordonatele acestora*. Distanaţa dintre două puncte p cu coordonatele (x,y) şi q cu coordonatele (x,y), se calculează după formula:

$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

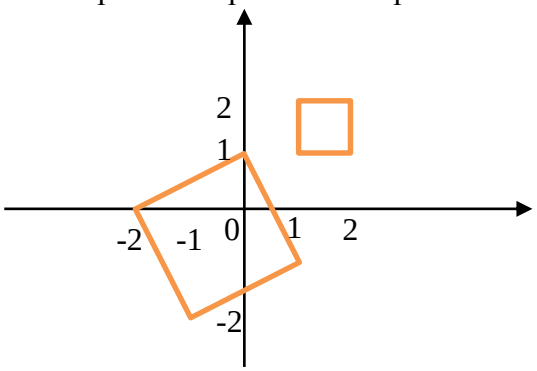
sau $d^2 = (x_1 - x_2)^2 + (y_1 - y_2)^2$

b. *Relaţiile dintre laturile şi diagonala pătratului.*



c. *Calculul ariei unui pătrat după latura lui*. Fie avem un pătrat cu latura d, atunci aria acestui pătrat va fi d^2.

Obs.1: Pentru păstrarea rezultatelor va fi ales tipul longint, deaorece nu are rost să extragem rădăcina pătrată (care poate fi o valoare reală). Aria pătratului va fi egală cu distanța dinte puncte la pătrat.
Obs. 2: Laturile pătratului pot să nu fie paralele cu axele de coordonate.

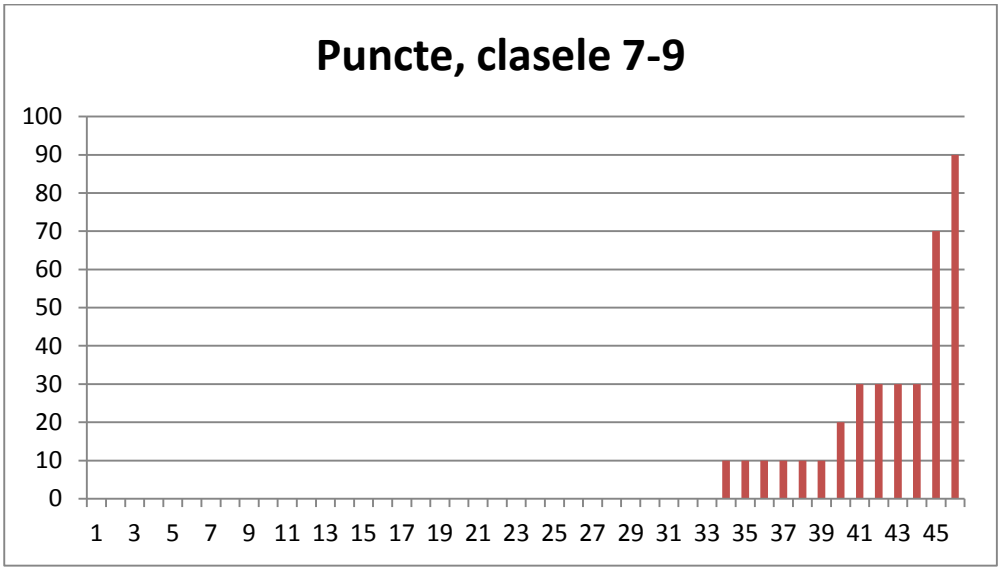



```
Program Puncte;
{clasele 07-09}

Type
    Point=record
     x:integer;
     y:integer;
end;

// Funcţia ce determină pătratul distanţei dintre punctele  'p' şi 'q'
function distSq(p:Point; q:Point):longint;
    Begin
     distSq:=(p.x - q.x)*(p.x - q.x) +
                (p.y - q.y)*(p.y - q.y);
    End;

// următoarea funcţie verifică dacă punctele (p1, p2, p3, p4) formează pătrat şi
dacă formează //returnează aria acestuia, în caz contrar returnează 0
Function aria(p1:Point; p2:Point; p3:Point; p4:Point):longint;
    var  d  :longint;
         d2 :longint;
         d3 :longint;
         d4 :longint;

   Begin
    aria:= -1;
    d2:= distSq(p1, p2);
    d3:= distSq(p1, p3);
    d4:= distSq(p1, p4);
// dacă distanţa dintre (p1, p2) şi (p1, p3) este aceeaşi, atunci pentru a fi
pătrat trebuie:
// 1) lungimea pătrată (p1, p4) este de două ori mai mare ca distanţa pătrată
(p1, p2)
// 2) p4 este egal depărtat de la  p2 şi p3
    if (d2 = d3) and (2*d2 = d4) then
        begin
      d:= distSq(p2, p4);
      if (d = distSq(p3, p4)) and (d = d2) then
          aria:= d
            else
                aria:=0;
       end;
    if (d3 = d4) and (2*d3 = d2) then
         begin
```

```
        d:= distSq(p2, p3);
        if (d = distSq(p2, p4)) and (d = d3) then
                  aria:= d
              else
                  aria:=0;
          end;
    if (d2 = d4) and (2*d2 = d3) then
          begin
       d:= distSq(p2, p3);
       if (d = distSq(p3, p4)) and (d = d2) then
             aria:=d
             else
                   aria:=0;
          end;
end;

  var P:array[1..101] of Point;
      x,y : integer;
      n:integer;
      aria_max: longint;
      aria_cur: longint;
      i1, i2, i3, i4, i:integer;
      fin:text;
      fout:text;

  Begin
     Assign(fin,'PUNCTE.IN');
     Assign(fout,'PUNCTE.OUT');
     Reset(fin);
     readln(fin,N);

     for i:=1 to N do
         begin
       read (fin, x);
             readln (fin, y);
            P[i].x := x;
            P[i].y := y;
         end;

     close(fin);

    aria_max:= 0; aria_cur:= 0;
    writeln;

    i1:=1;

    for i1:= 1 to n-3 do
         begin
            for i2:= i1+1 to n-2 do
        begin
         for i3:= i2+1 to n-1 do
                begin
                  for i4:= i3+1 to n do
                    begin
                      aria_cur:= aria(P[i1], P[i2], P[i3], P[i4]);
                      if (aria_cur > aria_max) then
                                  aria_max:= aria_cur;
                     end;
                  end;
              end;
         end;

    if (aria_max = 0) then
        aria_max := -1;
```

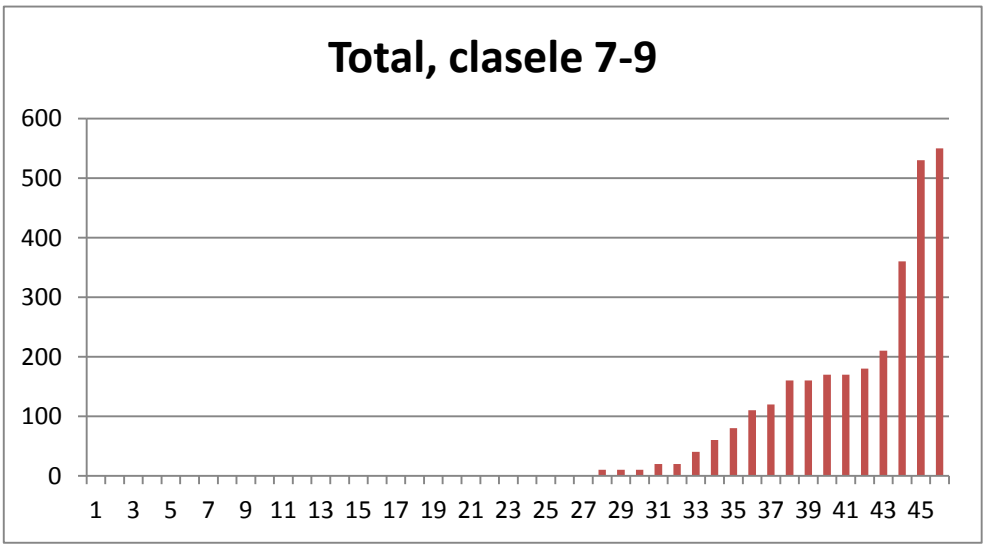
```
    rewrite(fout);
    writeln (fout, aria_max);

    close (fout);

end.
```

Punctajul acumulat de fiecare competitor



Notă: Pe orizontală sunt competitorii, simbolizaţi prin numerele 1, 2, 3, ... ş.a.m.d., iar pe verticală punctajul fiecăruia din ei

Punctajul total acumulat de fiecare competitor



Notă: Pe orizontală sunt competitorii, simbolizaţi prin numerele 1, 2, 3, ... ş.a.m.d., iar pe verticală punctajul fiecăruia din ei

Problemele pentru elevii claselor 10 – 12

Denumirea problemei	Numărul de puncte alocat problemei	Denumirea fişierului sursă	Denumirea fişierului de intrare	Denumirea fişierului de ieşire
Acoperiș	100	ACOPERIS.PAS ACOPERIS.C ACOPERIS.CPP	ACOPERIS.IN	ACOPERIS.OUT
Mulțimi proporţionale	100	MULTIMI.PAS MULTIMI.C MULTIMI.CPP	MULTIMI.IN	MULTIMI.OUT
Relaţii de simpatie	100	SIMPATIE.PAS SIMPATIE.C SIMPATIE.CPP	SIMPATIE.IN	SIMPATIE.OUT
Acoperire	100	ACOPERIRE.PAS ACOPERIRE.C ACOPERIRE.CPP	ACOPERIRE.IN	ACOPERIRE.OUT
Drepte în plan	100	DREPTE.PAS DREPTE.C DREPTE.CPP	DREPTE .IN	DREPTE.OUT
Segmente	100	SEGMENTE.PAS SEGMENTE.C SEGMENTE.CPP	SEGMENTE.IN	SEGMENTE.OUT
Total	**600**	-	-	-

Acoperiș

În Ecolandia acoperişurile clădirilor sunt formate din baterii solare. În timpul unei furtuni s-a produs un fulger ce a deteriorat parţial acoperişul unei clădiri. Specialiştii de la departamentul Situaţii Excepţionale au scanat acoperişul şi au format harta lui: cu **1** au fost notate sectoarele acoperişului deteriorate de fulger, iar cu **0** – cele rămase întregi. Acoperişul este de formă pătrată $m \times m$. Pentru refacerea acoperişului sunt necesare blocuri de baterii solare de lăţime **1** şi lungime x, $x = 1, 2, ..., m$. Din cauza unor dispozitive speciale care unesc blocurile, ele pot fi plasate pe acoperiş doar vertical sau doar orizontal.

Sarcină. Alcătuiţi un program, care ar determina numărul minim total de blocuri şi numărul de blocuri de fiecare tip pentru a repara acoperişul.

Date de intrare. Fişierul text de intrare **ACOPERIS.IN** conţine pe prima linie un număr natural, egal cu dimensiunea m a laturii acoperişului, iar pe fiecare din următoarele m linii – câte o secvenţă de m cifre **1** sau **0** separate printr-un spaţiu.

Date de ieşire. Fişierul **ACOPERIS.OUT** va conţine pe prima linie un număr întreg egal cu numărul minim de blocuri, iar pe următoarele linii, ordonate crescător după x, secvenţe din două numere întregi, separate printr-un spaţiu: x şi num, unde x este lungimea (tipul) blocului, iar num este numărul de blocuri de lungime x.

Observaţie: Dacă numărul de blocuri pe verticală şi pe orizontală sunt egale, se va afişa varianta pe verticală.

Restricţii:

- $1 \leq m \leq 200$
- Nu se vor afişa blocurile de lungime x, a căror număr este zero (0)
- Timpul de execuţie nu va depăşi 1 secundă
- Programul va folosi cel mult 32 MB de memorie operativă.

Fişierul sursă va avea denumirea **ACOPERIS.PAS, ACOPERIS.C** sau **ACOPERIS.CPP.**

Exemplul 1

ACOPERIS.IN	ACOPERIS.OUT	Explicaţie
5 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 0 1 0 1 0	7 1 5 3 1 5 1	Blocurile vor fi plasate pe orizontală, în total 7 blocuri (5 blocuri de lungime 1, 1 bloc de lungime 3 şi 1 bloc de lungime 5); în cazul plasării blocurilor pe verticală vom avea nevoie de 7 blocuri de lungime 1 (1 – 7) şi 3 blocuri de lungime 2 (2 – 3), în total 10 blocuri.

Exemplul 2

ACOPERIS.IN	ACOPERIS.OUT	Explicaţie
10 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 0 0 0 1 1 1	20 1 9 2 5 3 1 4 2 10 3	S-a afişat varianta pe verticală. Numărul de blocuri pe orizontală este tot 20 (1- 2, 2- 4, 3- 8, 4 -4, 5 -2).

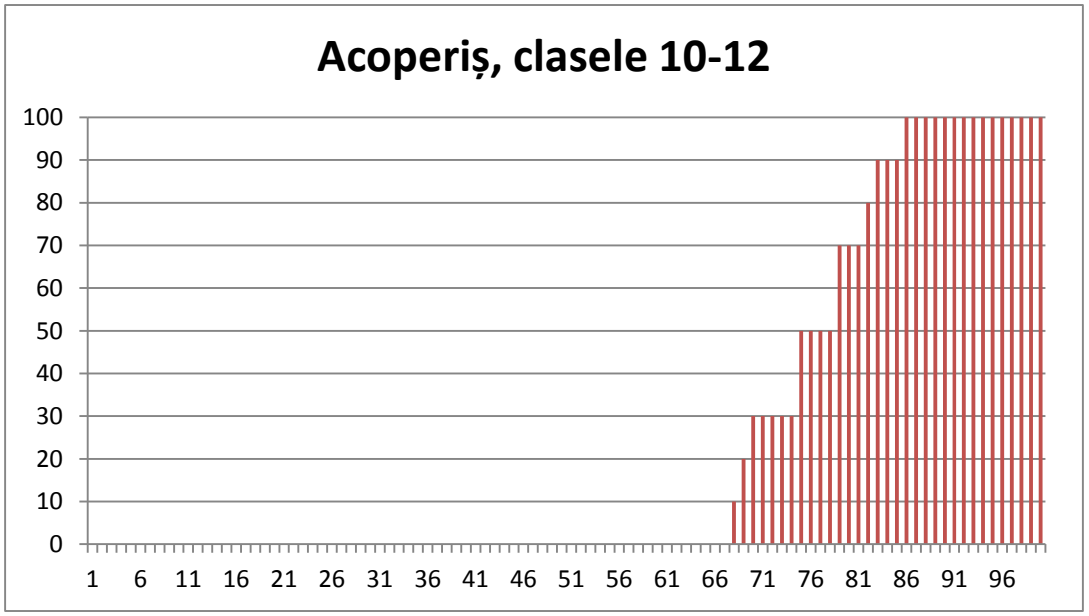
Rezolvare

Rezolvarea problemei necesită competențe de a lucra cu tipul de date tablou (unudimensionale și bidimensionale). Numărul de blocuri de lungimea `m`, care pot fi plasate pe orizontală (verticală) se vor păstra într-un vector cu `m` elemente (`blok` și `blok1`, respectiv), adică `bloc[i]` va reține numărul de blocuri de lungimea `i`, care pot fi plasate pe orizontală, iar `bloc1[i]` va reține numărul de blocuri de lungimea `i`, care pot fi plasate pe verticală. Acest lucru este necesar, deoarece trebuie să se țină cont de observația: dacă numărul de blocuri pe verticală și pe orizontală sunt egale, se va afișa varianta pe verticală. Odată cu incrementarea numărului de blocuri de fiecare lungime, se va introduce câte un contor (`orizontal`, `vertical`), care va număra numărul total de blocuri care pot fi plasate sau numai vertical sau numai orizontal pe acoperiș.

```
Program p1;
{clasele 10-12}

type vector=array[1..200] of longint;
     matrice=array[1..200,1..200] of byte;

var a:matrice;
    blok,blok1:vector;
    i,j,k,m:byte;
    orizontal,vertical:integer;
    f:text;

Begin
 Assign (f,'ACOPERIS.IN');
 reset(f);
 readln(f,m);

 for i:=1 to m do
  for j:=1 to m do
   read(f,a[i,j]);
 close(f);

 orizontal:=0;
 for i:=1 to m do
  begin
   j:=1;
   while j<=m do
    begin
     k:=0;
     while (a[i,j]<>0) and (j<=m) do
      begin
       inc(k);inc(j);
      end;
     if k<>0 then begin inc(blok[k]);inc(orizontal);end;
     inc(j);
    end;
  end;

 vertical:=0;
 for j:=1 to m do
  begin
   i:=1;
   while i<=m do
    begin
     k:=0;
     while (a[i,j]<>0) and (i<=m) do
      begin
```

```
       inc(k);inc(i);
      end;
     if k<>0 then begin inc(blok1[k]);inc(vertical);end;
     inc(i);
    end;
  end;

 assign(f,'ACOPERIS.OUT');
 rewrite(f);
 if vertical>orizontal then
  begin
   writeln(f,orizontal);
   for i:=1 to m do
    if blok[i]<>0 then writeln(f,i,' ',blok[i]);
  end                   else
  begin
   writeln(f,vertical);
   for i:=1 to m do
    if blok1[i]<>0 then writeln(f,i,' ',blok1[i]);
  end;

 close(f);

End.
```

Punctajul acumulat de fiecare competitor



Notă: Pe orizontală sînt competitorii, simbolizaţi prin numerele 1, 2, 3, ... ş.a.m.d., iar pe verticală punctajul fiecăruia din ei

Mulțimi proporţionale

Fie două șiruri de numere naturale fiecare din care este urmat de numărul 13 ce nu aparține șirului: $x_1, x_2, \ldots, x_n, 13, y_1, y_2, \ldots, y_m, 13$.

Sarcină. Alcătuiţi un program, care ar determina dacă cele două şiruri formează două mulțimi de numere direct proporționale, invers proporționale, neproporționale sau între ele nu se poate stabili proporţionalitatea. A stabili proporţionalitatea înseamnă a decide, dacă cu toate numerele din cele două şiruri se pot forma perechi (o pereche fiind formata dintr-un element al primului şir şi un element al celui de-al doilea şir), astfel încât rapoartele (pentru proporţionalitatea directă) sau produsele (pentru proporţionalitatea inversă) elementelor ce formează perechile sa fie egale. Evident, dacă numărul de elemente în cele două şiruri este diferit, nu se poate stabili proporţionalitatea.

Date de intrare. Fişierul text **`MULTIMI.IN`** conţine numere întregi separate prin spaţiu: mai întâi elementele primului şir urmat de numărul 13, iar apoi elementele celui de-al doilea şir urmat de numărul 13. Şirurile nu conţin nici un număr 13.

Date de ieşire: Fişierul text **`MULTIMI.OUT`** va conţine una din liniile:

```
Direct proportionale. Raportul R
Invers proportionale. Produsul R
Neproportionale
Nu se poate stabili proportionalitatea
```

Mărimea *R* are următoarea semnificaţie:

- valoarea raportului – pentru cazul de proporţionalitate directă;
- valoarea produsului – pentru cazul de proporţionalitate inversă.

În cazul în care şirurile nu sunt nici direct şi nici invers proporționale, în linie se va afişa cuvântul `Neproportionale`, iar dacă proporţionalitatea nu poate fi stabilită, se va afişa fraza `Nu se poate stabili proportionalitatea`.

Restricţii:

- Valoarea maximala a fiecărui element al şirurilor este 4294967295. Lungimea maximală a unui şir este 1000
- Timpul de execuţie nu va depăşi 1 secundă
- Programul va folosi cel mult 32 MB de memorie operativă
- Valoarea lui *R* se tipăreşte în format ştiinţific: `c.cEsccc`, unde `c` este cifră, iar `s` este semnul (+ sau -).

Fişierul sursă va avea denumirea **`MULTIMI.PAS`**, **`MULTIMI.C`** sau **`MULTIMI.CPP`**.

Exemplul 1

MULTIMI.IN	**MULTIMI.OUT**
`12 80 40 20 4 13 100 25 15 5 50 13`	`Direct proportionale. Raportul 8.0E-001`

Exemplul 2

MULTIMI.IN	**MULTIMI.OUT**
`2 8 4 20 4 13 10 250 5 50 13`	`Nu se poate stabili proportionalitatea`

Exemplul 3

MULTIMI.IN	**MULTIMI.OUT**
`16 17 18 19 20 1 2 3 4 5 6 7 8 9 10 11 12 14 15 13 34 35 36 37 38 39 20 21 22 23 24 25 26 17 28 29 30 32 33 13`	`Neproportionalitate`

Exemplul 4

MULTIMI.IN	MULTIMI.OUT
16 2 4 8 13 4 2 1 8 13	Direct proportionale. Raportul 2.0E+000 Invers proportionale. Produsul 1.6E+001

Rezolvare

A stabili proporţionalitatea înseamnă a decide dacă se pot forma cu toate numerele din cele două şiruri perechi (o pereche fiind formată dintr-un element al primului şir şi un element al celui de-al doilea şir) astfel încât rapoartele (produsele-pentru proporţionalitatea indirectă) elementelor ce formează perechile să fie egale. Evident, dacă nu există un acelaşi număr de elemente, nu se poate vorbi despre proporţionalitate. O metodă ar fi imperecherea în toate modurile posibile elementele celor două mulţimi şi să verificăm dacă toate perechile au fie raportul, fie produsul conctant. Însă am avea practic de generat toate permutările unei mulţimi pentru a forma perechi cu elementele celeilalte mulţimi, ceea ce înseamnă un **algoritm exponenţial**. Vom propune un alt algoritm, obţinut în baza observării unor proprietăţi matematice ale numerelor care formează un şir de rapoarte egale. Aceste observaţii sunt:

- Dacă numitorii sunt în ordine crescătoare, atunci şi numărătorii sunt tot în ordine crescătoare;
- Pentru şir de produse egale, dacă elementele din prima mulţime sunt în ordine crescătoare, atunci cele din a doua mulţime sunt în ordine descrescătoare.

Astfel ordonăm elementele fiecărei mulţimi şi formăm perechi de elemente corespunzătoare pentru a verifica proporţionalitatea directă şi perechi de elemente simetric corespunzătoare (primul cu ultimul, al doilea cu penultimul etc.) pentru a verifica proporţionalitate indirectă.

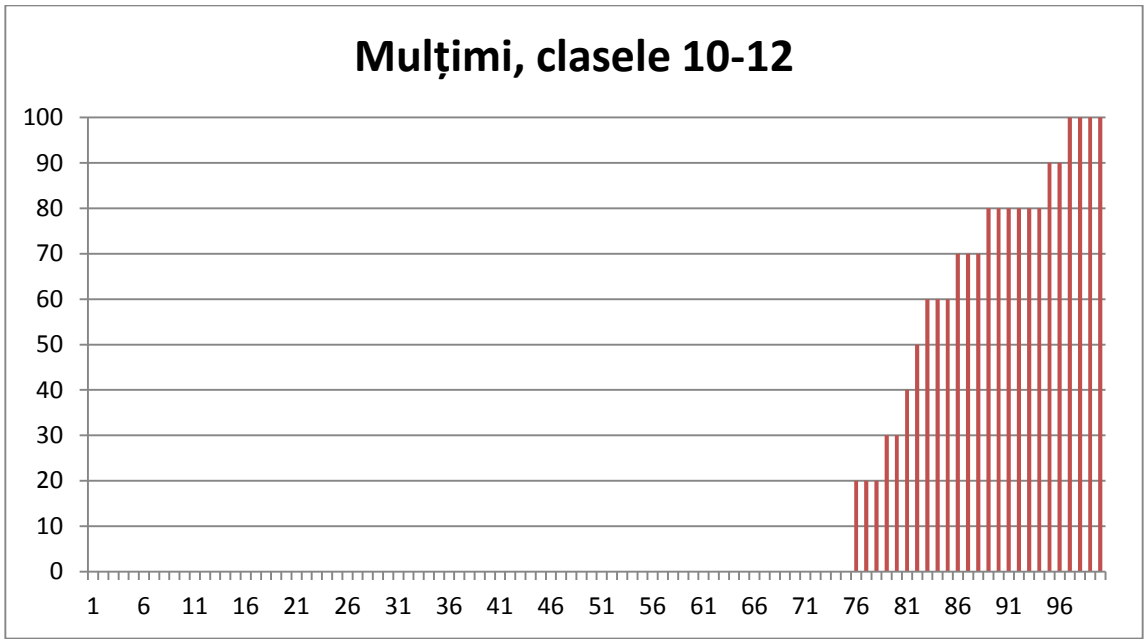
Considerăm următoarele două şiruri *{12, 80, 40, 20, 4}* si *{100 25 15 5 50}*. Atunci vom avea:

- ✓ Şirurile ordonate în ordine crescătoare: *{4, 12, 20, 40, 80 }, {5, 15, 25, 50, 100}*.
- ✓ Verificarea proportionalităţii perechilor de elemente corespunzătoare: *4/5=12/15=20/25=40/25=80/50*.

```
program multimi_proportionale;
{clasele 10-12}

uses crt;
var a,b:array[1..1000] of longword;
    na,nb,i,j:word;
    aux:longword;
    rap,rap1,prod,prod1:real;
    gata,dir_prop,inv_prop:boolean;
    fin,fout:text;

begin
 assign(fin,'MULTIMI.IN');
 reset(fin);
 assign(fout,'MULTIMI.OUT');
 rewrite(fout);

 while not seekeof(fin) do
 begin
    na:=0;
    read(fin,aux);
    while (aux<>13)do
           begin
               //writeln('citit caracter: ', aux);
               inc(na);
               a[na]:=aux;
               read(fin,aux)
           end;
    nb:=0;
```

```
    read(fin,aux);
    while (aux<>13) do
           begin
               //writeln('citit caracter: ', aux);
               inc(nb);
               b[nb]:=aux;
               read(fin,aux)
           end;
    if na<>nb then
         writeln(fout,'Nu se poate stabili proportionalitatea')
         else
          begin {ordonarea elementelor multimii a}
      i:=0;
     repeat
        i:=i+1; {a cita parcurgere se executa}
        gata:=true;
        for j:=1 to na-i do { parcurgerea sirului}
          if a[j]>a[j+1] then
           begin {interschimbarea elementelor}
             aux:=a[j];
             a[j]:=a[j+1];
             a[j+1]:=aux;
             gata:=false;
           end;
     until gata;
{.. la fel ordonarea elementelor multimii b}
  i:=0;
      repeat
         i:=i+1; {a cita parcurgere se executa}
         gata:=true;
         for j:=1 to nb-i do { parcurgerea sirului}
           if b[j]>b[j+1] then
             begin {interschimbarea elementelor}
               aux:=b[j];
               b[j]:=b[j+1];
               b[j+1]:=aux;
               gata:=false;
             end;
      until gata;

  rap:=a[1]/b[1];
  prod:=a[1]*b[nb];
  dir_prop:=true;
  inv_prop:=true;
  for i:=2 to nb do
        begin
           rap1:=a[i]/b[i];
            prod1:=a[i]*b[nb-i+1];
           if rap1 <> rap then dir_prop:=false;
           if prod1<>prod then inv_prop:=false;
        end;
  if dir_prop then
      writeln (fout,'Direct proportionale. Raportul ', rap:8);
  if inv_prop then
      writeln (fout,'Invers proportionale. Produsul ', prod:8);
  if not dir_prop and not inv_prop then
      writeln (fout,'Neproportionalitate');
   end;
 end;
 close(fin);
 close(fout);
 readkey
end.
```

Punctajul acumulat de fiecare competitor



Notă: Pe orizontală sînt competitorii, simbolizaţi prin numerele 1, 2, 3, ... ş.a.m.d., iar pe verticală punctajul fiecăruia din ei

Relaţii de simpatie

Fie dată o mulţime de persoane numerotate cu numere întregi de la **1** la ***N.*** Relația SIMPATIE este definită în felul următor: pentru fiecare persoană ***i*** sunt specificate numerele persoanelor pe care aceasta le simpatizează.

Sarcină. Alcătuiţi un program care ar determina, dacă pentru mulţimea dată de persoane toate simpatiile sunt reciproce.

Date de intrare. Fişierul text **SIMPATIE.IN** conţine un sir de numere, separate prin spaţiu, reprezentând grupuri de numere. Fiecare grup începe din linie nouă şi conţine: pe prima poziţie – numărul ce reprezintă persoana care simpatizează persoanele din mulţime; urmează numerele persoanelor simpatizate; sfârşitul grupului de numere este specificat de cifra 0.

Date de ieşire: Fişierul text **SIMPATIE.OUT** va conţine o linie, în care se va afişa:

```
N=v. SIMPATIILE SUNT RECIPROCE.
```

sau

```
N=v. SIMPATIILE NU SUNT RECIPROCE.
```

Aici v reprezintă numărul de persoane în mulţime.

Restricţii:

- $N \leq 100$
- Timpul de execuţie nu va depăşi 1 secundă
- Programul va folosi cel mult 32 MB de memorie operativă.

Fişierul sursă va avea denumirea **SIMPATIE.PAS, SIMPATIE.C** sau **SIMPATIE.CPP.**

Exemplul 1

SIMPATIE.IN	**SIMPATIE.OUT**
1 2 4 0 2 1 0 3 2 4 0	N=4. SIMPATIILE NU SUNT RECIPROCE.

Exemplul 2

SIMPATIE.IN	**SIMPATIE.OUT**
1 2 3 0 2 1 3 0 3 2 1 0	N=3. SIMPATIILE SUNT RECIPROCE.

Exemplul 3

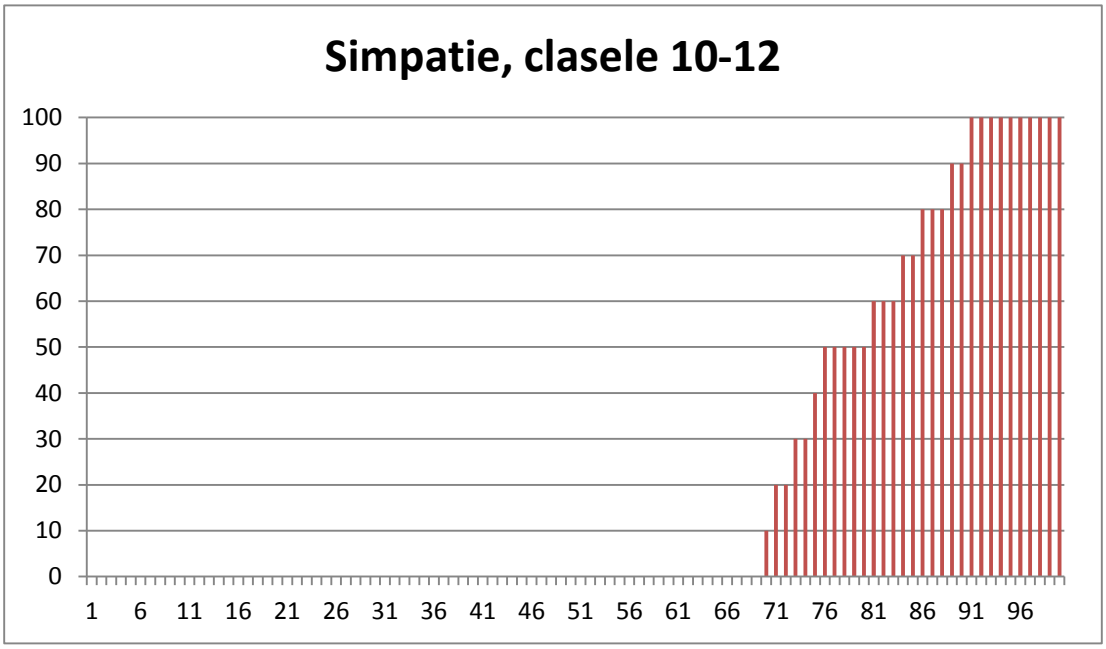
SIMPATIE.IN	**SIMPATIE.OUT**
1 2 3 4 5 6 7 0 5 1 2 3 4 6 7 0 6 1 2 3 4 5 7 0 7 1 2 3 4 5 6 0 2 1 3 4 5 6 7 0 3 1 2 4 5 6 7 0 4 1 2 3 5 6 7 0	N=7. SIMPATIILE SUNT RECIPROCE.

Rezolvare

Datele de intrare se reprezintă în forma unor liste de adiacenţă referitoare la un graf. Se constuieşte matricea de adiacenţă a grafului şi se cercetează proprietatea de graf neorientat în care matricea de adiacenţă este simetrică faţă de diagonala principală *($a_{ij}=a_{ji}$ pentru orice $1<=i,j=>n$).*

```
program relatii_simpatie;
{clasele 10-12}

uses math;
type matr_adiac=array[1..100,1..100] of byte;

var a: matr_adiac;
     n:byte;
     persoane:set of byte;
     i,j,k:integer;
     dot:char;
     este:boolean;
     fin,fout:text;
     nrPersoane: byte;
     label fine;

begin
 assign(fin,'SIMPATIE.IN');
 reset(fin);
 assign(fout,'SIMPATIE.OUT');
 rewrite(fout);

 {se determina valoarea n}
{ writeln('n=',n);}
  fillchar(a,sizeof(a),0); {zerografierea tabloului}
 n:= 0;
 persoane:= [];
 while not eof(fin) do
   begin
    read(fin,i,j);
     persoane := persoane + [i, j];
     n:=max(n, max(i, j));
     repeat
            a[i,j]:=1;
            read(fin,j);
             n:=max(n, j);
             if j <> 0 then persoane := persoane + [j];
      until j=0;
   end;
nrPersoane := 0;
for i:=1 to n do
  if i in persoane then inc(nrPersoane);

este:=true;
for i:=1 to n-1 do
        for j:=i+1 to n do
          if a[i,j]<>a[j,i] then
          begin
          este:=false;
           writeln(fout,'N=',n,'. SIMPATIILE NU SUNT RECIPROCE.');
           goto fine;
          end;
if este then writeln(fout, 'N=',n,'. SIMPATIILE SUNT RECIPROCE.');
fine:
 close(fin);
 close(fout);
readkey

end.
```

Punctajul acumulat de fiecare competitor



Notă: Pe orizontală sînt competitorii, simbolizaţi prin numerele 1, 2, 3, ... ş.a.m.d., iar pe verticală punctajul fiecăruia din ei

Acoperire

Dintr-un dreptunghi cu laturile *m* şi *n* sunt tăiate două pătrate cu latura 1 şi coordonatele (i_1, j_1) şi (i_2, j_2). Coordonate ale pătratului se consideră coordonatele colțului din dreapta-sus al lui.

Sarcină. Alcătuiţi un program, care ar determina dacă figura obţinută poate fi acoperită cu dreptunghiuri 1x2 fără suprapunerea lor.

Date de intrare. Fişierul text de intrare **ACOPERIRE.IN** conţine pe prima linie 2 numere întregi *m* și *n* – dimensiunile laturilor dreptunghiului, separate prin spațiu, iar pe a doua linie – 4 numere întregi - coordonatele (i_1, j_1) și (i_2, j_2) ale pătratelor unitare ce au fost tăiate din dreptunghiul inițial, de asemenea separate prin spațiu.

Date de ieşire. Fişierul **ACOPERIRE.OUT** va conţine pe prima linie cuvântul `Yes`, dacă acoperirea figurii este posibilă, sau cuvântul `No`, în caz contrar.

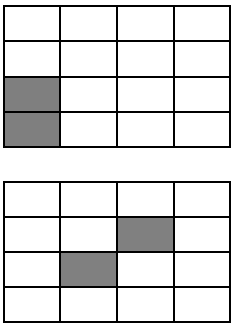
Restricţii:

- $1 \le i_1, j_1, i_2, j_2 \le m, n \le 100000$
- Timpul de execuţie nu va depăşi 1 secundă
- Programul va folosi cel mult 16 MB de memorie operativă.

Fişierul sursă va avea denumirea **ACOPERIRE.PAS, ACOPERIRE.C** sau **ACOPERIRE.CPP.**

Exemplul

ACOPERIRE.IN	ACOPERIRE.OUT
2 4 4 1 1 1 2 4 4 2 2 3 3	YES No

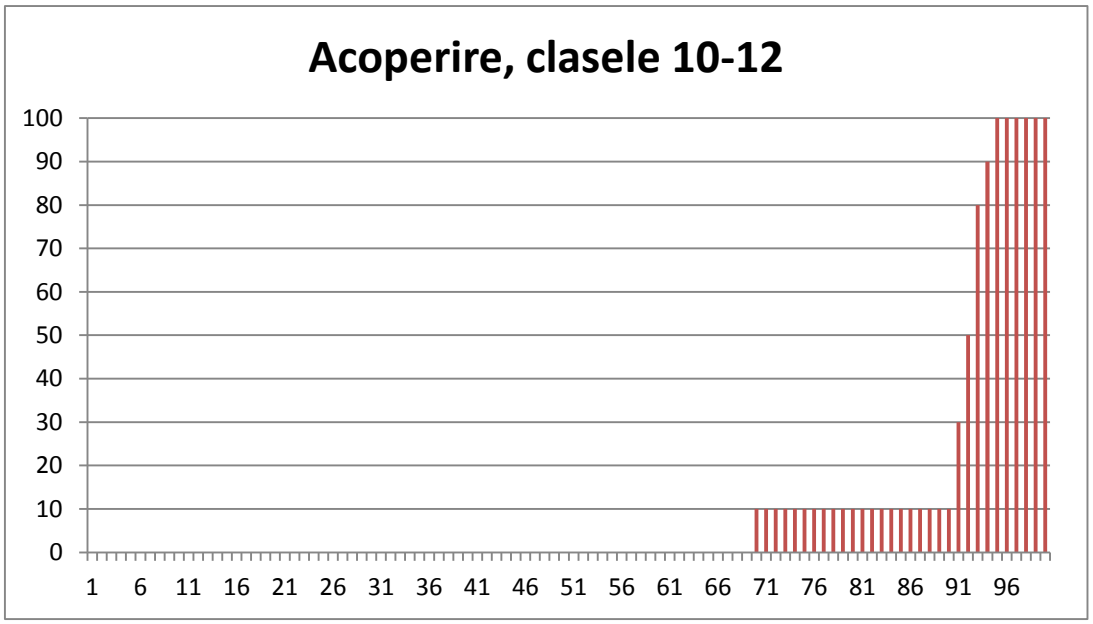


Soluţia

```
{clasele 10-12}

program acoperire;
var m, n, i1, j1, i2, j2, min: integer;
    num_tests: integer;
    FIN, FOUT: TEXT;
begin
        assign(FIN, 'ACOPERIRE.IN');
        reset(FIN);
        assign(FOUT, 'ACOPERIRE.OUT');
        rewrite(FOUT);

        readln(FIN, num_tests);

        while num_tests > 0 do
        begin
                readln(FIN, m, n, i1, j1, i2, j2);
                if (m mod 2 = 1) and (n mod 2 = 1) then
                        writeln(FOUT, 'NO')
```

```
                else if (m = 1) then begin
                  min := j1;
                     if min > j2 then min := j2;
                     if min mod 2 = 0 then writeln (FOUT, 'NO')
                     else writeln(FOUT, 'YES');
                end
                else if (n = 1) then begin
                  min := i1;
                     if min > i2 then min := i2;
                     if min mod 2 = 0 then writeln (FOUT, 'NO')
                     else writeln(FOUT, 'YES');
                end
                else if (i1 + j1) mod 2 = (i2 + j2) mod 2 then
                        writeln(FOUT, 'NO')
                else
                        writeln(FOUT, 'YES');
                dec(num_tests);
        end;

        close(FOUT);
        close(FIN);
end.
```

Punctajul acumulat de fiecare competitor



Notă: Pe orizontală sînt competitorii, simbolizaţi prin numerele 1, 2, 3, ... ş.a.m.d., iar pe verticală punctajul fiecăruia din ei

Drepte în plan

Într-un plan sunt definite n drepte $a_i x+b_i y+c_i = 0$, $i = 1, 2, \ldots, n$.

Sarcină. Alcătuiţi un program, care ar determina în câte părţi este împărţit planul de aceste drepte.

Date de intrare. Fişierul text de intrare **`DREPTE.IN`** conţine pe prima linie un număr întreg n – numărul de linii în plan, iar pe fiecare din următoarele n linii – cîte 3 numere întregi a_i, b_i şi c_i.

Date de ieşire. Fişierul text de ieşire **`DREPTE.OUT`** va conţine pe prima linie un număr natural, egal cu numărul părţilor în care a fost divizat planul de cele n drepte.

Restricţii:

- $n, |a|, |b|, |c| \le 1000$
- Timpul de execuţie nu va depăşi 1 secundă
- Programul va folosi cel mult 16 MB de memorie operativă.

Fişierul sursă va avea denumirea **`DREPTE.PAS`**, **`DREPTE.C`** sau **`DREPTE.CPP`**.

Exemplul 1

DREPTE.IN	**DREPTE.OUT**
2 1 0 0 0 1 -1	4

Exemplul 2

DREPTE.IN	**DREPTE.OUT**
3 1 1 0 1 1 2 2 2 3	4

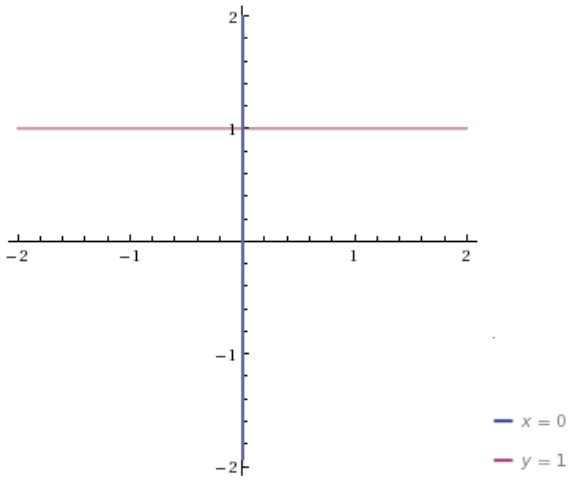


Fig. 1. La exemplul 1

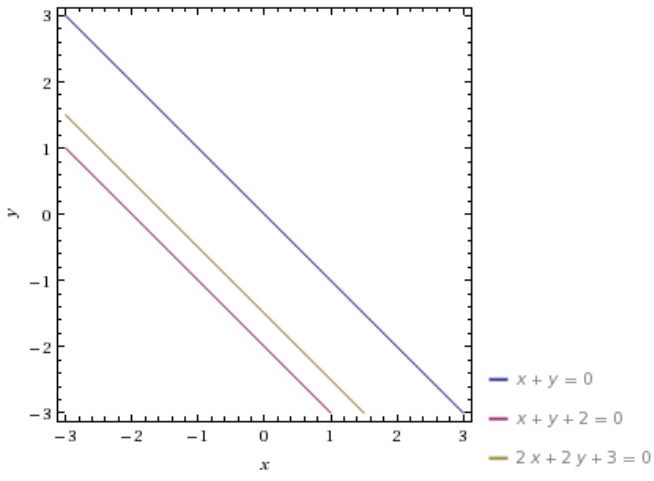


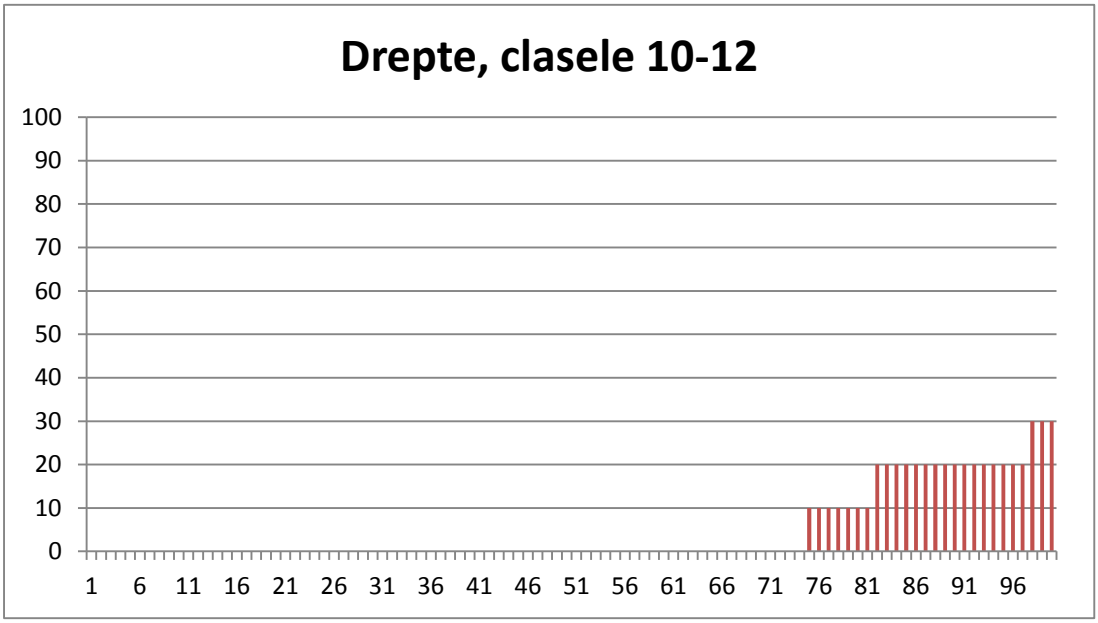
Fig. 2. La exemplul 2

Soluţia

```
{clasele 10-12}

#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include <cmath>

enum LREL {
    PARALLEL = 0, MATCH, INTERSECT
};
const double EPS = 1e-5;

struct point{
    double x, y;
    point(double px = 0, double py = 0): x(px), y(py) {}
    point(const point& p): x(p.x), y(p.y) {}
};

bool operator == (const point& p1, const point& p2){
    return fabs(p1.x - p2.x) < EPS && fabs(p1.y - p2.y) < EPS;
}

std::ostream& operator <<(std::ostream& out, const point& p){
    out << "( " << p.x << ", " << p.y << ")";
    return out;
}

struct line {
    double a, b, c;

    line(double pa = 0, double pb = 0, double pc = 0)
        : a(pa), b(pb), c(pc) {
    }

    line(const line& p)
        : a(p.a), b(p.b), c(p.c) {
    }
};

LREL linesRelations(const line& l1, const line& l2) {
    if (fabs(l1.a * l2.b - l1.b * l2.a) < EPS && fabs(l1.a * l2.c - l1.c * l2.a)
< EPS)
        return MATCH;
    if (fabs(l1.a * l2.b - l1.b * l2.a) < EPS)
        return PARALLEL;
    return INTERSECT;
}

struct cmp_line {
    line mem;

    cmp_line(const line& p) : mem(p) {
    }

    bool operator()(const line& p) {
        return linesRelations(mem, p) == MATCH;
    }
};
```

```
std::istream& operator>>(std::istream& in, line& l) {
    in >> l.a >> l.b >> l.c;
    return in;
}
```

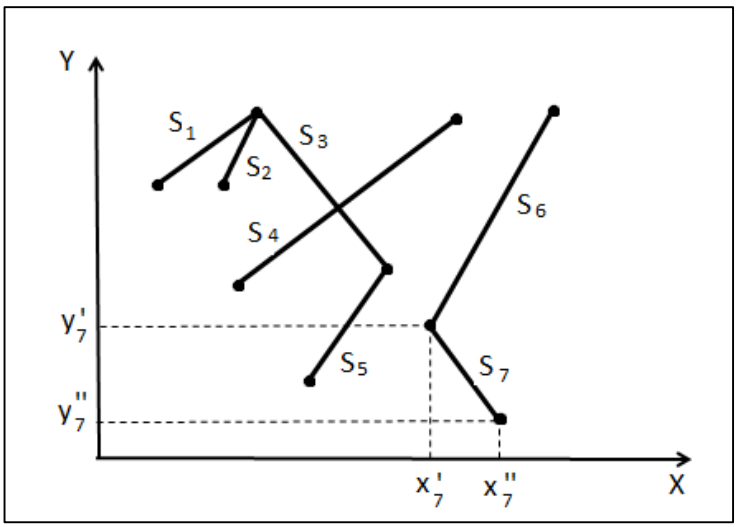
Punctajul acumulat de fiecare competitor



Notă: Pe orizontală sînt competitorii, simbolizaţi prin numerele 1, 2, 3, ... ş.a.m.d., iar pe verticală punctajul fiecăruia din ei

Segmente

Se consideră mulţimea S, formată din n segmente distincte, notate prin S_1, S_2, …, S_i, …, S_n. Fiecare segment S_i este definit prin coordonatele întregi carteziene $(x_i', y_i'), (x_i'', y_i'')$ ale extremităţilor sale.



Segmentele S_i, S_j se numesc *adiacente*, dacă ele au o extremitate comună.

Segmentele S_i, S_j se numesc *conectate* dacă există o succesiune de segmente, care începe cu S_i şi se termină cu S_j şi în care oricare două segmente consecutive sunt adiacente.

De exemplu, segmentele S_1, S_5 de pe figura alăturată sunt conectate întrucât există o succesiune de segmente consecutiv adiacente: S_1, S_3, S_5.

Submulţimea de segmente **Q**, $\boldsymbol{Q \subseteq S}$, formează o *reţea*, dacă segmentele respective sunt conectate între ele, fără a avea însă conexiuni cu segmentele din submulţimea $\boldsymbol{S \setminus Q}$.

În general, mulţimea de segmente ***S*** poate avea mai multe reţele.

De exemplu, mulţimea de segmente de pe figura de mai sus conţine 3 reţele: {S_1, S_2, S_3, S_5}, {S_4} şi {S_6, S_7}.

Sarcină. Elaboraţi un program, care, cunoscând mulţimea de segmente ***S***, calculează numărul de reţele ***k***.

Date de intrare. Fişierul text **`SEGMENT.IN`** conţine pe prima linie numărul întreg *n*. Fiecare din următoarele *n* linii ale fişierului de intrare conţine numerele reale x_i', y_i', x_i'', y_i'', separate prin spaţiu. Linia i+1 a fişierului de intrare conţine coordonatele extremităţilor segmentului S_i.

Date de ieşire. Fişierul text **`SEGMENT.OUT`** va conţine pe singura linie numărul întreg *k*.

Restricţii:

- $1 \le n \le 300000$; $-10000 \le x_i', y_i', x_i'', y_i'' \le 10000$
- Timpul de execuţie nu va depăşi 1,0 secunde
- Programul va folosi cel mult 32 MB de memorie operativă.

Fişierul sursă va avea denumirea **`SEGMENT.PAS`**, **`SEGMENT.C`** `sau` **`SEGMENT.CPP`**.

Exemplu

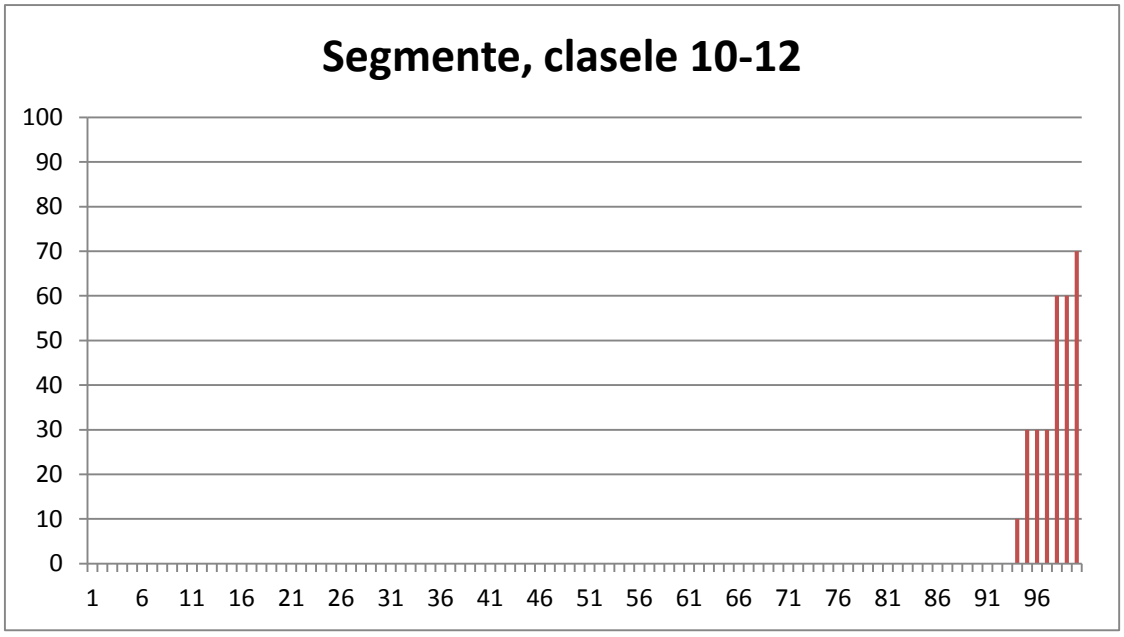
SEGMENT.IN	**SEGMENT.OUT**
7 1 7 4 8 2 6 4 8 4 8 7 4 2 4 9 7 5 2 7 4 8 3 11 8 8 3 13 1	3

```
Program segmentele;
{clasele 10-12}

const MAXN = 300000;
type TPunct = record
                x, y: longint;
                segment: longint;
              end;

    TPuncte = array[1..2 * MAXN] of TPunct;

var capete: TPuncte;
    n: longint;
    buf: array [1..20000] of byte;

procedure citeste;
var f: text;
    x1, y1, x2, y2: double;
    i: longint;
begin
  assign(f, 'SEGMENT.IN');
  reset(f);
  SetTextBuf(f, buf);
  readln(f, n);
  for i := 1 to n do
  begin
    readln(f, x1, y1, x2, y2);
    capete[2 * i - 1].x := round(x1 * 1000);
    capete[2 * i - 1].y := round(y1 * 1000);
    capete[2 * i - 1].segment := i;

    capete[2 * i].x := round(x2 * 1000);
    capete[2 * i].y := round(y2 * 1000);
    capete[2 * i].segment := i;
  end;
  close(f);
end;

function comparaPuncte(var a, b:TPunct): integer;
begin
  if a.x < b.x then exit(-1);
  if a.x > b.x then exit(1);

  {a.x = b.x}
  if a.y < b.y then exit(-1);
  if a.y > b.y then exit(1);
  exit(0);
end;

procedure qsort(var data: TPuncte; a, b: longint);
var left, right: longint;
    aux, pivot : TPunct;
begin
  pivot := data[(a + b) div 2];
  left := a;
  right := b;

  while left <= right do
  begin
    while comparaPuncte(data[left], pivot) < 0 do
      left := left + 1;  { Parting for left }
    while comparaPuncte(data[right], pivot) > 0 do
      right := right - 1;{ Parting for right}
    if left <= right then   { Validate the change }
```

```
    begin
      //swap Data[left] with Data[right];
       aux := data[left];
       data[left] := data[right];
       data[right] := aux;
       left := left + 1;
       right:= right - 1;
    end;
  end;
  if right > a then qsort(data, a, right); { Sort the LEFT  part }
  if b > left  then qsort(data, left, b); { Sort the RIGHT part }
end;

var i: longint;
type TNod = record
               tata: longint;
               sz: longint;
             end;

var retele: array[1.. 2 * MAXN] of TNod;

function aflaComponenta(a: longint): longint;
var pozitie, z, nextPos: longint;
begin
  z := 1;
  pozitie := a;
  while retele[pozitie].tata > 0 do
   pozitie := retele[pozitie].tata;
  {id-ul retelei e pos}
  z := pozitie;
  pozitie := a;
  while pozitie <> z do
   begin
     nextPos := retele[pozitie].tata;
     retele[pozitie].tata := z;
     pozitie := nextPos;
   end;
   aflaComponenta := z;
end;

procedure uneste(a, b: longint);
var componentaA, componentaB: longint;
begin
   componentaA := aflaComponenta(a);
   componentaB := aflaComponenta(b);
   if componentaA = componentaB then exit; {nu e nimic de facut, sunt deja
unite}
   if retele[componentaA].sz < retele[componentaB].sz then
    begin
      retele[componentaA].tata := componentaB;
      inc(retele[componentaB].sz, retele[componentaA].sz);
    end
    else
    begin
      retele[componentaB].tata := componentaA;
      inc(retele[componentaA].sz, retele[componentaB].sz);
    end;
end;

var capDeRetea: array[1..MAXN] of boolean;
     rezultat, componenta: longint;
     f: text;

begin
```

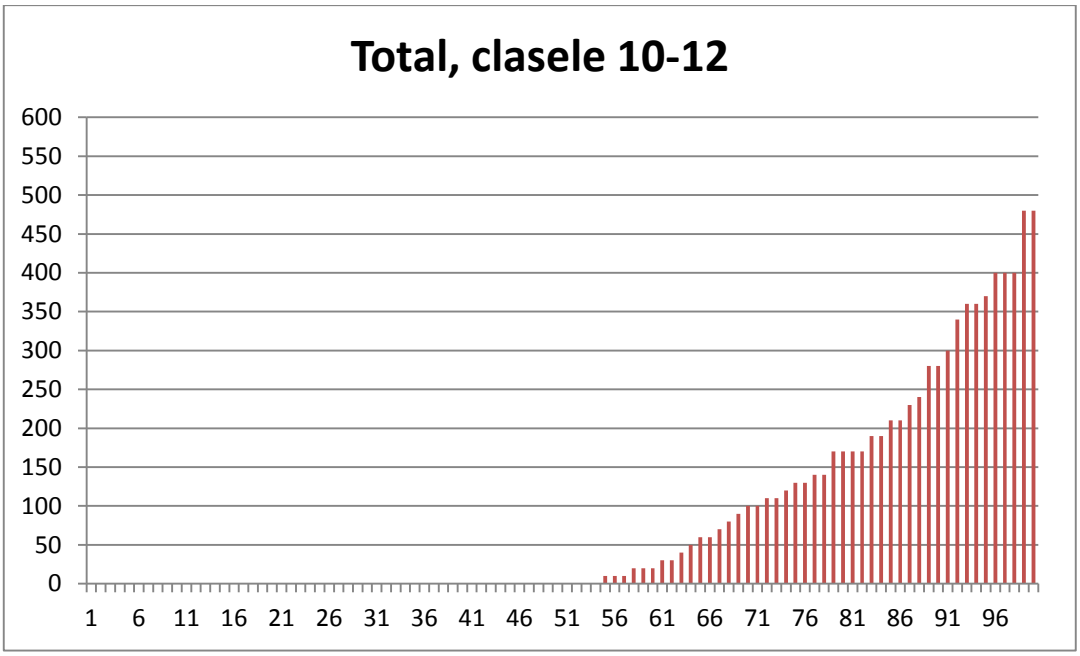
```
  citeste;
  qsort(capete, 1, 2 * n);
  for i := 1 to 2 * n do
   begin
     retele[i].tata := -1;
     retele[i].sz := 1;
   end;

  for i := 2 to 2 * n do
  begin
    if comparaPuncte(capete[i - 1], capete[i]) = 0 then
       uneste(capete[i-1].segment, capete[i].segment);
  end;

  fillchar(capDeRetea, sizeof(capDeRetea), false);
  for i := 1 to n do
   begin
     componenta := aflaComponenta(i);
     capDeRetea[componenta] := true;
   end;

  rezultat := 0;
  for i := 1 to n do
   if capDeRetea[i] then
    inc(rezultat);

  assign(f, 'SEGMENT.OUT');
  rewrite(f);
  writeln(f, rezultat);
  close(f);
  {for i := 1 to 2 * n do
    writeln(capete[i].x, ' ', capete[i].y, ' ', capete[i].segment);}
end.
```

Punctajul acumulat de fiecare competitor



Notă: Pe orizontală sînt competitorii, simbolizaţi prin numerele 1, 2, 3, ... ş.a.m.d., iar pe verticală punctajul fiecăruia din ei

Punctajul total acumulat de fiecare competitor



Notă: Pe orizontală sînt competitorii, simbolizaţi prin numerele 1, 2, 3, ... ş.a.m.d., iar pe verticală punctajul fiecăruia din ei

Lista premianţilor Olimpiadei republicane de Informatică din anul 2015

Nr. crt.	Numele, prenumele elevului	Clasa	Instituţia, localitatea, raionul/municipiul	Profesor	Locul
1.	Cojocaru Gabriel	8	LT "Orizont", Durleşti, mun. Chişinău	Corlat Sergiu	1
2.	Bezdrighin Marcel	11	LT "Orizont", Durleşti, mun. Chişinău	Corlat Sergiu	1
3.	Ţarigradschi Mihail	11	LT "Orizont", Durleşti, mun. Chişinău	Corlat Sergiu	1
4.	Griza Daniel	12	LT "Orizont", Durleşti, mun. Chişinău	Corlat Sergiu	1
5.	Umanschii Ianic	9	LT "M.Eminescu", or. Drochia	Chistruga Gheorghe	2
6.	Moglan Mihai	8	LT "Iulia Haşdeu", mun.Chişinău	Ţurcanu Ludmila	2
7.	Zatîc Petru	10	Liceul Academiei de Ştiinţe, mun. Chişinău	Miron Raisa	2
8.	Russu Vadim	an. II (11)	Colegiul de Informatică, mun. Chişinău	Iordăchiţă Elena	2
9.	Gorpinevici Vlad	11	LT "Orizont", Durleşti, mun. Chişinău	Corlat Sergiu	2
10.	Morgun Vadim	11	Liceul Teoretic nr. 1 din Tiraspol	Şagoian Tamara	2
11.	Trifan Tamara	12	Liceul Academiei de Ştiinţe, mun. Chişinău	Miron Raisa	2
12.	Chihai Mihai	12	Liceul Academiei de Ştiinţe, mun. Chişinău	Miron Raisa	2
13.	Cojocaru Cătălin	8	LT "M.Eminescu", or. Drochia	Chistruga Gheorghe	3
14.	Vişanu Cristian	9	LT "Ion Luca Caragiale", or. Orhei	Gurău Vitalie	3
15.	Vozian Valentin	10	Liceul Academiei de Ştiinţe, mun. Chişinău	Miron Raisa	3
16.	Ciornei Florin	10	LCI "Prometeu-Prim", mun. Chişinău	Zaim Sofia	3
17.	Iusiumbeli Vladislav	10	LT "S.Baranovski", s. Copceac, UTAG	Dragan Nicolai	3
18.	Valeanu Valentin	11	LT "B.Cazacu", or. Nisporeni	Brodicico Valeriu	3
19.	Căliman Laura	11	LT "M.Marinciuc", mun. Chişinău	Gurmeza Inga	3
20.	Grosu Daniel	11	Liceul Academiei de Ştiinţe, mun. Chişinău	Miron Raisa	3
21.	Savin Vadim	an. II (11)	Colegiul Financiar Bancar, mun. Chişinău	Bagrin Diana	3
22.	Movila Alexandru	12	LT ”M.Viteazul”, mun. Chişinău	Bușila Snejana	3
23.	Şulghin Valeriu	an. III (12)	Colegiul de Informatică, mun. Chişinău	Iordăchiţă Elena	3

24.	Motroi Valeriu	12	Liceul Academiei de Ştiinţe, mun. Chişinău	Miron Raisa	3
25.	Cepalîga Vladislav	9	Liceul Teoretic nr. 1 din Tiraspol	Şagoian Tamara	M
26.	Chiriţa Sandu	8	Liceul ORT "B.Z.Herţli", mun. Chişinău	Belomenova Aliona	M
27.	Nicolaişin-Şisciuc David	9	LT "M.Lomonosov", mun. Bălţi	Rotari Iurie	M
28.	Paşa Corneliu	8	LT "M. Eliade", mun. Chişinău	Anton Ghenadie	M
29.	Dodon Ion	10	LT "B.Cazacu", or. Nisporeni	Brodicico Valeriu	M
30.	Maşurceac Serghei	an. I (10)	Colegiul de Informatică, mun. Chişinău	Botoşanu Mihail	M
31.	Cojocari Valeriu	10	LT "Orizont", Durleşti	Corlat Sergiu	M
32.	Stariţin Denis	10	LT "P. Movila", mun. Chişinău	Vîdîş Alla	M
33.	Drumea Vasile	11	LT "B.Cazacu", or. Nisporeni	Brodicico Valeriu	M
34.	Rozimovschi Denis	11	LT "Olimp", Sîngerei	Sandu Pavel	M
35.	Mihalache Andrei	11	Liceul Academiei de Ştiinţe, mun. Chişinău	Miron Raisa	M
36.	Marusic Diana	11	LT "Ion Creangă", mun. Chişinău	Josu Larisa	M
37.	Rusu Iurie	12	LT "Orizont", Durleşti, mun. Chişinău	Corlat Sergiu	M
38.	Savva Dumitru	12	LT "Orizont", Durleşti, mun. Chişinău	Corlat Sergiu	M
39.	Bereghici Ştefan	12	Liceul Academiei de Ştiinţe, mun. Chişinău	Miron Raisa	M
40.	Pupeza Alexandru	12	LT ”N. Gogol”, mun. Chişinău	Panoceac Tatiana	M