

MINISTERUL EDUCAȚIEI
AL REPUBLICII MOLDOVA



МИНИСТЕРСТВО ПРОСВЕЩЕНИЯ
РЕСПУБЛИКИ МОЛДОВА

Olimpiada Republicană la Informatică

Ediția 2011

Chișinău 2011

Olimpiada Republicană la Informatică. Ediția 2011.

Lucrarea conține problemele propuse la Olimpiada Republicană la Informatică a elevilor, ediția 2011. Pentru fiecare problemă sunt descrise algoritmul și soluția respectivă în limbajul de programare PASCAL.

Materialele Olimpiadei pot fi de un real folos la studierea Informaticii atât elevilor, cât și profesorilor de Informatică.

La elaborarea ediției au contribuit:

Ion Bolun,	profesor universitar, Academia de Studii Economice a Moldovei
Anatol Gremalschi,	profesor universitar, Institutul de Politici Publice
Iurie Mocanu,	șef de direcție, Ministerul Educației
Viorel Afteni,	consultant superior, Ministerul Educației
Nicolai Falico,	lector superior, Universitatea Tehnică a Moldovei
Dumitru Codreanu,	doctorand, Universitatea București
Dumitru Ciubafii,	director, TenerLab SRL
Bogdănaș Denis,	asistent, Universitatea Tehnică, Iași
Constantin Dolghier,	inginer, ICS Micrologic Design Automation SRL

Cuprins

Cuvânt de salut	4
Clasele 7–9.....	6
Anagrame.....	7
Bilete	10
Hrubele de la Cricova	12
Jocul	16
Pătratul numărului binar.....	18
Turnuri.....	20
Punctajul competitorilor	23
Complexitatea problemelor	23
Clasele 10–12.....	24
Comoara.....	25
Bilete	29
Hrubele de la Cricova	31
Jocul	35
Rădăcina numărului binar	38
Turnuri.....	41
Punctajul competitorilor	44
Complexitatea problemelor	44
Lista premianților	45

Cuvânt de salut

Dragi elevi și stimați profesori,
care vă dăruieți acestui domeniu de vârf
al științei și tehnologiei moderne pe nume
Informatica,

Orice competiție este o experiență pentru participanți. Cu cât mai înalt este rating-ul competiției, cu atât mai bogată și fructuoasă este experiența obținută. Olimpiada Republicană la Informatică întrunește cei mai buni elevi în domeniu din republică și are ca scop atât încurajarea și susținerea interesului elevilor către informatică, cât și evaluarea gradului de pregătire a lor în domeniu, elucidarea lacunelor în instruire și definirea unor căi de îmbunătățire continuă a acesteia. Ea permite, de asemenea, stabilirea de noi cunoștințe între elevi și profesori, dar și reevaluarea valorilor. Rezultatele acesteia sunt folosite la selectarea candidaților pentru echipa de elevi ce vor participa la competiții internaționale în informatică.

Printre olimpiadele republicane ale elevilor, cea la Informatică are, totuși, un rol aparte, determinat de rolul deosebit al informaticii în societatea modernă. Informația a devenit, în ultimele decenii, un neofactor de producție mult mai important, deseori, decât materiile prime și energia: „Cine deține informația – stăpânește situația”. Informatica, la rândul său, facilitează considerabil valorificarea eficientă a informațiilor deja cunoscute și, de asemenea, procesele de obținere a unor noi informații.

Calculatoarele – cea mai complexă operă a rațiunii umane, fortifică considerabil, uneori greu de imaginat, capacitățile intelectuale umane. Astfel, în timpul parcurgerii de către o rază de lumină în vid a distanței de un singur milimetru, supercalculatorul Tianhe-1A al Universității NUDT (China) execută cca. 15000 instrucțiuni. Efectul aplicării unor asemenea capacități de procesare a informației este considerabil. Se rezolvă probleme care era imposibil de rezolvat, crește semnificativ productivitatea și se îmbunătățește considerabil calitatea muncii. Avantajele folosirii mijloacelor informatice au condus la implementarea lor tot mai largă. Sectorul informaticii a devenit un sector strategic de susținere a creșterii economice și de prosperare a societății. Cercetări recente arată că până la 30-50% din creșterea economică a unei țări se datorează sectorului Tehnologiilor informaționale și de telecomunicații.

Anume impactul benefic considerabil al informaticii asupra societății a condus la concluzia despre evoluția spre societatea informațională-societatea cunoașterii. Pași concreți în edificarea Societății informaționale sunt întreprinși și în Republica Moldova. De exemplu, agenții economici care activează în domeniul TI se bucură de anumite facilități fiscale. Solicitarea informaticienilor pe piața muncii este atât de înaltă că chiar și în perioade de criză economică în multe țări se simte un deficit de specialiști în domeniu.

Dacă tinerii sunt viitorul societății, atunci tinerilor informaticieni le revine rolul de avangardă în edificarea societății viitorului – societății informaționale. Ne mândrim cu acest rol deosebit de important al informaticii în societate, dar și responsabilitatea ce ne revine este pe măsură. Pentru a face față sarcinilor

respective, trebuie să ne pregătim bine din timp, orientat și sistematic. Conform raportului grupului Bangemann către Consiliul Europei: țările, care nu vor întreprinde măsuri radicale orientate la edificarea intensă a societății informaționale, vor rămâne economic în urmă pentru totdeauna. Nu trebuie să ne liniștească, în această privință, „legea vaselor comunicante” în era Internet-ului. Iar în avangardă întotdeauna se află cei mai buni. De ei depinde, în primul rând, bunăstarea zilei de mâine pentru toți. Cu alte cuvinte, mâine voi, tinerii informaticieni de azi, veți fi aceea de care va depinde substanțial prosperarea acestei palme de pământ – Republica Moldova. Sarcinile sunt diverse și fiecare din noi la fel. Astăzi unul din noi găsește o soluție originală, iar mâine – un altul și în rezultat avem de câștigat cu toții.

Lumea aparține celor capabili și energici. În primii cinci, din cei mai bogați oameni ai lumii la ora actuală, trei sunt din domeniul informaticii și telecomunicațiilor: pe primul loc este *Carlos Slim* – telecomunicații, pe al doilea – *William (Bill) Gates* (fondator al companiei *Microsoft*) – informatică și pe al cincilea - *Larry Ellison* (fondator al companiei *Oracle*) – informatică. Există și multe alte exemple demne de urmat.

Din 2007 la 17 mai se sărbătorește Ziua Mondială a Telecomunicațiilor și Societății Informaționale. Felicitări călduroase tuturor cu această sărbătoare – este sărbătoarea noastră a informaticienilor, și sperăm ca cele două zile de competiții în informatică ce urmează vă vor mobiliza la noi realizări în domeniu.

În numele Consiliului Olimpic la Informatică,

Ion Bolun

Profesor universitar, ASEM

Clasele 7–9

Denumirea problemei	Numărul de puncte alocat problemei	Denumirea fișierului sursă	Denumirea fișierului de intrare	Denumirea fișierului de ieșire
Anagrame	100	ANA.PAS ANA.C ANA.CPP	ANA.IN	ANA.OUT
Bilete	100	BILETE.PAS BILETE.C BILETE.CPP	BILETE.IN	BILETE.OUT
Hrubele de la Cricova	100	HRUBE.PAS HRUBE.C HRUBE.CPP	HRUBE.IN	HRUBE.OUT
Jocul	100	JOCUL.PAS JOCUL.C JOCUL.CPP	JOCUL.IN	JOCUL.OUT
Pătratul numărului binar	100	PATRAT.PAS PATRAT.C PATRAT.CPP	PATRAT.IN	PATRAT.OUT
Turnuri	100	TURNURI.PAS TURNURI.C TURNURI.CPP	TURNURI.IN	TURNURI.OUT
Total	600	–	–	–

Anagrame

Se consideră mulțimea șirurilor de caractere $S = \{s_1, s_2, \dots, s_n\}$. Prin definiție, mulțimea S este o *mulțime fără anagrame*, dacă în ea nu există nici o pereche $\{s_i, s_j\}$ de șiruri, unul din care poate fi obținut din celălalt prin operații de anagramare.

Amintim, că anagramarea este un procedeu literar, ce constă în schimbarea ordinii literelor într-un cuvânt. De exemplu, în antichitate, numele regelui *Ptolemaios*, prin anagramare, a fost transformat în *Apomelitos* („care provine din miere”). Cuvintele obținute prin anagramare se numesc anagrame.

Sarcină. Elaborați un program care, cunoscând mulțimea $S = \{s_1, s_2, \dots, s_n\}$, calculează numărul m de șiruri din cea mai mare submulțime fără anagrame a mulțimii S .

Date de intrare. Prima linie a fișierului text ANA.IN conține numărul întreg n . Fiecare din următoarele n linii ale fișierului de intrare conține câte un șir de caractere. Linia $i + 1$ a fișierului de intrare conține șirul s_i .

Date de ieșire. Fișierului text ANA.OUT va conține pe o singură linie numărul întreg m .

Exemplu.

ANA.IN

```
6
AnaAreMere
MereAreAna
IonAreMere
AreIonMere
SiCeDacaIonAreMere
IonAreMulteMere
```

ANA.OUT

```
4
```

Restricții. $1 \leq n \leq 10\,000$. Șirurile de caractere din mulțimea S sunt formate din literele mari și mici ale alfabetului latin. Fiecare șir conține cel mult 250 de caractere. Timpul de execuție nu va depăși 0,5 secunde. Programul va folosi cel mult 32 Megaocteți de memorie operativă. Fișierul sursă va avea denumirea ANA.PAS, ANA.C sau ANA.CPP.

Rezolvare

Mai întâi vom încerca sa rezolvăm această problemă prin metoda forței brute sau, prin alte cuvinte, prin metoda trierii.

Vom elabora în acest scop următoarele subprograme:

Function SuntAnagrame($A, B : \text{string}$) : **boolean** – funcția returnează TRUE dacă șirurile de caractere A și B sunt anagrame și FALSE în caz contrar. Valorile acestei funcții pot fi calculate, verificând dacă frecvențele de apariție a fiecărui caracter în ambele șiruri sunt egale.

Function EsteFaraAnagrame($C : \text{set of string}$) : **boolean** – funcția repunează TRUE dacă mulțimea C de șiruri de caractere este fără anagrame și FALSE în caz contrar. Valorile acestei funcții pot fi calculate formând toate perechile posibile de șiruri (A, B) , $A \in C$ și $B \in C$, apelând pentru fiecare pereche funcția EsteFaraAnagrame.

1. Un posibil algoritm, bazat pe metoda trierii, va avea următoarea structură:
2. Formăm o submulțime nevidă C a mulțimii de șiruri S .
3. Cu ajutorul apelului EsteFaraAnagrame(C) verificăm dacă submulțimea

respectivă este fără anagrame. Dacă valoarea returnată de funcția `EsteFaraAnagrame` este `TRUE`, memorăm numărul de elemente ale mulțimii C .

4. În continuare formăm o nouă submulțime C a mulțimii S ș.a.m.d. până vor fi examinate toate submulțimile posibile.

Întrucât numărul de submulțimi nevide ale mulțimii S este 2^n , complexitatea temporală a unui astfel de algoritm va fi $O(2^n)$. Evident, pentru $n \geq 20$, programul bazat pe metoda trierii nu se va încadra în timpul indicat în restricțiile problemei.

Prin urmare, competitorii care au mers pe această cale, au avut șanse să obțină puncte doar în cazul testelor cu valori mici ale lui n .

Complexitatea temporală a algoritmului poate fi însă redusă, dacă, în loc de trierea tuturor submulțimilor lui S , vom construi doar submulțimea ce conține cel mai mare număr de cuvinte fără anagrame. O astfel de mulțime poate fi construită după cum urmează:

1. Sortăm caracterele din fiecare șir al mulțimii S în ordine alfabetică. Evident, după sortare, șirurile care sunt anagrame devin egale între ele.
2. Sortăm șirurile din mulțimea S în ordine lexicografică. După sortare, în mulțimea ordonată S , șirurile egale vor ocupa poziții consecutive.
3. Excludem din S submulțimile de șiruri ce coincid, lăsând în S doar câte un șir din submulțimile excluse. Evident, excluderea propriu-zisă nici nu este necesară, este suficient să numărăm doar șirurile ce diferă.

Complexitatea temporală a unui astfel de algoritm depinde de metodele de sortare utilizate. În cazul metodei bulelor, complexitatea sortării caracterelor din fiecare șir al mulțimii S este $O(nm^2)$, unde m este numărul maximal de caractere într-un șir, iar complexitatea sortării șirurilor este $O(n^2)$. Întrucât cu creșterea lui m și n valorile $O(nm^2)$ cresc cu mult mai încet decât valorile $O(2^n)$, șansele că vor trece mai multe teste cresc. Totuși, pentru $m = 250$ și $n = 10\,000$, valoarea $nm^2 \approx 6,25 \cdot 10^8$ este prea “aproape” de viteza de procesare a calculatoarelor moderne.

Prin urmare, pentru a fi siguri că programul se va încadra în restricțiile temporale, trebuie utilizată o metodă rapidă de sortare, de exemplu, *QuickSort*, *MergeSort* sau *HeapSort*, care sunt descrise în literatură. Complexitatea unor astfel de metode de sortare este $O(n \log n)$, fapt ce ne garantează respectarea restricțiilor problemei.

În programul ce urmează, șirurile de caractere sunt stocate în memoria dinamică, iar sortarea lor se face prin metoda *QuickSort*.

```
Program Anagrame;

type TSiruri = Array[1..10000] of String;

var n : Integer;
    siruri : TSiruri;
    i, count : Integer;
    f, g : Text;

procedure Swap(var a, b : string);
var temp : string;
begin
    temp := a;
    a := b;
    b := temp;
end;

procedure Sort(var s : String);
{ sorteaza caracterele unui sir de caracter prin numarare }
```



```

var i, k : Integer;
    c : Char;
    l : integer;
    hist: Array[char] of ShortInt;
begin
    { numara caracterele }
    FillChar(hist, sizeof(hist), 0);
    for i := 1 to Length(s) do Inc(hist[s[i]]);
    l := 1;
    for c := char(0) to char(250) do
        for k := 1 to hist[c] do
            begin s[l]:=c; Inc(l); end;
end; { Sort }

function Partition(var v : TSiruri; left, right : Integer) : Integer;
{ partitionarea pentru procedeul QuickSort:
  Aduce elementele mai mici ca valoarea pivotului pe pozitiile 1..p, si
  returneaza acea valoare p }
var pv : String;
    p, i : Integer;
begin
    pv := v[right];
    p := left;
    for i := left to right - 1 do
        if v[i] < pv then begin
            Swap(v[i], v[p]);
            Inc(p);
        end;
    Swap(v[p], v[right]);
    Partition := p;
end; { Partition }

procedure QuickSort(var v : TSiruri; left, right : integer);
var pivotIndex : Integer;
begin
    if left >= right then exit;
    pivotIndex := Partition(v, left, right);
    QuickSort(v, left, pivotIndex -1);
    QuickSort(v, pivotIndex + 1, right);
end; { QuickSort }

begin
    { Citeste Datele }
    Assign(f, 'ana.in');
    Reset(f);
    Readln(f, n);
    for i := 1 to n do Readln(f, siruri[i]);
    Close(f);
    { sorteaza caracterele sirurilor }
    for i := 1 to n do Sort(Siruri[i]);
    { sorteaza lexicografic sirurile de caractere }
    QuickSort(Siruri, 1, n);
    { numara cate sunt diferite }
    count := 1;
    for i:= 2 to n do
        if Siruri[i] <> Siruri[i-1] then Inc(count);
    { Scrie raspunsul }
    Assign(g, 'ana.out');
    Rewrite(g);
    Writeln(g, count);
    Close(g);
end.

```

Bilete



Biletele din transportul public din marele oraşe sunt numerotate. De obicei, un număr de bilet este format din şase cifre zecimale.

Unii elevi şi studenţi consideră că în cazul în care suma primelor trei cifre din componenţa numărului de bilet este egală cu suma ultimelor trei cifre, biletul respectiv aduce noroc sau, prin alte cuvinte, este un bilet „fericit”.

De exemplu, biletul cu numărul 032050 este „fericit”, iar cel cu numărul 283357 – nu.

Sarcină. Elaboraţi un program care, cunoscând numărul de bilet, determină dacă biletul respectiv este „fericit”.

Date de intrare. Prima linie a fişierului text BILETE.IN conţine numărul întreg n – numărul total de bilete supuse examinării. Fiecare din următoarele n linii ale fişierului de intrare conţine câte un număr de bilet, format din şase cifre zecimale.

Date de ieşire. Prima linie a fişierului text BILETE.OUT va conţine numărul întreg n . Fiecare din următoarele n linii ale fişierului de ieşire va conţine cuvântul DA dacă biletul din linia respectivă a fişierului de intrare este „fericit” din şi NU în caz contrar.

Exemplu.

BILETE.IN	BILETE.OUT
3	3
032050	DA
283375	NU
121301	DA

Restricţii. $1 \leq n \leq 1000$. Timpul de execuţie nu va depăşi 0,5 secunde. Programul va folosi cel mult 32 Megaocteţi de memorie operativă. Fişierul sursă va avea denumirea BILETE.PAS, BILETE.C sau BILETE.CPP.

Rezolvare

Vom nota prin $c_1, c_2, c_3, c_4, c_5, c_6$ cifrele numărului de bilet. De exemplu, în cazul numărului de bilet 360710 avem $c_1 = 3, c_2 = 6, c_3 = 0, c_4 = 7, c_5 = 1, c_6 = 0$.

Cifrele numărului de bilet pot fi calculate în ordinea $c_6, c_5, c_4, c_3, c_2, c_1$ prin împărţiri succesive a numărului de bilet şi a câturilor obţinute la baza 10 a sistemului zecimal de numeraţie, reţinând resturile împărţirilor.

Evident, un bilet este “fericit” atunci când $(c_1 + c_2 + c_3) = (c_4 + c_5 + c_6)$.

```
Program Bilete;
{ Clasele 07-09 }
var Intrare, Iesire : text;
    n : longint;      { numarul de bilete in fisierul de intrare }
    Numar : longint;  { numarul de bilet }
    c1, c2, c3, c4, c5, c6 : integer; { cifrele numarului de bilet }
    i : longint;
begin
    assign(Intrare, 'BILETE.IN');
    reset(Intrare);
    assign(Iesire, 'BILETE.OUT');
    rewrite(Iesire);
```

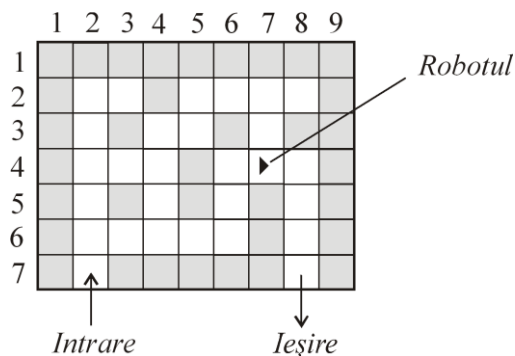
```

readln(Intrare, n);
writeln(Iesire, n);
for i:=1 to n do
    begin
        readln(Intrare, Numar);
        c6:=Numar mod 10; Numar:=Numar div 10;
        c5:=Numar mod 10; Numar:=Numar div 10;
        c4:=Numar mod 10; Numar:=Numar div 10;
        c3:=Numar mod 10; Numar:=Numar div 10;
        c2:=Numar mod 10; Numar:=Numar div 10;
        c1:=Numar mod 10;
        if (c1+c2+c3)=(c4+c5+c6) then writeln(Iesire, 'DA')
            else writeln(Iesire, 'NU');
    end; { for }
close(Intrare);
close(Iesire);
end.

```

Hrubele de la Cricova

După o vizită la renumitele hrube* de la Cricova, un informatician a construit un robot care lucrează într-un câmp divizat în pătrățele (vezi desenul). Pătrățelele hașurate reprezintă obstacolele, iar cele nehașurate – spațiile libere. Pătrățelele de pe perimetrul planului, cu excepția celor de intrare sau ieșire, sînt hașurate prin definiție.



Robotul poate executa doar instrucțiunile SUS, JOS, DREAPTA, STANGA, conform cărora el se deplasează în unul din pătrățele vecine. Dacă în pătratul vecin este un obstacol, de exemplu, un perete sau un butoi, are loc un accident și robotul iese din funcțiune.

În formă numerică planul hrubelor este redat prin tabloul A cu n linii și m coloane. Elementele $A[i, j]$ ale acestui tablou au următoarea semnificație: 0 – spațiu liber; 1 – obstacol; 2 – intrarea în hrube; 3 – ieșirea din hrube. Inițial, robotul se află în pătrățul pentru care $A[i, j] = 2$.

Sarcină. Elaborați un program, care, cunoscând planul hrubelor, deplasează robotul prin încăperile subterane, de la intrarea în hrube la ieșire. Colecția de vinuri fiind foarte bogată, nu se cere vizitarea obligatorie a tuturor încăperilor subterane.

Date de intrare. Fișierul text HRUBE.IN conține pe prima linie numerele întregi n, m separate prin spațiu. Pe fiecare din următoarele n linii se conțin câte m numere întregi $A[i, j]$, separate prin spațiu.

Date de ieșire. Fișierul text HRUBE.OUT va conține pe fiecare linie câte una din instrucțiunile SUS, JOS, DREAPTA, STANGA, scrise în ordinea executării lor de către robot. Dacă problema admite mai multe soluții, în fișierul de ieșire se va scrie doar una, oricare din ele.

Exemplu:

HRUBE.IN

7	9							
1	1	1	1	1	1	1	1	1
1	0	0	1	0	0	0	0	1
1	0	1	0	0	1	0	1	1
1	0	0	0	1	0	0	0	1
1	0	1	0	1	0	1	0	1
1	0	0	0	0	0	1	0	1
1	2	1	1	1	1	1	3	1

HRUBE.OUT

SUS
DREAPTA
DREAPTA
DREAPTA
DREAPTA
SUS
SUS
DREAPTA
DREAPTA
JOS
JOS
JOS

Restricții. $5 \leq n, m \leq 100$. Timpul de execuție nu va depăși 0,5 secunde. Fișierul sursă se va numi HRUBE.PAS, HRUBE.C sau HRUBE.CPP.

* Hrubă – încăpere sau galerie subterană care servește la depozitarea produselor alimentare. În hrubele de la Cricova pe parcursul a mai multor decenii sunt depozitate cele mai bune vinuri din Republica Moldova.

Rezolvare

Vom denumi *perete* orice succesiune de laturi adiacente de pătrățele, fiecare din care separă un pătrățel nehașurat (spațiu liber) de unul hașurat (obstacol). Conform condițiilor problemei, întotdeauna există un perete, care se începe la intrare și se termină la ieșire.

Prin urmare, pentru a găsi ieșirea, este suficient să deplasăm robotul de-a lungul unui și acelui perete, de exemplu, cel din „mâna dreaptă” a robotului. Evident, în procesul deplasării, robotul nu trebuie să se îndepărteze de peretele respectiv sau, prin alte cuvinte, el trebuie să țină permanent “mâna dreaptă” pe perete.

Pentru a simula procesul de deplasare a robotului, introducem în studiu următoarele variabile:

x, y – coordonatele curente ale robotului;

D – direcția în care este orientat robotul (vom nota aceste direcții prin NORD, SUD, EST, VEST);

C – instrucțiunea pe care trebuie să o execute robotul.

Coordonatele următorului pătrățel liber în care trebuie să treacă robotul se determină, analizând direcția în care el este orientat și starea pătrățelelor adiacente.

De exemplu, presupunem că robotul este orientat în direcția $D = \text{NORD}$.

Analizăm mai întâi pătrățelul $(x, y + 1)$ din “mâna dreaptă” a robotului. Dacă acest pătrățel este nehașurat, robotul trece în el și își schimbă orientarea: $C := \text{DREAPTA}$; $y := y + 1$; $D := \text{EST}$.

Dacă pătrățelul din „mâna dreaptă” a robotului este hașurat, analizăm pătrățelul $(x - 1, y)$ din „fața” robotului. Dacă acest pătrățel este nehașurat, robotul trece în el fără ași schimba orientarea: $C := \text{SUS}$, $x := x - 1$.

Dacă ambele pătrățele, și cel din „mâna dreaptă”, și cel din „fața” robotului sunt hașurate, analizăm pătrățelul $(x, y - 1)$ din “mâna stângă” a robotului. Dacă acest pătrățel este nehașurat, robotul trece în el și își schimbă orientarea: $C := \text{STANGA}$; $y := y - 1$; $D := \text{VEST}$.

În sfârșit, dacă pătrățelele din „mâna dreaptă”, din „față” și din “mâna stângă” a robotului sunt hașurate, robotul doar se „întoarce” înapoi, cu „fața” către pătrățelul din care a venit: $D := \text{SUD}$.

Într-un mod similar se analizează și cazurile în care robotul este orientat spre SUD, EST sau VEST.

```
Program Hrube;
{ Clasele 07-09 }
const nmax=100; mmax=100;
type Directie = (NORD, SUD, EST, VEST);

var A : array[1..nmax, 1..mmax] of integer;
    n, m : integer; { dimensiunile campului }
    p, q : integer; { coordonatele intrarii }
    r, s : integer; { coordonatele iesirii }
    DI : Directie; { orientarea initiala a robotului }
    Iesire : text;

procedure Citeste;
{ Citeste datele de intrare }
var i, j : integer;
    Intrare : text;
begin
    assign(Intrare, 'HRUBE.IN');
```

```

reset(Intrare);
readln(Intrare, n, m);
for i:=1 to n do
  begin
    for j:=1 to m do
      begin
        read(Intrare, A[i, j]);
        if A[i, j]=2 then begin p:=i; q:=j; A[i, j]:=0; end;
        if A[i, j]=3 then begin r:=i; s:=j; A[i, j]:=0; end;
      end; { for }
    readln(Intrare);
  end; { for }
close(Intrare);
if q=1 then DI:=EST;
if q=m then DI:=VEST;
if p=1 then DI:=SUD;
if p=n then DI:=NORD;
end; { Citeste }

procedure Mergi(x, y : integer; D : Directie);
label 1;
begin
  repeat
    if D=NORD then
      begin
        if A[x, y+1]=0 then
          begin writeln(Iesire, 'DREAPTA'); y:=y+1; D:=EST; goto 1; end;
        if A[x-1, y]=0 then
          begin writeln(Iesire, 'SUS'); x:=x-1; D:=NORD; goto 1; end;
        if A[x, y-1]=0 then
          begin writeln(Iesire, 'STANGA'); y:=y-1; D:=VEST; goto 1; end;
        D:=SUD; goto 1;
      end; { Nord }
    if D=SUD then
      begin
        if A[x, y-1]=0 then
          begin writeln(Iesire, 'STANGA'); y:=y-1; D:=VEST; goto 1; end;
        if A[x+1, y]=0 then
          begin writeln(Iesire, 'JOS'); x:=x+1; D:=SUD; goto 1; end;
        if A[x, y+1]=0 then
          begin writeln(Iesire, 'DREAPTA'); y:=y+1; D:=EST; goto 1; end;
        D:=NORD; goto 1;
      end; { SUD }
    if D=EST then
      begin
        if A[x+1, y]=0 then
          begin writeln(Iesire, 'JOS'); x:=x+1; D:=SUD; goto 1; end;
        if A[x, y+1]=0 then
          begin writeln(Iesire, 'DREAPTA'); y:=y+1; D:=EST; goto 1; end;
        if A[x-1, y]=0 then
          begin writeln(Iesire, 'SUS'); x:=x-1; D:=NORD; goto 1; end;
        D:=VEST; goto 1;
      end; { EST }
    if D=VEST then
      begin
        if A[x-1, y]=0 then
          begin writeln(Iesire, 'SUS'); x:=x-1; D:=NORD; goto 1; end;
        if A[x, y-1]=0 then
          begin writeln(Iesire, 'STANGA'); y:=y-1; D:=VEST; goto 1; end;
        if A[x+1, y]=0 then
          begin writeln(Iesire, 'JOS'); x:=x+1; D:=SUD; goto 1; end;
        D:=EST; goto 1;
      end; { VEST }
  until 1;

```

```

    until ((x=r) and (y=s));
end; { Mergi }

begin
    Citeste;
    assign(Iesire, 'HRUBE.OUT');
    rewrite(Iesire);
    Mergi(p, q, DI);
    close(Iesire);
end.


```

Pentru a evalua complexitatea temporală a algoritmului, vom porni de la faptul că numărul de pătrățele ale câmpului de lucru a robotului nu depășește valoarea nm . Prin urmare, numărul de repetări ale instrucțiunilor din ciclul **repeat ... until** al procedurii MERGI este de ordinul nm .

Conform restricțiilor problemei, $5 \leq n, m \leq 100$. Evident, $nm = 10^4$, mărime cu mult mai mică decât capacitatea de prelucrare a calculatoarelor personale din laboratorul de informatică.

Jocul

Se consideră un câmp de joc, format din n rânduri și m coloane (vezi desenul). La intersecțiile de rânduri și coloane se formează căsuțe. Fiecare căsuță este definită prin coordonatele sale: numărul de rând x și numărul de coloană y .

	1	2	3	4	5
1					
2					
3					
4					
5					
6					

Jucătorul aruncă un zar, care cade la întâmplare în una din căsuțe. În general, coordonatele acestei căsuțe nu sunt cunoscute apriori, însă ele pot fi aflate explorând câmpul de joc.

De exemplu, în cazul câmpului de joc de pe desen, zarul se află în căsuța cu coordonatele $x = 5$ și $y = 3$.

Sarcină. Elaborați un program care, cunoscând situația de pe câmpul de joc, află coordonatele căsuței în care se află zarul.

Date de intrare. Fișierul text `JOCUL.IN` conține pe prima linie numerele întregi n și m , separate prin spațiu. Fiecare din următoarele n linii ale fișierului de intrare conține câte un șir de caractere, format din exact m cifre binare $\{0, 1\}$. Dacă zarul se află în căsuța cu coordonatele x, y , atunci cifra binară din poziția y a liniei $x + 1$ a fișierului de intrare are valoarea 1. În caz contrar, cifra binară respectivă are valoarea 0.

Date de ieșire. Fișierul text `JOCUL.OUT` va conține pe o singură linie două numere întregi, separate prin spațiu: coordonatele x, y ale căsuței în care se află zarul.

Exemplu. Pentru desenul din enunțul problemei, fișierele de intrare și ieșire sunt:

JOCUL.IN	JOCUL.OUT
6 5 00000 00000 00000 00000 00100 00000	5 3

Restricții. $1 \leq n, m \leq 1000$. Timpul de execuție nu va depăși 1,0 secunde. Programul va folosi cel mult 32 Megaocteți de memorie operativă. Fișierul sursă va avea denumirea `JOCUL.PAS`, `JOCUL.C` sau `JOCUL.CPP`.

Rezolvare

Vom explora fișierul de intrare caracter cu caracter. Evident, explorarea fișierului de intrare poate fi terminată imediat cum a fost citită cifra binară cu valoarea 1.

```
Program Jocul;  
{ Clasele 07-09 }  
label 1;  
var Intrare, Iesire : text;  
    n, m : longint;  
    x, y : longint;  
    c : char;  
begin  
    assign(Intrare, 'JOCUL.IN');  
    reset(Intrare);  
    readln(Intrare, n, m);  
    for x:=1 to n do  
        begin  
            for y:=1 to m do  
                begin  
                    read(Intrare, c);  
                    if c='1' then goto 1;  
                end; { for }  
            readln(Intrare);  
        end; { for }  
1:  
    close(Intrare);  
    assign(Iesire, 'JOCUL.OUT');  
    rewrite(Iesire);  
    writeln(Iesire, x, ' ', y);  
    close(Iesire);  
end.
```

Pătratul numărului binar

Se consideră un număr binar natural x , format din n cifre binare $\{0, 1\}$. De exemplu, numărul binar $x = 1001$ este format din $n = 4$ cifre binare.

Sarcină. Elaborați un program care calculează numărul binar y , $y = x^2$.

Date de intrare. Fișierului text PATRAT.IN conține pe o singură linie numărul binar x .

Date de ieșire. Fișierului text PATRAT.OUT va conține pe o singură linie numărul binar y , scris fără zerouri ne semnificative.

Exemplu.

PATRAT.IN	PATRAT.OUT
1001	1010001

Restricții. $1 \leq n \leq 120$. Timpul de execuție nu va depăși 0,5 secunde. Programul va folosi cel mult 32 Megaocteți de memorie operativă. Fișierul sursă va avea denumirea PATRAT.PAS, PATRAT.C sau PATRAT.CPP.

Rezolvare

Vom memora numărul binar x într-un șir de caractere. Pătratul acestui număr îl vom calcula implementând algoritmul de înmulțire a două numere binare „în coloniță”.

```
Program PatratulNumaruluiBinar;
{ Clasele 07-09 }

var A, B : string;

procedure Citeste(var X : string);
{ Citeste X din fisierul de intrare }
var Intrare : text;
begin
    assign(Intrare, 'PATRAT.IN');
    reset(Intrare);
    readln(Intrare, X);
    close(Intrare);
end; { Citeste }

procedure Scrie(X : string);
{ Scrie X in fisierul de iesire }
var Iesire : text;
begin
    assign(Iesire, 'PATRAT.OUT');
    rewrite(Iesire);
    writeln(Iesire, X);
    close(Iesire);
end; { Scrie }

procedure AdunaCifreBinare(a, b, c : char; var s, t : char);
{ Aduna cifrele binare a, b, c }
{ Returneaza suma s si transportul t }
begin
    if (a='0') and (b='0') and (c='0') then begin t:='0'; s:='0' end;
    if (a='0') and (b='0') and (c='1') then begin t:='0'; s:='1' end;
    if (a='0') and (b='1') and (c='0') then begin t:='0'; s:='1' end;
```

```

    if (a='0') and (b='1') and (c='1') then begin t:='1'; s:='0' end;
    if (a='1') and (b='0') and (c='0') then begin t:='0'; s:='1' end;
    if (a='1') and (b='0') and (c='1') then begin t:='1'; s:='0' end;
    if (a='1') and (b='1') and (c='0') then begin t:='1'; s:='0' end;
    if (a='1') and (b='1') and (c='1') then begin t:='1'; s:='1' end;
end; { AdunaCifre }

procedure AliniazaNumereBinare(var X, Y : string);
{ Aduce numerele X, Y la acelasi numar de cifre binare }
label 1;
var n, m, i : integer;
begin
    n:=length(X);
    m:=length(Y);
    if n=m then goto 1;
    if n<m then for i:=1 to m-n do X:='0'+X
        else for i:=1 to n-m do Y:='0'+Y;
1: end; { AliniazaNumereBinare }

procedure AdunaNumereBinare(X, Y : string; var Z : string);
{ Calculeaza Z:=X+Y }
var t : char; { transportul }
    i : integer;
begin
    AliniazaNumereBinare(X, Y);
    Z:='0';
    AliniazaNumereBinare(X, Z);
    t:='0';
    for i:=length(X) downto 1 do
        AdunaCifreBinare(t, X[i], Y[i], Z[i], t);
    if t='1' then Z:='1'+Z;
end; { AdunaNumereBinare }

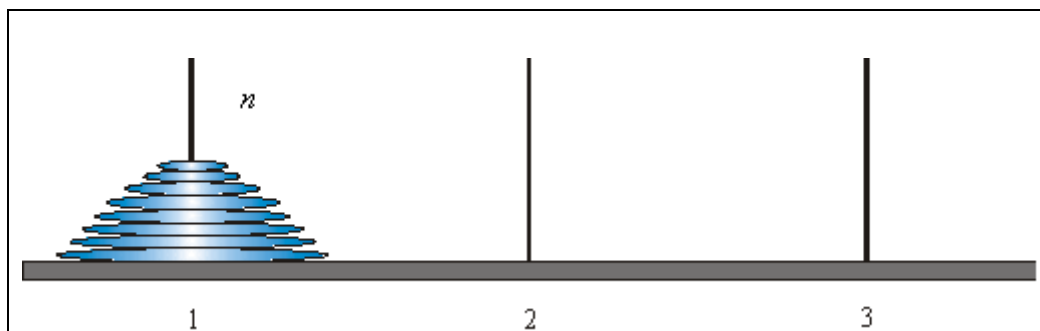
procedure RidicaLaPatrat(X : string; var Z : string);
{ Calculeaza Z:=X*X }
var Y : string;
    i : integer;
begin
    Y:=X;
    Z:='0';
    for i:=length(X) downto 1 do
        begin
            if X[i]='1' then AdunaNumereBinare(Y, Z, Z);
            Y:=Y+'0'; { Deplaseaza numarul Y la stanga }
        end; { for }
end; { RidicaLaPatrat }

begin
    Citeste(A);
    RidicaLaPatrat(A, B);
    Scrie(B);
end.

```

Turnuri

Se consideră trei tije notate prin 1, 2 și 3 și n discuri perforate de dimensiuni diferite (vezi desenul). Inițial toate discurile sunt pe tija 1, fiind aranjate în ordinea descrescătoare a diametrelor, considerând sensul de la bază la vârf.



Sarcină. Elaborați un program care mută toate discurile de pe tija 1 pe tija 3, folosind tija 2 ca tijă de manevră și respectând următoarele reguli:

- la fiecare pas se mută un singur disc;
- orice disc poate fi așezat doar peste un disc cu diametrul mai mare.

Date de intrare. Fișierul text `TURNURI.IN` conține pe o singură linie numărul întreg n – numărul de discuri de pe tija 1.

Date de ieșire. Fișierul text `TURNURI.OUT` va conține pe fiecare linie câte două numere întregi i și j , separate prin spațiu. Fiecare linie a fișierului de ieșire descrie câte o mutare de disc: i indică tija de pe care se ia discul, iar j – tija pe care se pune discul. Mutările apar în fișierul de ieșire în ordinea efectuării lor.

Exemplu. Pentru $n = 3$, fișierele de intrare și ieșire sunt:

TURNURI.IN	TURNURI.OUT
3	1 3
	1 2
	3 2
	1 3
	2 1
	2 3
	1 3

Restricții. $1 \leq n \leq 20$. Nu se admit mutări în care $i = j$. Timpul de execuție nu va depăși 1,0 secunde. Programul va folosi cel mult 32 Megaocteți de memorie operativă. Fișierul sursă va avea denumirea `TURNURI.PAS`, `TURNURI.C` sau `TURNURI.CPP`.

Rezolvare

Pentru a rezolva această problemă vom folosi o procedură recursivă, pe care o vom denumi-o *MutaDiscurile*. Evident, această procedură trebuie să aibă un antet de forma:

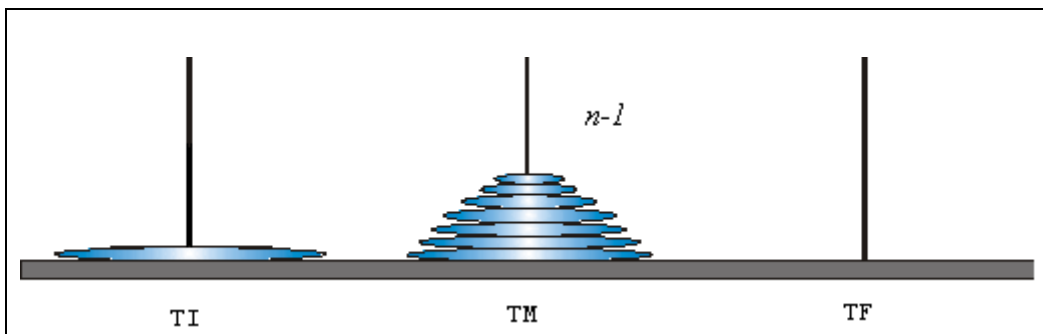
```
procedure MutaDiscurile(n, TI, TF, TM : integer);
```

unde n este numărul de discuri, TI – tija inițială, TF – tija finală, iar TM – tija de manevră.

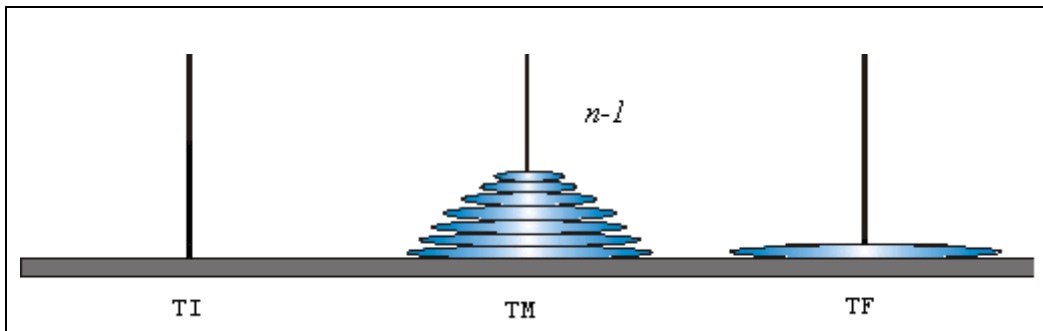
În cazul unui singur disc, adică $n = 1$, această procedură trebuie, pur și simplu, să mute discul de pe tija TI pe tija TF .

În cazurile cu mai multe discuri, adică $n > 1$, reducem problema cu n discuri la cea cu $(n - 1)$ discuri:

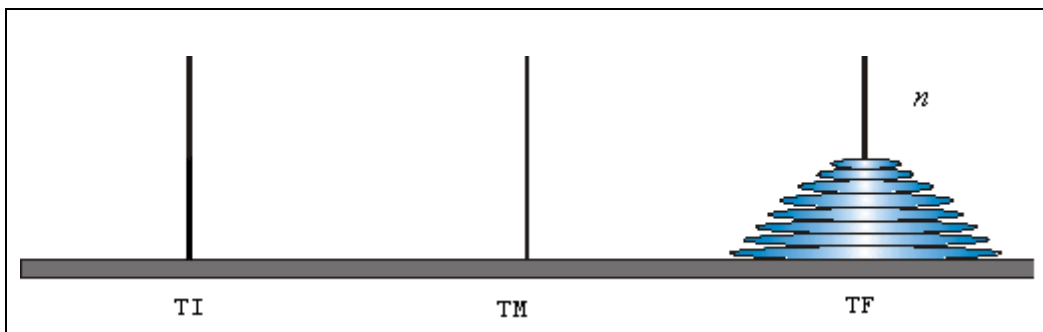
1. Apelăm procedura *MutaDiscurile* pentru a muta $n - 1$ discuri de pe tija TI pe tija TM , folosind ca tija de manevră tija TF .



2. Mutăm discul rămas de pe tija TI pe tija TF .



3. Apelăm procedura *MutaDiscurile* pentru a muta discurile de pe tija TM pe tija TF , folosind ca tija de manevră tija TI .



Prezentăm în continuare programul ce implementează procedura MutaDiscurile.

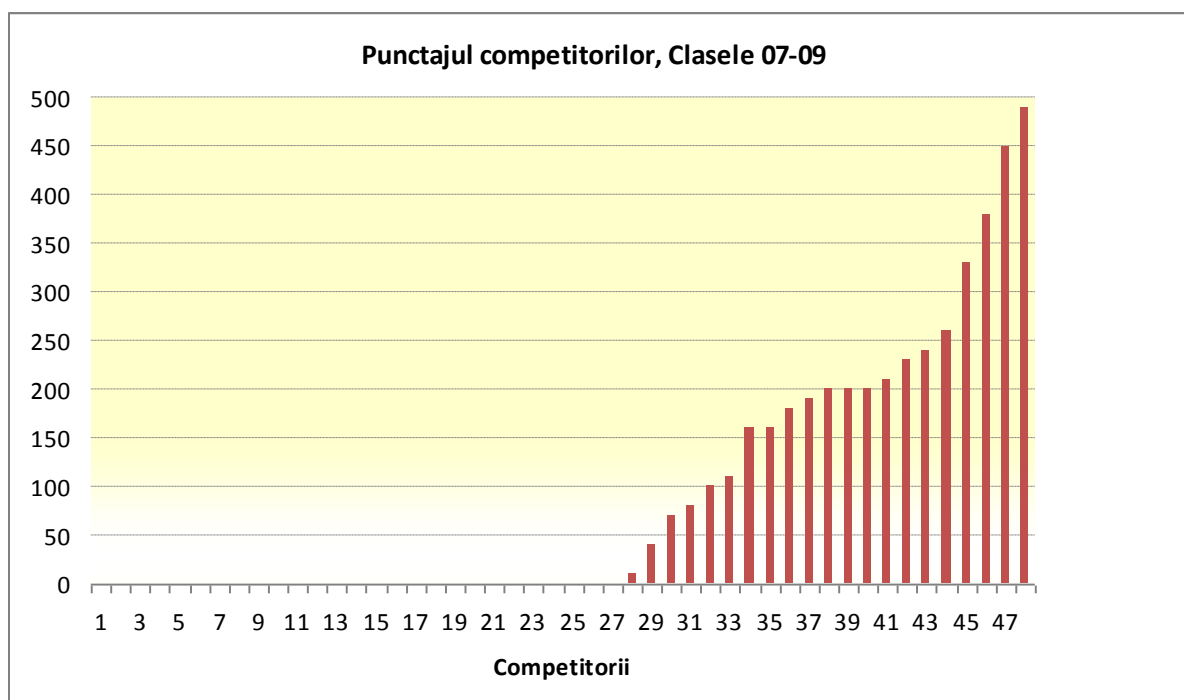
```
program Turnuri;
{ Clasele 07-09 }
var n : integer;
    Intrare, Iesire : text;

procedure MutaDiscurile(n, TI, TF, TM : integer);
begin
    if n=1 then writeln (Iesire, TI, ' ', TF)
    else
        begin
            MutaDiscurile(n-1, TI, TM, TF);
            writeln (Iesire, TI, ' ', TF);
            MutaDiscurile(n-1, TM, TF, TI);
        end; { else }
end; { MutaDiscurile }

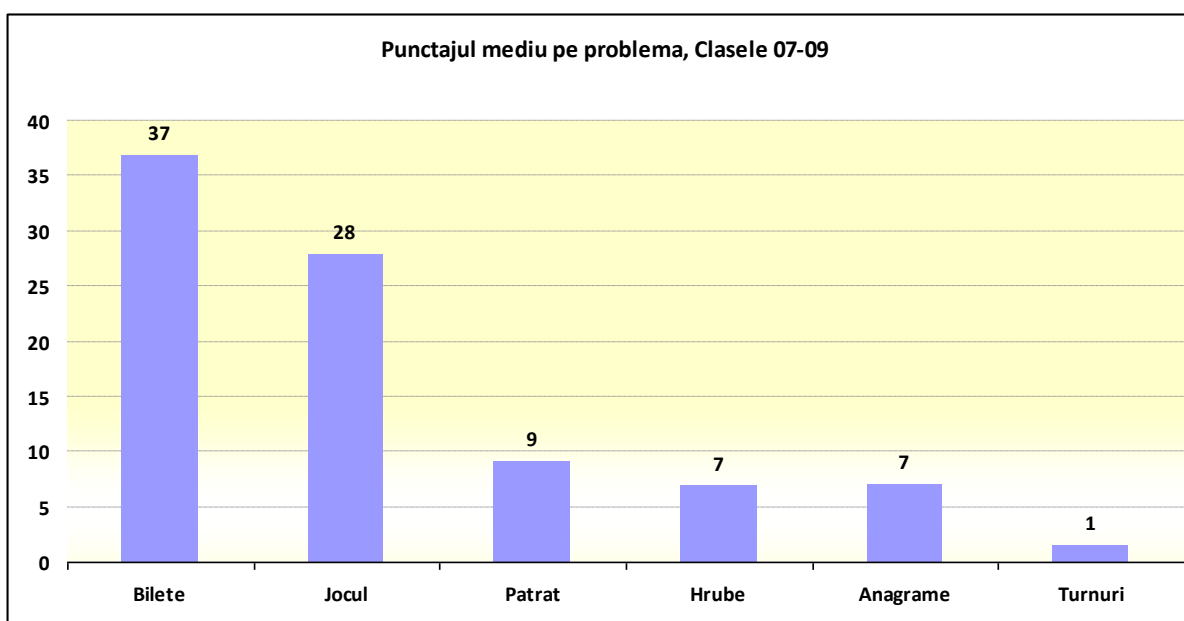
begin
    assign(Intrare, 'TURNURI.IN');
    reset(Intrare);
    readln(Intrare, n);
    close(Intrare);
    assign(Iesire, 'TURNURI.OUT');
    rewrite(Iesire);
    MutaDiscurile(n, 1, 3, 2);
    Close(Iesire);
end.
```

Prin măsurări directe de putem convinge că timpul de execuție a programului Turnuri corespunde restricțiilor problemei.

Punctajul competitorilor



Complexitatea problemelor



Clasele 10–12

Denumirea problemei	Numărul de puncte alocat problemei	Denumirea fișierului sursă	Denumirea fișierului de intrare	Denumirea fișierului de ieșire
Comoara	100	COMOARA.PAS COMOARA.C COMOARA.CPP	COMOARA.IN	COMOARA.OUT
Bilete	100	BILETE.PAS BILETE.C BILETE.CPP	BILETE.IN	BILETE.OUT
Hrubele de la Cricova	100	HRUBE.PAS HRUBE.C HRUBE.CPP	HRUBE.IN	HRUBE.OUT
Jocul	100	JOCUL.PAS JOCUL.C JOCUL.CPP	JOCUL.IN	
Rădăcina numărului binar	100	RAD.PAS RAD.C RAD.CPP	RAD.IN	RAD.OUT
Turnuri	100	TURNURI.PAS TURNURI.C TURNURI.CPP	TURNURI.IN	TURNURI.OUT
Total	600	–	–	–

Comoara

Vânătorii de comori au descoperit în una din încăperile închise ale unui castel medieval n lingouri de aur de dimensiuni distincte. Fiecare lingou i , $i = 1, 2, 3, \dots, n$, reprezintă un paralelipiped dreptunghiular cu dimensiunile $x_i \times y_i \times z_i$. Pentru a scoate lingourile la lumina zilei, vânătorii de comori trebuie să perforeze într-un zid de piatră unul sau mai multe orificii dreptunghiulare, care nu au puncte de tangență. Un lingou poate fi scos printr-un orificiu dreptunghiular doar atunci când lățimea și înălțimea orificiului sunt egale sau mai mari ca lățimea și înălțimea uneia din fețele dreptunghiulare ale paralelipipedului. Evident, lingoul poate fi rotit în orice fel.

Pentru ași ușura munca, vânătorii de comori doresc ca suma ariilor orificiilor ce trebuie perforate să fie cât mai mică.

Sarcină. Elaborați un program care, cunoscând dimensiunile lingourilor de aur, calculează valoarea minimă S a sumei ariilor orificiilor de perforat, ce vor fi suficiente pentru a scoate toate lingourile.

Date de intrare. Fișierul text `COMOARA.IN` conține pe prima linie numărul întreg n . Fiecare din următoarele n linii ale fișierului de intrare conține câte trei numere întregi, separate prin spațiu. Linia $i+1$ a fișierului de intrare conține dimensiunile x_i , y_i și z_i ale lingoului i .

Date de ieșire. Fișierul text `COMOARA.OUT` va conține pe o singură linie numărul întreg S – valoarea minimă a sumei ariilor orificiilor de perforat.

Exemplu.

COMOARA.IN	COMOARA.OUT
3	8
1 4 4	
5 3 2	
1 2 2	

Restricții. $1 \leq n \leq 5000$; $1 \leq x_i, y_i, z_i \leq 10000$. Timpul de execuție nu va depăși 3,0 secunde. Programul va folosi cel mult 32 Megaocteți de memorie operativă. Fișierul sursă va avea denumirea `COMOARA.PAS`, `COMOARA.C` sau `COMOARA.CPP`.

Rezolvare

Mai întâi vom menționa, că în cazul unui singur lingou i , dimensiunile dreptunghiului de cea mai mică arie se determină în mod trivial, selectând din valorile x_i, y_i, z_i doar două, cele mai mici din ele. În continuare, pentru fiecare lingou i vom nota astfel de dimensiuni prin x_i, y_i și le vom aranja în ordine descrescătoare, adică $x_i \geq y_i$.

Valoarea minimă a sumei ariilor orificiilor de perforat poate fi calculată prin metoda programării dinamice după cum urmează:

1. Conform dimensiunilor x , sortăm lingourile în ordine crescătoare. Dacă există două sau mai multe lingouri cu aceeași dimensiune x , sortăm astfel de lingouri în ordine crescătoare conform dimensiunii y .

2. Eliminăm din studiu lingourile mici, care „încap” integral în alte lingouri, mai mari. În acest scop, parcurgem lingourile deja sortate și, la fiecare pas i , eliminăm toate lingourile

precedente j , pentru care $x_j \leq x_i$ și $y_j \leq y_i$. Se poate observa că după eliminare, pentru lingourile ramase se respectă inegalitățile $x_{i-1} < x_i$ și $y_{i-1} > y_i$.

3. Fie n numărul curent de lingouri rămase. Introducem în studiu șirul $F(0)$, $F(1)$, ..., $F(i)$, ..., $F(n)$, în care $F(i)$ este aria minima a orificiilor ce trebuie perforate pentru a scoate primele i lingouri în ordinea stabilită la pasul 2. Valorile acestui șir pot fi calculate conform următoarelor formule recurente:

$$F(0) = 0;$$

$$F(i) = \min\{F(j-1) + y_j x_i \mid j = 1, \dots, i\}, i = 1, 2, \dots, n.$$

```

Program Comoara;
const MAXN = 5000;
        MAXXY = 10000;
type TLingou = record
                x, y : longint;
            end;
var n : longint;
    L : array[1..MAXN] of TLingou;
    F : array[0..MAXN] of longint;

procedure Citeste;
{ Citeste datele de intrare }
{ Selecteaza cele doua, mai mici, dimensiuni ale lingoului }
{ Ordoneaza dimensiunile selectate in ordine descrescatoare }
var Intrare : text;
    x, y, z : longint; { dimensinule lingoului curent }
    i : longint;
    r : longint;
begin
    assign(Intrare, 'COMOARA.IN');
    reset(Intrare);
    readln(Intrare, n);
    for i:=1 to n do
        begin
            readln(Intrare, x, y, z);
            if ((z>=x) and (z>=y)) then begin L[i].x:=x; L[i].y:=y end;
            if ((y>=x) and (y>=z)) then begin L[i].x:=x; L[i].y:=z end;
            if ((x>=y) and (x>=z)) then begin L[i].x:=y; L[i].y:=z end;
            if (L[i].x<L[i].y) then
                begin
                    r:=L[i].x; L[i].x:=L[i].y; L[i].y:=r;
                end; { then }
        end; { for }
    close(Intrare);
end; { Citeste }

procedure Scrie;
var Iesire : text;
begin
    assign(Iesire, 'COMOARA.OUT');
    rewrite(Iesire);
    writeln(Iesire, F[n]);
    close(Iesire);
end; { Scrie }

function MaiMic(i, j : longint) : boolean;
{ Compara lingourile i si j }
begin
    if (L[i].x<>L[j].x) then MaiMic:=(L[i].x<L[j].x)

```

```

        else MaiMic:=(L[i].y<L[j].y);
end; { MaiMic }

Procedure SorteazaLingourile;
var i : longint;
    aux : TLingou;
    ok : boolean;
begin
    ok := true;
    while ok do
        begin
            ok := false;
            for i:=1 to n-1 do
                if MaiMic(i+1, i) then
                    begin
                        aux:=L[i];
                        L[i]:=L[i+1];
                        L[i+1]:=aux;
                        ok:=true;
                    end; { for }
            end; { while }
        end; { SortareaLingourilor }

procedure EliminaLingourileMici;
var i, j : longint;
begin
    j:=1;
    for i:=2 to n do
        begin
            while (j>0) and (L[j].x<=L[i].x) and (L[j].y<=L[i].y)
                do j:=j-1;
            j:=j+1;
            L[j]:=L[i];
        end; { for }
    n:=j;
end; { EliminaLingourileMici }

procedure CalculeazaAria;
var i, j, min, cost : longint;
begin
    F[0]:=0;
    for i:=1 to n do
        begin
            min:=MAXXY*MAXXY+1;
            for j:=1 to i do
                begin
                    cost:=F[j-1]+L[j].y*L[i].x;
                    if (cost<min) then min:=cost;
                end;
            F[i]:=min;
        end;
    end; { CalculeazaAria }

begin
    Citeste;
    SorteazaLingourile;
    EliminaLingourileMici;
    CalculeazaAria;
    Scrie;
end.

```

Timpul cerut de procedurile programului Comoara poate fi estimat analizând textele respective:

SorteazaLingourile	– $T(n) \approx 14n^2 \cdot \Delta$;
EliminaLingourileMici	– $T(n) \approx 8n \cdot \Delta$;
CalculeazaAria	– $T(n) \approx 12n^2 \cdot \Delta$,

unde Δ este timpul necesar pentru efectuarea unei operații elementare. Amintim, că în cazul calculatoarelor din laboratorul de informatică, $\Delta \approx 10^{-9}$ secunde.

Prin urmare, timpul cerut de programul `Comoara` va fi $T(n) \approx 26n^2 \cdot \Delta$. Pentru cel mai greu caz, $n = 5000$ și $T(5000) \approx 0,65$ secunde, mărime comparabilă cu cea indicată în restricțiile problemei.

Întrucât metoda de evaluare a timpului cerut de programul `Comoara` este una orientativă, pentru a fi siguri că programul se încadrează în timpul cerut de restricțiile problemei, ar fi bine să elaborăm un test cu $n = 5000$ de lingouri și să efectuăm măsurări directe.

Generând în mod aleator 5000 de paralelipede cu dimensiuni ce se încadrează în intervalul indicat în restricțiile problemei, ne putem convinge că timpul de execuție a programului `Comoara` nu depășește valoarea de 3,0 secunde.

Bilete



Biletele de troleibuz din marele orașe sunt numerotate. De obicei, un număr de bilet este format din 6 cifre zecimale.

Unii elevi și studenți consideră că în cazul în care suma primelor trei cifre din componența numărului de bilet este egală cu suma ultimelor trei cifre, biletul respectiv aduce noroc sau, prin alte cuvinte, este un bilet „fericit”.

De exemplu, biletul cu numărul 032050 este „fericit”, iar cel cu numărul 283375 – nu.

În general, la început de rută, taxatorul primește o bobină de bilete numerotate consecutiv și semnează un borderou în care este indicat numărul primului bilet m și numărul total de bilete n din bobina respectivă.

Apare întrebarea, câte bilete “fericite” conține bobina taxatorului.

Sarcină. Elaborați un program care, cunoscând numărul primului bilet m și numărul total de bilete n din bobina taxatorului, calculează numărul de bilete „fericite” k din ea.

Date de intrare. Fișierul text BILETE.IN conține pe o singură linie numerele întregi m și n , separate prin spațiu.

Date de ieșire. Fișierul text BILETE.OUT va conține pe o singură linie numărul întreg k .

Exemplu.

BILETE.IN	BILETE.OUT
332206 10	2

Restricții. $000000 \leq m \leq 999999$; $n \leq (999999 - m + 1)$. Timpul de execuție nu va depăși 1,0 secunde. Programul va folosi cel mult 32 Megaocteți de memorie operativă. Fișierul sursă va avea denumirea BILETE.PAS, BILETE.C sau BILETE.CPP.

Rezolvare

Vom nota prin i numărul biletului curent, iar prin $c_1, c_2, c_3, c_4, c_5, c_6$ – cifrele din componența acestui număr. De exemplu, în cazul numărului de bilet $i = 360710$ avem $c_1 = 3$, $c_2 = 6$, $c_3 = 0$, $c_4 = 7$, $c_5 = 1$, $c_6 = 0$.

Cifrele numărului de bilet i pot fi calculate în ordinea $c_6, c_5, c_4, c_3, c_2, c_1$ prin împărțiri succesive a numărului de bilet și a căturilor obținute la baza 10 a sistemului zecimal de numerație, reținând resturile împărțirilor.

Evident, un bilet este “fericit” atunci când $(c_1 + c_2 + c_3) = (c_4 + c_5 + c_6)$.

Pentru a calcula numărul de bilete „fericite” din bobină, este suficient să examinăm toate valorile lui i din intervalul m (numărul primului bilet din bobină) ... $m + n - 1$ (numărul ultimului bilet din bobină).

```
Program Bilete;  
{ Clasele 10-12 }  
var m : longint; { numarul primului bilet din bobina }  
    n : longint; { numarul total de bilete in bobina }  
    k : longint; { numarul biletelor fericite din bobina }
```

```

procedure Citeste;
var Intrare : text;
begin
    assign(Intrare, 'BILETE.IN');
    reset(Intrare);
    readln(Intrare, m, n);
    close(Intrare);
end; { Citeste }

procedure Scrie;
var iesire : text;
begin
    assign(Iesire, 'BILETE.OUT');
    rewrite(Iesire);
    writeln(Iesire, k);
    close(Iesire);
end; { Scrie }

procedure Cifre(i : longint; var c1, c2, c3, c4, c5, c6 : integer);
    { returneaza cifrele c1, c2, ..., c6 ale numarului de bilet i }
begin
    c6:=i mod 10; i:=i div 10;
    c5:=i mod 10; i:=i div 10;
    c4:=i mod 10; i:=i div 10;
    c3:=i mod 10; i:=i div 10;
    c2:=i mod 10; i:=i div 10;
    c1:=i mod 10;
end; { Cifre }

procedure Calculeaza;
    { calculeaza numarul de bilete fericite }
var
    u : longint; { numarul ultimului bilet din bobina }
    i : longint; { numarul biletului curent }
    c1, c2, c3, c4, c5, c6 : integer; { cifrele biletului curent }
begin
    u:=m+n-1;
    k:=0;
    for i:=m to u do
        begin
            Cifre(i, c1, c2, c3, c4, c5, c6);
            if (c1+c2+c3)=(c4+c5+c6) then k:=k+1;
        end; { for }
end; { Calculeaza }

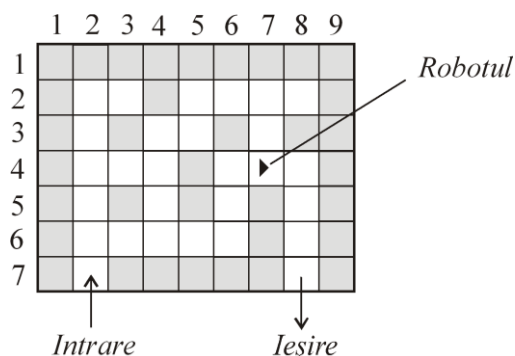
begin
    Citeste;
    Calculeaza;
    Scrie;
end.

```

Din analiza textului procedurilor Cifra și Calculeaza se observă că timpul cerut de algoritm este de ordinul $T(n) \approx 20n \cdot \Delta$, unde Δ este timpul necesar pentru efectuarea unei operații elementare. Conform restricțiilor problemei, $n \leq (999999 - m + 1)$ și $000000 \leq m \leq 999999$. Evident, $n \leq 10^7$ și, pentru cel mai greu caz, $T(10^7) \approx 2\Delta \cdot 10^8$. Întrucât pentru calculatoarele din laboratorul de informatică $\Delta \approx 10^{-9}$ secunde, pentru cazul cel mai greu $T \approx 0,2$ secunde, mărime ce nu încalcă restricțiile problemei.

Hrubele de la Cricova

După o vizită la renumitele hrube* de la Cricova, un informatician a construit un robot care lucrează într-un câmp divizat în pătrățele (vezi desenul). Pătrățelele hașurate reprezintă obstacolele, iar cele nehașurate – spațiile libere. Pătrățelele de pe perimetrul planului, cu excepția celor de intrare sau ieșire, sînt hașurate prin definiție.



Robotul poate executa doar instrucțiunile SUS, JOS, DREAPTA, STANGA, conform cărora el se deplasează în unul din pătrățele vecine. Dacă în pătratul vecin este un obstacol, de exemplu, un perete sau un butoi, are loc un accident și robotul iese din funcțiune.

În formă numerică planul hrubelor este redat prin tabloul A cu n linii și m coloane. Elementele $A[i, j]$ ale acestui tablou au următoarea semnificație: 0 – spațiu liber; 1 – obstacol; 2 – intrarea în hrube; 3 – ieșirea din hrube. Inițial, robotul se află în pătrățul pentru care $A[i, j] = 2$.

Sarcină. Elaborați un program, care, cunoscând planul hrubelor, deplasează robotul prin încăperile subterane, de la intrarea în hrube la ieșire, utilizând un număr cât mai mic de instrucțiuni. Colecția de vinuri fiind foarte bogată, nu se cere vizitarea obligatorie a tuturor încăperilor subterane.

Date de intrare. Fișierul text HRUBE.IN conține pe prima linie numerele întregi n, m , separate prin spațiu. Pe fiecare din următoarele n linii se conțin câte m numere întregi $A[i, j]$, separate prin spațiu.

Date de ieșire. Fișierul text HRUBE.OUT va conține pe prima linie un număr întreg L – numărul minim de instrucțiuni, necesare pentru deplasarea robotului de la intrare la ieșire. Fiecare din următoarele L linii ale fișierului de ieșire va conține câte una din instrucțiunile SUS, JOS, DREAPTA, STANGA, scrise în ordinea executării lor de către robot. Dacă problema admite mai multe soluții, în fișierul de ieșire se va scrie doar una, oricare din ele.

Exemplu:

HRUBE.IN

7	9							
1	1	1	1	1	1	1	1	1
1	0	0	1	0	0	0	0	1
1	0	1	0	0	1	0	1	1
1	0	0	0	1	0	0	0	1
1	0	1	0	1	0	1	0	1
1	0	0	0	0	0	1	0	1
1	2	1	1	1	1	1	3	1

HRUBE.OUT

12
SUS
DREAPTA
DREAPTA
DREAPTA
DREAPTA
SUS
SUS
DREAPTA
DREAPTA
JOS
JOS
JOS

* Hrubă – încăpere sau galerie subterană care servește la depozitarea produselor alimentare. În hrubele de la Cricova pe parcursul a mai multor decenii sunt depozitate cele mai bune vinuri din Republica Moldova.

Restricții. $5 \leq n, m \leq 100$. Timpul de execuție nu va depăși 3 secunde. Programul va folosi cel mult 32 Megaocteți de memorie operativă. Fișierul sursă se va numi HRUBE.PAS, HRUBE.C sau HRUBE.CPP.

Rezolvare

Pentru a găsi unul din cele mai scurte drumuri ce leagă intrarea cu ieșirea, vom propaga prin hrube o undă numerică, care pornește din pătrățelul de intrare.

Inițial, înscriem în pătrățelul de intrare valoarea unei numerice $u = 0$ și marcăm acest pătrățel ca fiind unul ocupat.

Presupunem, că la un anumit pas, unda numerică a ajuns în anumite pătrățele și are deja valoarea curentă u .

Următorul pas constă în identificarea pătrățelelor în care este înscrisă valoarea u și înscrierea în pătrățelele vecine, care sunt libere, a valorii $u + 1$.

În continuare, stabilim $u := u + 1$, efectuăm următorul pas ș.a.m.d., până când unda numerică ajunge în pătrățelul de ieșire.

Evident, valoarea unei numerice u , care este înscrisă în pătrățelul de ieșire, reprezintă numărul minim de instrucțiuni L , necesare pentru a deplasa robotul de la intrare la ieșire.

Pentru a afla instrucțiunile propriu-zise, care deplasează robotul pe cel mai scurt drum, ne vom mișca imaginar de la ieșire la intrare, în sens contrar direcției de propagare a unei numerice.

Inițial, stabilim ca punct de plecare pătrățelul de ieșire și valoarea curentă a unei numerice $u = L$.

Presupunem, că la un anumit pas s-a ajuns la pătrățelul în care este înscrisă valoarea numerică u .

Următorul pas constă în identificarea oricăruia din pătrățelele vecine, în care este înscrisă valoarea $u - 1$. În funcție de poziție relativă a unuia astfel de pătrățel, memorăm în secvența de instrucțiuni una din instrucțiunile SUS, JOS, STANGA, DREAPTA și trecem în pătrățelul respectiv.

În continuare, stabilim $u := u - 1$ ș.a.m.d., până se ajunge în pătrățelul de intrare.

În programul ce urmează, pentru a nu confunda valorile unei numerice cu valorile inițiale 1, 2 și 3 din fișierul de intrare, în pătrățelele ce reprezintă obstacole se înscrie valoarea „-1”.

```
Program Hrube;
{ Clasele 10-12 }
const nmax=100; mmax=100;
      Lmax=(nmax * mmax) div 2;
      Ocupat=-1;

var A : array[1..nmax, 1..mmax] of integer;
    n, m : integer; { dimensiunile campului }
    p, q : integer; { coordonatele intrarii }
    r, s : integer; { coordonatele iesirii }
    L : integer;    { lungimea secvenței de instructiuni }
    C : array[1..Lmax] of string[7]; { secvența de instructiuni }

procedure Citeste;
var i, j : integer;
    Intrare : text;
begin
    assign(Intrare, 'HRUBE.IN');
    reset(Intrare);
```



```

readln(Intrare, n, m);
for i:=1 to n do
  begin
    for j:=1 to m do
      begin
        read(Intrare, A[i, j]);
        if A[i, j]=2 then begin p:=i; q:=j; A[i, j]:=0; end;
        if A[i, j]=3 then begin r:=i; s:=j; A[i, j]:=0; end;
        if A[i, j]=1 then A[i, j]:=Ocupat;
      end; { for }
    readln(Intrare);
  end; { for }
close(Intrare);
end; { Citeste }

procedure Scrie;
var i : integer;
    Iesire : text;
begin
  assign(Iesire, 'HRUBE.OUT');
  rewrite(Iesire);
  writeln(Iesire, L);
  for i:=1 to L do
    writeln(Iesire, C[i]);
  close(Iesire);
end; { Scrie }

procedure UndaNumerica;
{ Propagarea undei numerice }
var i, j : integer;
    u : integer; { unda numerica }
begin
  u:=1;
  if p=1 then A[2, q]:=1;
  if p=n then A[n-1, q]:=1;
  if q=1 then A[p, 2]:=1;
  if q=m then A[p, m-1]:=1;
  A[p, q]:=Ocupat; { blocam intrarea }
  while A[r, s]=0 do
    begin
      for i:=1 to n do
        for j:=1 to m do
          begin
            if A[i, j]=u then
              begin
                if A[i-1, j]=0 then A[i-1, j]:=u+1; { Nord }
                if A[i+1, j]=0 then A[i+1, j]:=u+1; { Sud }
                if A[i, j-1]=0 then A[i, j-1]:=u+1; { Est }
                if A[i, j+1]=0 then A[i, j+1]:=u+1; { Vest }
              end; { then }
            end; { for }
          u:=u+1;
        end; { while }
      L:=u;
      A[p, q]:=0; { deblocam intrarea }
    end; { UndaNumerica }

procedure Drumul;
{ Scrie in C instructiunile ce deplaseaza robotul }
label 1;
var i, j : integer;
    u : integer;
begin
  if r=1 then begin C[L]:='SUS'; i:=2; j:=s; end;

```

```

if r=n then begin C[L]:='JOS'; i:=n-1; j:=s; end;
if s=1 then begin C[L]:='STANGA'; i:=r; j:=2; end;
if s=m then begin C[L]:='DREAPTA'; i:=r; j:=m-1; end;
u:=L-1;
while not ((i=p) and (j=q)) do
  begin
    if A[i-1, j]=u-1 then begin C[u]:='JOS'; i:=i-1; goto 1; end;
    if A[i+1, j]=u-1 then begin C[u]:='SUS'; i:=i+1; goto 1; end;
    if A[i, j-1]=u-1 then begin C[u]:='DREAPTA'; j:=j-1; goto 1; end;
    if A[i, j+1]=u-1 then begin C[u]:='STANGA'; j:=j+1; goto 1; end;
1:   u:=u-1;
    end; { while }
end; { Drumul }

begin
  Citeste;
  UndaNumerica;
  Drumul;
  Scrie;
end.

```

Timpul cerut de procedurile programului *Hrube* poate fi estimat analizând textele respective:

UndaNumerică $- T(n, m) \approx 26Lnm \cdot \Delta$;

Drumul $- T(n, m) \approx 11L \cdot \Delta$,

unde Δ este timpul necesar pentru efectuarea unei operații elementare, iar L – lungimea drumului.

Întrucât, $L < nm$, timpul cerut de programul *Hrube* va fi $T(n, m) \approx 26n^2m^2 \cdot \Delta$.

Amintim, că în cazul calculatoarelor din laboratorul de informatică, $\Delta \approx 10^{-9}$ secunde.


Pentru cel mai greu caz, $n = m = 100$ și $T(100, 100) \approx 2,6$ secunde, mărime comparabilă cu cea indicată în restricțiile problemei.

Întrucât metoda de evaluare a timpului cerut de programul *Hrube* este una orientativă, pentru a fi siguri că programul se încadrează în timpul cerut de restricțiile problemei, ar fi bine să elaborăm un test cu $n = m = 100$ și să efectuăm măsurări directe.

Generând în mod aleator un plan de hrube cu $n = m = 100$, ne putem convinge că timpul de execuție a programului *Hrube* nu depășește valoarea de 3,0 secunde.

Jocul

Se consideră un câmp de joc, format din n rânduri și m coloane (vezi desenul). La intersecțiile de rânduri și coloane se formează căsuțe. Fiecare căsuță este definită prin coordonatele sale: numărul de rând x și numărul de coloană y .

	1	2	3	4	5
1					
2					
3					
4					
5					
6					

Jucătorul aruncă un zar, care cade la întâmplare în una din căsuțe. În general, coordonatele acestei căsuțe nu sunt cunoscute apriori, însă ele pot fi aflate apelând, ori de câte ori este nevoie, funcția booleană `EsteInZona`.

În Pascal antetul acestei funcții are forma:

```
function EsteInZona(i, j, k, l: longint):  
Boolean;
```

iar în C/C++:

```
bool EsteInZona(long i, j, k, l)
```

unde parametrii i, j, k, l definesc o zonă dreptunghiulară a câmpului de joc, i fiind numărul rândului de sus al zonei, j – numărul rândului de jos al zonei, k – numărul coloanei din stânga a zonei, iar l – numărul coloanei din dreapta a zonei.

Funcția `EsteInZona` returnează valoarea `true` dacă zarul se află în una din căsuțele ce aparține zonei respective și `false` în caz contrar.

De exemplu, în cazul câmpului de joc de pe desen, apelul `EsteInZona(3, 6, 2, 4)` returnează valoarea `true`, iar apelul `EsteInZona(1, 4, 2, 4)` returnează valoarea `false`.

Sarcină. Elaborați un program care află coordonatele x, y ale căsuței în care se află zarul, cu condiția ca numărul de apeluri q ale funcției `EsteInZona` să fie cât mai mic posibil.

Date de intrare. Fișierul text `JOCUL.IN` conține pe o singură linie numerele întregi n și m separate prin spațiu.

Date de ieșire. Fișier de ieșire nu există. Pentru a comunica evaluatorului coordonatele x, y ale căsuței în care se află zarul, executați apelul de procedură `Raspuns(x, y)`, unde x și y sunt variabile de tipul `longint` în Pascal sau `long` în C/C++.

Notă. Funcția `EsteInZona` și procedura `Raspuns` nu trebuie definite în programul vostru. Însă, pentru a putea folosi aceste subprograme fără a avea descrierea lor, pur și simplu includeți la începutul programului vostru următoarea linie:

în Pascal:

```
uses Campul;
```

în C/C++:

```
#include "campul.h" C/C++
```

Exemplu. Pentru desenul din enunțul problemei, fișierele de intrare și ieșire sunt:

Restricții. $1 \leq n, m \leq 10^6$. Timpul de execuție nu va depăși 0,2 secunde. Programul va folosi cel mult 32 Megaocteți de memorie operativă. Fișierul sursă va avea denumirea JOCUL.PAS, JOCUL.C sau JOCUL.CPP.

Rezolvare

Vom trata câmpul de joc ca o sursă de informație, mesajele posibile ale căreia au forma (x, y) , unde x și y sunt coordonate de căsuță. Evident, numărul total al mesajelor posibile este egal cu numărul de căsuțe nm .

Cantitatea de informație într-un mesaj al acestei surse se determină conform formulei cunoscute din manualul școlar de Informatică:

$$I = \log_2 nm = \log_2 n + \log_2 m \quad (\text{biți}).$$

Este cunoscut faptul, că cantitatea de informație I reprezintă numărul minim de întrebări ce trebuie formulate pentru a identifica un mesaj concret al sursei de informați, în cazul nostru – pentru a identifica căsuța în care se află zarul. Întrebările propriu-zise se formulează prin metoda dihotomiei, împărțind consecutiv zona de căutare în jumătate.

Prin urmare, pentru a afla rândul x , în una din căsuțele căruia se află zarul, mai întâi vom efectua apelul funcției `EsteInZona` pentru jumătatea de sus a câmpului de joc și, ulterior, în dependență de răspunsul primit, vom apela din nou această funcție pentru o jumătate din zona nouă de căutare ș.a.m.d.

Într-un mod similar, pentru a afla coloana y , în una din căsuțele căruia se află zarul, mai întâi vom efectua apelul funcției `EsteInZona` pentru jumătatea din stânga a câmpului de joc și, ulterior, în dependență de răspunsul primit, vom apela din nou această funcție pentru o jumătate din zona nouă de căutare ș.a.m.d.

```

Program Jocul;
uses Campul;
{ Clasele 10-12 }
var n, m, x, y, q : longint;
procedure Citeste;
var Intrare : text;
begin
    assign(Intrare, 'JOCUL.IN');
    reset(Intrare);
    readln(Intrare, n, m);
    close(Intrare);
end; { Citeste }
procedure Scrie;
var Iesire : text;
begin
    assign(Iesire, 'JOCUL.OUT');
    rewrite(Iesire);
    writeln(Iesire, x, ' ', y, ' ', q);
    close(Iesire);
end; { Scrie }
procedure Cauta;
var i, j, k, l, c : longint;

```

```

begin
  q:=0;
  { cautam randul in care se afla zarul }
  i:=1; j:=n;
  while i<>j do
    begin
      q:=q+1;
      c:=i+((j-i) div 2); { divizam zona pe orizontala };
      if EsteInZona(i, c, 1, m) then j:=c else i:=c+1;
    end;
  x:=i;
  { cautam coloana in care se afla zarul }
  k:=1; l:=m;
  while k<>l do
    begin
      q:=q+1;
      c:=k+((l-k) div 2); { divizam zona pe verticala }
      if EsteInZona(1, n, k, c) then l:=c else k:=c+1;
    end;
  y:=k;
end; { Cauta }
begin
  Citeste;
  Cauta;
  Scrie;
end.

```

În condițiile problemei, $1 \leq n, m \leq 10^6$. Prin urmare, numărul de apeluri ale funcției EsteInZonă este egal cu cel mult $\log_2 nm = \log_2 n + \log_2 m \approx 30 + 30 = 60$, mărime cu mult mai mică decât capacitatea de prelucrare a calculatoarelor moderne.

Rădăcina numărului binar

Se consideră numărul binar natural x , format n cifre binare $\{0, 1\}$, care este, de fapt, pătratul unui alt număr binar natural y . De exemplu, numărul binar $x = 1010001$ este format din $n = 7$ cifre binare și este pătratul numărului binar $y = 1001$.

Sarcină. Elaborați un program care calculează numărul binar y , $y = \sqrt{x}$.

Date de intrare. Fișierului text RAD.IN conține pe o singură linie numărul binar x .

Date de ieșire. Fișierului text RAD.OUT va conține pe o singură linie numărul binar y , scris fără zerouri ne semnificative.

Exemplu.

RAD.IN	RAD.OUT
1010001	1001

Restricții. $1 \leq n \leq 250$. Timpul de execuție nu va depăși 1,5 secunde. Programul va folosi cel mult 32 Megaocteți de memorie operativă. Fișierul sursă va avea denumirea RAD.PAS, RAD.C sau RAD.CPP.

Rezolvare

Vom memora numărul binar x într-un șir de caractere. Evident, numărul de cifre m ale numărului binar $y = \sqrt{x}$ este de două ori mai mic decât numărul de cifre n ale numărului x sau, mai exact:

$$m = (n \text{ div } 2) + (n \text{ mod } 2).$$

Evident, prima cifră din stânga a numărului binar y are valoarea 1. Prin urmare, inițial stabilim $y = 100\dots 0$, în total m cifre binare.

Vom preciza valorile celorlalte $(m - 1)$ cifre binare ale numărului y cifra cu cifra, de la stânga la dreapta.

Presupunem, ca am calculat deja primele i cifre binare ale numărului y . Pentru a calcula următoarea cifră binară $(i + 1)$, îi atribuim ei valoarea 1 și ridicăm numărul y astfel obținut la pătrat. În continuare, sunt posibile următoarele cazuri:

- 1) $y^2 < x$ – trecem la calculul următoarei cifre binare $(i + 2)$;
- 2) $y^2 = x$ – calculele se încheie;
- 3) $y^2 > x$ – restabilim valoarea cifrei $(i + 1)$ în 0 și trecem la calculul următoarei cifre binare $(i + 2)$.

În programul ce urmează, pătratul numărului binar y este calculat cu ajutorul algoritmului de înmulțire „în coloană”.

```
Program RadacinalNumaruluiBinar;
{ Clasele 10-12 }
var A, B : string;

procedure Citeste(var X : string);
{ Citeste X din fisierul de intrare }
var Intrare : text;
```

```

begin
    assign(Intrare, 'RAD.IN');
    reset(Intrare);
    readln(Intrare, X);
    close(Intrare);
end; { Citeste }

procedure Scrie(X : string);
{ Scrie X in fisierul de iesire }
var Iesire : text;
begin
    assign(Iesire, 'RAD.OUT');
    rewrite(Iesire);
    writeln(Iesire, X);
    close(Iesire);
end; { Scrie }

procedure AdunaCifreBinare(a, b, c : char; var s, t : char);
{ Aduna cifrele binare a, b, c }
{ Returneaza suma s si transportul t }
begin
    if (a='0') and (b='0') and (c='0') then begin t:='0'; s:='0' end;
    if (a='0') and (b='0') and (c='1') then begin t:='0'; s:='1' end;
    if (a='0') and (b='1') and (c='0') then begin t:='0'; s:='1' end;
    if (a='0') and (b='1') and (c='1') then begin t:='1'; s:='0' end;
    if (a='1') and (b='0') and (c='0') then begin t:='0'; s:='1' end;
    if (a='1') and (b='0') and (c='1') then begin t:='1'; s:='0' end;
    if (a='1') and (b='1') and (c='0') then begin t:='1'; s:='0' end;
    if (a='1') and (b='1') and (c='1') then begin t:='1'; s:='1' end;
end; { AdunaCifre }

procedure AliniazaNumereBinare(var X, Y : string);
{ Aduce numerele X, Y la acelasi numar de cifre binare }
label 1;
var n, m, i : integer;
begin
    n:=length(X);
    m:=length(Y);
    if n=m then goto 1;
    if n<m then for i:=1 to m-n do X:='0'+X
                    else for i:=1 to n-m do Y:='0'+Y;
1: end; { AliniazaNumereBinare }

procedure AdunaNumereBinare(X, Y : string; var Z : string);
{ Calculeaza Z:=X+Y }
var t : char; { transportul }
    i : integer;
begin
    AliniazaNumereBinare(X, Y);
    Z:='0';
    AliniazaNumereBinare(X, Z);
    t:='0';
    for i:=length(X) downto 1 do
        AdunaCifreBinare(t, X[i], Y[i], Z[i], t);
        if t='1' then Z:='1'+Z;
    end; { AdunaNumereBinare }

procedure RidicaLaPatrat(X : string; var Z : string);
{ Calculeaza Z:=X*X }
var Y : string;
    i : integer;
begin
    Y:=X;
    Z:='0';

```

```

for i:=length(X) downto 1 do
  begin
    if X[i]='1' then AdunaNumereBinare(Y, Z, Z);
    Y:=Y+'0'; { Deplaseaza numarul Y la stanga }
  end; { for }
end; { RidicaLaPatrat }

function SuntEgale(X, Y : string) : boolean;
{ Returneaza TRUE daca X>Y si FALSE in caz contrar }
begin
  AliniazaNumereBinare(X, Y);
  SuntEgale := (X=Y);
end; { SuntEgale }

function EsteMaiMare(X, Y : string) : boolean;
{ Returneaza TRUE dac X>Y si FALSE in caz contrar }
var i : integer;
begin
  AliniazaNumereBinare(X, Y);
  EsteMaiMare := (X>Y);
end; { EsteMaiMare }

procedure ExtrageRadacina(X : string; var Y : string);
{ Calculeaza Y:=Radical(X) }
label 2;
var n, m, i : integer;
    Z : string;
begin
  n:=length(X);
  m:=(n div 2)+(n mod 2);
  Y:='1';
  for i:=2 to m do Y:=Y+'0';
  RidicaLaPatrat(Y, Z);
  if SuntEgale(Z, X) then goto 2;
  for i:=2 to m do
    begin
      Y[i]:='1';
      RidicaLaPatrat(Y, Z);
      if SuntEgale(Z, X) then goto 2;
      if EsteMaiMare(Z, X) then Y[i]:='0';
    end; { for }
2: end; { ExtrageRadacina }

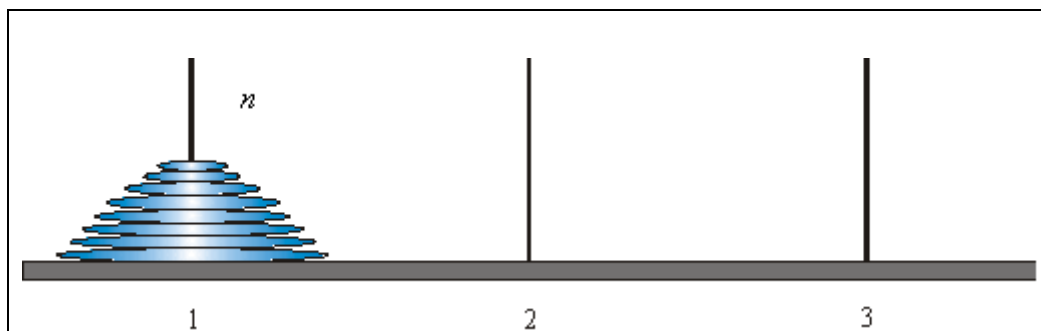
begin
  Citeste(A);
  ExtrageRadacina(A, B);
  Scrie(B);
end.

```

Din analiza textului programului RadacinalNumaruluiBinar rezultă că complexitatea temporală a algoritmului este $O(n^3)$. Conform restricțiilor problemei, $n \leq 250$. Prin urmare, timpul cerut de algoritm va fi proporțional cu 10^8 , mărime mai mică decât capacitatea de prelucrare a calculatoarelor personale din laboratoarele de informatică.

Turnuri

Se consideră trei tije notate prin 1, 2 și 3 și n discuri perforate de dimensiuni diferite (vezi desenul). Inițial toate discurile sunt pe tija 1, fiind aranjate în ordinea descrescătoare a diametrelor, considerând sensul de la bază la vârf.



Sarcină. Elaborați un program care mută toate discurile de pe tija 1 pe tija 3, folosind tija 2 ca tijă de manevră și respectând următoarele reguli:

- la fiecare pas se mută un singur disc;
- orice disc poate fi așezat doar peste un disc cu diametrul mai mare.

Date de intrare. Fișierul text `TURNURI.IN` conține pe o singură linie numărul întreg n – numărul de discuri de pe tija 1.

Date de ieșire. Fișierul text `TURNURI.OUT` va conține pe fiecare linie câte două numere întregi i și j , separate prin spațiu. Fiecare linie a fișierului de ieșire descrie câte o mutare de disc: i indică tija de pe care se ia discul, iar j – tija pe care se pune discul. Mutările apar în fișierul de ieșire în ordinea efectuării lor.

Exemplu. Pentru $n = 3$, fișierele de intrare și ieșire sunt:

TURNURI.IN	TURNURI.OUT
3	1 3
	1 2
	3 2
	1 3
	2 1
	2 3
	1 3

Restricții. $1 \leq n \leq 20$. Nu se admit mutări în care $i = j$. Timpul de execuție nu va depăși 1,0 secunde. Programul va folosi cel mult 32 Megaocteți de memorie operativă. Fișierul sursă va avea denumirea `TURNURI.PAS`, `TURNURI.C` sau `TURNURI.CPP`.

Rezolvare

Pentru a rezolva această problemă vom folosi o procedură recursivă, pe care o vom denumi-o `MutaDiscurile`. Evident, această procedură trebuie să aibă un antet de forma:

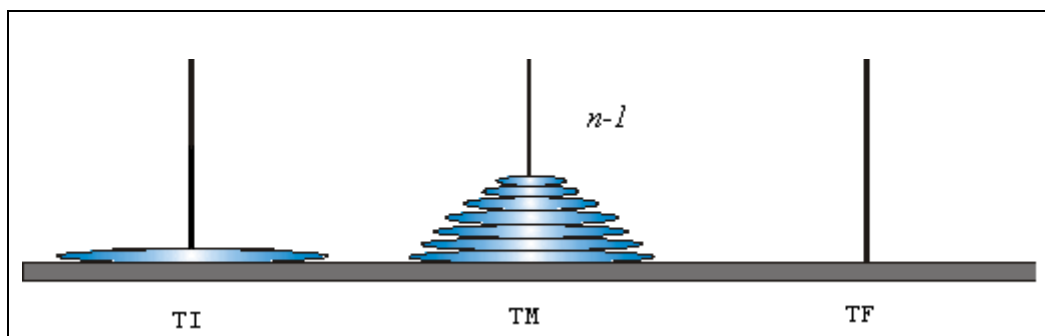
```
procedure MutaDiscurile(n, TI, TF, TM : integer);
```

unde n este numărul de discuri, TI – tija inițială, TF – tija finală, iar TM – tija de manevră.

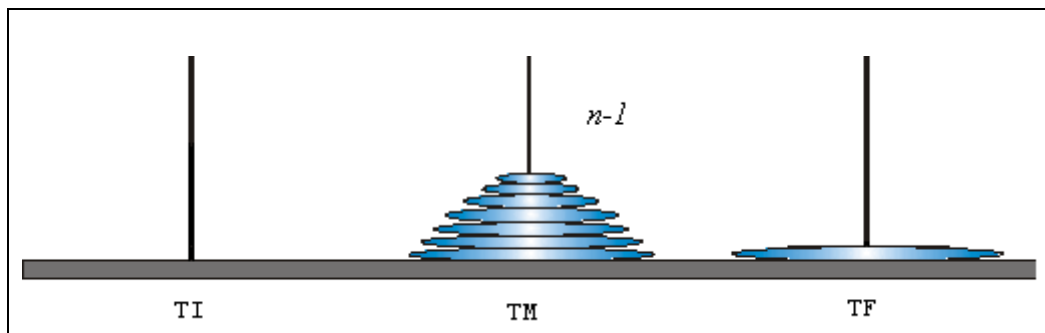
În cazul unui singur disc, adică $n = 1$, această procedură trebuie, pur și simplu, să mute discul de pe tija TI pe tija TF .

În cazurile cu mai multe discuri, adică $n > 1$, reducem problema cu n discuri la cea cu $(n - 1)$ discuri:

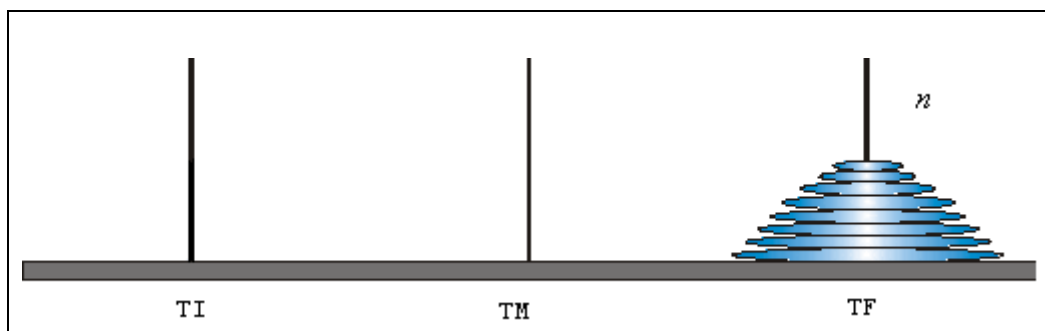
4. Apelăm procedura `MutaDiscurile` pentru a muta $n - 1$ discuri de pe tija TI pe tija TM , folosind ca tija de manevră tija TF .



5. Mutăm discul rămas de pe tija TI pe tija TF .



6. Apelăm procedura `MutaDiscurile` pentru a muta discurile de pe tija TM pe tija TF , folosind ca tija de manevră tija TI .



Prezentăm în continuare programul ce implementează procedura MutaDiscurile.

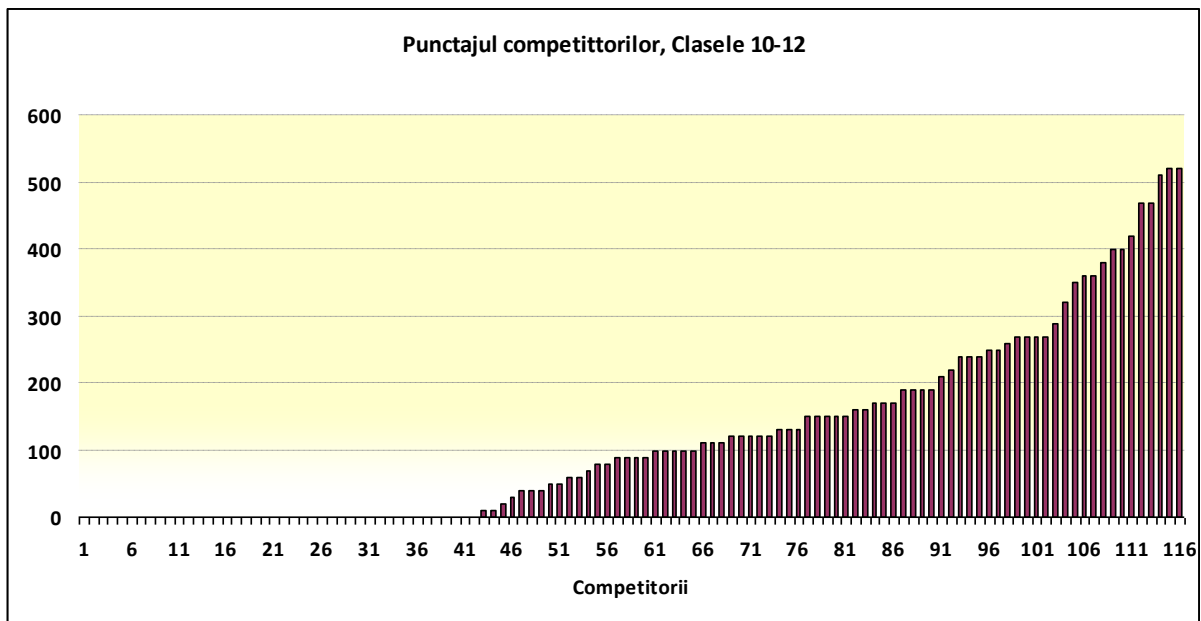
```
program Turnuri;
{ Clasele 10-12 }
var n : integer;
    Intrare, Iesire : text;

procedure MutaDiscurile(n, TI, TF, TM : integer);
begin
    if n=1 then writeln (Iesire, TI, ' ', TF)
    else
        begin
            MutaDiscurile(n-1, TI, TM, TF);
            writeln (Iesire, TI, ' ', TF);
            MutaDiscurile(n-1, TM, TF, TI);
        end; { else }
end; { MutaDiscurile }

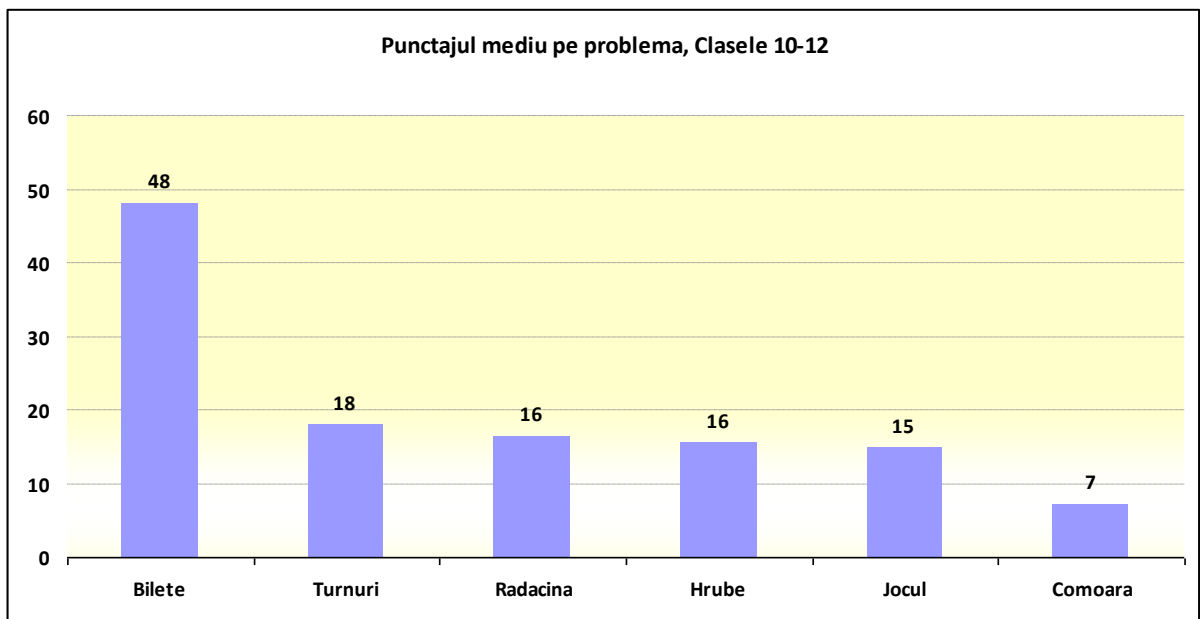
begin
    assign(Intrare, 'TURNURI.IN');
    reset(Intrare);
    readln(Intrare, n);
    close(Intrare);
    assign(Iesire, 'TURNURI.OUT');
    rewrite(Iesire);
    MutaDiscurile(n, 1, 3, 2);
    Close(Iesire);
end.
```

Prin măsurări directe de putem convinge că pentru $n = 20$ timpul de execuție a programului Turnuri nu depășește valoarea indicată în restricțiile problemei.

Punctajul competitorilor



Complexitatea problemelor



Lista premianților

Locul	Elevul	Clasa	Municipiul/Raionul	Institutia	Profesorul	Conducatorul echipei
1	Căliman Ion	12	Chișinău	Liceul Republican cu Profil Real	Butenco Inna	Minzelevschi Igor
1	Indricean Mihai	11	Chișinău	Liceul Teoretic "Orizont" (fil. Durlești)	Corlat Sergiu	Golubev Svetlana
1	Chiciuc Nicu	10	Chișinău	Liceul Teoretic „Spiru Haret”	Schițco Svetlana	Golubev Svetlana
1	Savva Dumitru	8	Chișinău	Liceul Teoretic "Orizont" (fil. Durlești)	Corlat Sergiu	Golubev Svetlana
2	Grigori Alexandru	12	Chișinău	Liceul Teoretic "Orizont" (fil. Durlești)	Corlat Sergiu	Golubev Svetlana
2	Cerceanu Cristian	11	Chișinău	Liceul Teoretic "Orizont" (fil. Durlești)	Corlat Sergiu	Golubev Svetlana
2	Ciuntu Victor	9	Chișinău	Liceul Teoretic "Prometeu-Prim"	Adam Ecaterina	Golubev Svetlana
2	Ivanov Andrei	11	Chișinău	Liceul Teoretic "Orizont" (fil. Durlești)	Corlat Sergiu	Golubev Svetlana
2	Druță Gheorghe	9	Chișinău	Liceul Teoretic "Prometeu-Prim"	Adam Ecaterina	Golubev Svetlana
2	Umbatov Ahmed	10	Chișinău	Liceul Teoretic "V. Lupu"	Movleanova Galina	Golubev Svetlana
2	Moraru Nicolai	10	Drochia	Liceul Teoretic "M. Eminescu"	Chistruga Gheorghe	Barsan Valentin
2	Pojoga Vasile	11	Chișinău	Liceul Acedemiei de Științe a Moldovei	Raisa Miron	Raisa Miron
3	Moroi Andrian	12	Soroca	Liceul Teoretic "C. Stere"	Baş Vasile	Gânga Sergiu
3	Neagaru Daniel	12	Chișinău	Liceul Acedemiei de Științe a Moldovei	Raisa Miron	Raisa Miron
3	Bicec Iuliu	12	Hîncești	Liceul Teoretic "M. Sadoveanu"	Brînză Valentina	Pojoga Maria
3	Nedelciuc Silviu	12	Bălți	Liceul Teoretic "M. Eminescu"	Curbet Gheorghe	Bîrsan Valentin
3	Griza Daniel	8	Chișinău	Liceul Teoretic "Orizont" (fil. Durlești)	Corlat Sergiu	Golubev Svetlana
3	Toloacă Ion	11	Chișinău	Liceul Teoretic „Mircea Eliade”	Anton Ghenadie	Golubev Svetlana
3	Preguza Mihai	11	Călărași	Liceul Teoretic "M. Sadoveanu"	Gavriliță Alexei	Gavriliță Alexei
3	Sîrbu Roman	10	Strășeni	Liceul Teoretic "I. Vatamanu"	Țirdea Ildia	Țirdea Ildia
3	Luncașu Victor	11	Nisporeni	Liceul Teoretic "Boris Cazacu"	Brodicico Valeriu	Brodicico Valeriu
3	Bezdrighin Marcel	7	Chișinău	Liceul Teoretic "Prometeu-Prim"	Dragan Galina	Golubev Svetlana
3	Martînin Alexandru	10	Tiraspol	Liceul Teoretic Tiraspol	Șagoian Tamara	Armaș Marina

Locul	Elevul	Clasa	Municipiul/Raionul	Institutia	Profesorul	Conducatorul echipei
3	Șapoval Nicolae	9	Chișinău	Liceul Teoretic "Orizont" (fil. Durlști)	Corlat Sergiu	Golubev Svetlana
Mentiune	Spînu Dorin	11	Fălești	Liceul Teoretic "M. Eminescu"	Mitrică Lilian	Bețșor Natalia
Mentiune	Bolboceanu Mariana	12	Leova	Liceul Teoretic "C. Spătaru"	Cernea Natalia	Carafizi Aliona
Mentiune	Dolgolev Filip	9	Tiraspol	Liceul Teoretic Tiraspol	Șagoian Tamara	Armaș Marina
Mentiune	Adascăliței Cristian	9	Drochia	Liceul Teoretic "M. Eminescu"	Chistruga Gheorghe	Chistruga Gheorghe
Mentiune	Cataraga Victor	12	Nisporeni	Liceul Teoretic "Boris Cazacu"	Brodicico Valeriu	Brodicico Valeriu
Mentiune	Marievschi Alexandru	9	Ungheni	Liceul Teoretic "A. Mateevici", Pîrlița	Ciupercă Romeo	Protasova Lucreția
Mentiune	Trandafil Nicolai	11	U.T.A.G.	Liceul Teoretic "B. Ianacoglo" Copceac	Arfanos Maria	Chilimicenco Serghei
Mentiune	Balan Denis	8	Strășeni	Liceul Teoretic. "I. Vatananu"	Gaidău Maria	Țirdea Ildia
Mentiune	Antohi Radu	12	Chișinău	Liceul Teoretic "Orizont" (fil. Durlști)	Corlat Sergiu	Golubev Svetlana
Mentiune	Romanciuc Alexandru	12	Chișinău	Liceul Teoretic "V. Alecsandri"	Vanovscaia Vera	Golubev Svetlana
Mentiune	Mogoreanu Anatolie	11	Chișinău	Liceul Teoretic "Titu Maiorescu"	Vîdiș Serghei	Golubev Svetlana
Mentiune	Gorpinevici Vlad	7	Chișinău	Liceul Teoretic "Iulia Hașdeu"	Țurcanu Ludmila	Golubev Svetlana