ALGORITMI ŞI STRUCTURI DE DATE 1

Note de Laborator

(uz intern - draft v2.3)

Adrian Răbâea

Cuprins

1 OJI 2002 clasa a IX-a **1**

1.1 Poarta 1

1.1.1 Indicaţii de rezolvare - descriere soluţie * 2

1.1.2 Rezolvare detaliată 2

1.1.3 Codul sursă * 2

1.2 Mouse 4

1.2.1 Indicaţii de rezolvare - descriere soluţie * 5

1.2.2 Rezolvare detaliată 5

1.2.3 Codul sursă * 5

2 OJI 2003 clasa a IX-a **9**

2.1 Text 9

2.1.1 Indicaţii de rezolvare - descriere soluţie * 10

2.1.2 Rezolvare detaliată * 11

2.1.3 Codul sursă * 11

2.2 Numere 13

2.2.1 Indicaţii de rezolvare - descriere soluţie * 14

2.2.2 Rezolvare detaliată 14

2.2.3 Codul sursă * 14

3 OJI 2004 clasa a IX-a **17**

3.1 Expresie 17

3.1.1 Indicaţii de rezolvare - descriere soluţie * 18

3.1.2 Rezolvare detaliată 19

3.1.3 Codul sursă * 19

3.2 Reactivi 24

3.2.1 Indicaţii de rezolvare - descriere soluţie * 25

3.2.2 Rezolvare detaliată 26

3.2.3 Codul sursă * 26

4 OJI 2005 clasa a IX-a **29**
4.1 Numere . . . 29
4.1.1 Indicaţii de rezolvare - descriere soluţie * . . . 30
4.1.2 Rezolvare detaliată . . . 30
4.1.3 Codul sursă * . . . 30
4.2 MaxD . . . 31
4.2.1 Indicaţii de rezolvare - descriere soluţie * . . . 32
4.2.2 Rezolvare detaliată . . . 33
4.2.3 Codul sursă * . . . 33

5 OJI 2006 clasa a IX-a **37**
5.1 Flori . . . 37
5.1.1 Indicaţii de rezolvare - descriere soluţie * . . . 38
5.1.2 Rezolvare detaliată . . . 39
5.1.3 Codul sursă * . . . 39
5.2 Pluton . . . 42
5.2.1 Indicaţii de rezolvare - descriere soluţie * . . . 43
5.2.2 Rezolvare detaliată . . . 44
5.2.3 Codul sursă * . . . 44

6 OJI 2007 clasa a IX-a **49**
6.1 Cartele - OJI 2007 . . . 49
6.1.1 Indicaţii de rezolvare - descriere soluţie * . . . 50
6.1.2 Rezolvare detaliată . . . 51
6.1.3 Codul sursă * . . . 51
6.2 Paritate - OJI 2007 . . . 57
6.2.1 Indicaţii de rezolvare - descriere soluţie * . . . 59
6.2.2 Rezolvare detaliată . . . 60
6.2.3 Codul sursă * . . . 60

7 ONI 2000 clasa a IX-a **63**
7.1 Algoritm . . . 63
7.1.1 Indicaţii de rezolvare - descriere soluţie * . . . 66
7.1.2 Rezolvare detaliată * . . . 66
7.1.3 Codul sursă * . . . 68
7.2 Cod de identificare . . . 70
7.2.1 Indicaţii de rezolvare - descriere soluţie * . . . 71
7.2.2 Rezolvare detaliată * . . . 71
7.2.3 Codul sursă * . . . 74
7.3 Comoara . . . 75
7.3.1 Indicaţii de rezolvare - descriere soluţie * . . . 76
7.3.2 Rezolvare detaliată . . . 77
7.3.3 Codul sursă * . . . 77
7.4 Cuburi . . . 79

7.4.1 Indicaţii de rezolvare - descriere soluţie * 81
7.4.2 Rezolvare detaliată 81
7.4.3 Codul sursă * 81
7.5 Fibo 83
7.5.1 Indicaţii de rezolvare - descriere soluţie * 84
7.5.2 Rezolvare detaliată 84
7.5.3 Codul sursă * 84
7.6 Kommando 87
7.6.1 Indicaţii de rezolvare - descriere soluţie * 89
7.6.2 Rezolvare detaliată 89
7.6.3 Codul sursă * 89

8 ONI 2001 clasa a IX-a **101**
8.1 Ferma 101
8.1.1 Indicaţii de rezolvare - descriere soluţie * 103
8.1.2 Rezolvare detaliată * 103
8.1.3 Codul sursă * 104
8.2 Fracţii 106
8.2.1 Indicaţii de rezolvare - descriere soluţie * 107
8.2.2 Rezolvare detaliată 107
8.2.3 Codul sursă * 107
8.3 Tablou 108
8.3.1 Indicaţii de rezolvare - descriere soluţie * 109
8.3.2 Rezolvare detaliată * 109
8.3.3 Codul sursă * 111
8.4 Competiţie dificilă 112
8.4.1 Indicaţii de rezolvare - descriere soluţie * 113
8.4.2 Rezolvare detaliată 113
8.4.3 Codul sursă * 113
8.5 Cuvinte 114
8.5.1 Indicaţii de rezolvare - descriere soluţie * 115
8.5.2 Rezolvare detaliată 116
8.5.3 Codul sursă * 116
8.6 Grup 118
8.6.1 Indicaţii de rezolvare - descriere soluţie * 119
8.6.2 Rezolvare detaliată 119
8.6.3 Codul sursă * 119

9 ONI 2002 clasa a IX-a **123**
9.1 Pentagon 123
9.1.1 Indicaţii de rezolvare - descriere soluţie * 124
9.1.2 Rezolvare detaliată 125
9.1.3 Codul sursă * 125
9.2 Pod 126

9.2.1 Indicaţii de rezolvare - descriere soluţie * 128
9.2.2 Rezolvare detaliată * 129
9.2.3 Codul sursă * 134
9.3 Suma 137
9.3.1 Indicaţii de rezolvare - descriere soluţie * 138
9.3.2 Rezolvare detaliată 139
9.3.3 Codul sursă * 139
9.4 Becuri 140
9.4.1 Indicaţii de rezolvare - descriere soluţie * 141
9.4.2 Rezolvare detaliată * 142
9.4.3 Codul sursă * 147
9.5 Discuri 150
9.5.1 Indicaţii de rezolvare - descriere soluţie * 151
9.5.2 Rezolvare detaliată 152
9.5.3 Codul sursă * 152
9.6 Cod 154
9.6.1 Indicaţii de rezolvare - descriere soluţie * 155
9.6.2 Rezolvare detaliată 156
9.6.3 Codul sursă * 156

10 ONI 2003 clasa a IX-a **159**
10.1 Seti 159
10.1.1 Indicaţii de rezolvare - descriere soluţie * 160
10.1.2 Rezolvare detaliată 160
10.1.3 Codul sursă * 161
10.2 Scaune 164
10.2.1 Indicaţii de rezolvare - descriere soluţie * 165
10.2.2 Rezolvare detaliată 166
10.2.3 Codul sursă * 166
10.3 Circular 170
10.3.1 Indicaţii de rezolvare - descriere soluţie * 170
10.3.2 Rezolvare detaliată 171
10.3.3 Codul sursă * 172
10.4 Criptare 174
10.4.1 Indicaţii de rezolvare - descriere soluţie * 176
10.4.2 Rezolvare detaliată 176
10.4.3 Codul sursă * 178
10.5 Maşina 179
10.5.1 Indicaţii de rezolvare - descriere soluţie * 180
10.5.2 Rezolvare detaliată 181
10.5.3 Codul sursă * 181
10.6 Operaţii 182
10.6.1 Indicaţii de rezolvare - descriere soluţie * 183
10.6.2 Rezolvare detaliată 184

10.6.3 Codul sursă * 184

11 ONI 2004 clasa a IX-a **187**
11.1 Coduri 187
11.1.1 Indicaţii de rezolvare - descriere soluţie * 188
11.1.2 Rezolvare detaliată * 188
11.1.3 Codul sursă * 189
11.2 Logic 189
11.2.1 Indicaţii de rezolvare - descriere soluţie * 191
11.2.2 Rezolvare detaliată * 191
11.2.3 Codul sursă * 195
11.3 Poligon 197
11.3.1 Indicaţii de rezolvare - descriere soluţie * 199
11.3.2 Rezolvare detaliată 199
11.3.3 Codul sursă * 199
11.4 Şablon 203
11.4.1 Indicaţii de rezolvare - descriere soluţie * 204
11.4.2 Rezolvare detaliată * 205
11.4.3 Codul sursă * 206
11.5 Şir 208
11.5.1 Indicaţii de rezolvare - descriere soluţie * 208
11.5.2 Rezolvare detaliată 209
11.5.3 Codul sursă * 209
11.6 Snipers 209
11.6.1 Indicaţii de rezolvare - descriere soluţie * 211
11.6.2 Rezolvare detaliată 211
11.6.3 Codul sursă * 211

12 ONI 2005 clasa a IX-a **215**
12.1 Bifo 215
12.1.1 Indicaţii de rezolvare - descriere soluţie * 217
12.1.2 Rezolvare detaliată 217
12.1.3 Codul sursă * 219
12.2 Romeo 225
12.2.1 Indicaţii de rezolvare - descriere soluţie * 227
12.2.2 Rezolvare detaliată * 230
12.2.3 Codul sursă * 230
12.3 Seceta 230
12.3.1 Indicaţii de rezolvare - descriere soluţie * 232
12.3.2 Rezolvare detaliată 232
12.3.3 Codul sursă * 232
12.4 Biblos 244
12.4.1 Indicaţii de rezolvare - descriere soluţie * 245
12.4.2 Rezolvare detaliată 246

12.4.3 Codul sursă * 246
12.5 Joc 249
12.5.1 Indicaţii de rezolvare - descriere soluţie * 251
12.5.2 Rezolvare detaliată 252
12.5.3 Codul sursă * 252
12.6 Pal 258
12.6.1 Indicaţii de rezolvare - descriere soluţie * 260
12.6.2 Rezolvare detaliată 260
12.6.3 Codul sursă * 260

13 ONI 2006 clasa a IX-a **265**
13.1 Factorial 265
13.1.1 Indicaţii de rezolvare - descriere soluţie * 266
13.1.2 Rezolvare detaliată 267
13.1.3 Codul sursă * 267
13.2 Limbaj 268
13.2.1 Indicaţii de rezolvare - descriere soluţie * 270
13.2.2 Rezolvare detaliată 270
13.2.3 Codul sursă * 270
13.3 Panouri 277
13.3.1 Indicaţii de rezolvare - descriere soluţie * 278
13.3.2 Rezolvare detaliată 279
13.3.3 Codul sursă * 279
13.4 Pereţi 282
13.4.1 Indicaţii de rezolvare - descriere soluţie * 284
13.4.2 Rezolvare detaliată 284
13.4.3 Codul sursă * 284
13.5 Şanţ 286
13.5.1 Indicaţii de rezolvare - descriere soluţie * 287
13.5.2 Rezolvare detaliată 288
13.5.3 Codul sursă * 288
13.6 Zumzi 290
13.6.1 Indicaţii de rezolvare - descriere soluţie * 292
13.6.2 Rezolvare detaliată 293
13.6.3 Codul sursă * 294

14 ONI 2007 clasa a IX-a **303**
14.1 Agitaţie - ONI 2007 303
14.1.1 Indicaţii de rezolvare - descriere soluţie * 305
14.1.2 Rezolvare detaliată 306
14.1.3 Codul sursă * 306
14.2 Coduri - ONI 2007 307
14.2.1 Indicaţii de rezolvare - descriere soluţie * 309
14.2.2 Rezolvare detaliată 309

14.2.3 Codul sursă * 309
14.3 Lacuri - ONI 2007 313
14.3.1 Indicaţii de rezolvare - descriere soluţie * 315
14.3.2 Rezolvare detaliată 316
14.3.3 Codul sursă *! 316
14.4 Secv - ONI 2007 319
14.4.1 Indicaţii de rezolvare - descriere soluţie * 320
14.4.2 Rezolvare detaliată 321
14.4.3 Codul sursă * 321
14.5 Şotron - ONI 2007 323
14.5.1 Indicaţii de rezolvare - descriere soluţie * 324
14.5.2 Rezolvare detaliată 325
14.5.3 Codul sursă * 325
14.6 Triunghi - ONI 2007 327
14.6.1 Indicaţii de rezolvare - descriere soluţie * 328
14.6.2 Rezolvare detaliată 329
14.6.3 Codul sursă * 329

Capitolul 1

OJI 2002 clasa a IX-a

1.1 Poarta

Se consideră harta universului ca fiind o matrice cu 250 de linii şi 250 de coloane. În fiecare celulă se găseşte o aşa numită poartă stelară, iar în anumite celule se găsesc echipaje ale porţii stelare.

La o deplasare, un echipaj se poate deplasa din locul în care se află în oricare alt loc în care se găseşte o a doua poartă, în cazul nostru în orice altă poziţie din matrice.

Nu se permite situarea simultană a mai mult de un echipaj într-o celulă. La un moment dat un singur echipaj se poate deplasa de la o poartă stelară la alta.

Cerinţă

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

Datele de intrare

Se citesc din fişierul text **poarta.in** în următorul format:

− pe prima linie numărul natural p reprezentând numărul de echipaje,

− pe următoarele p linii câte 4 numere naturale, primele două reprezentând coordonatele poziţiei iniţiale a unui echipaj (*linie coloană*), următoarele două reprezentând coordonatele poziţiei finale a aceluiaşi echipaj (*linie coloană*).

Datele de ieşire

Pe prima linie a fişierului text **poarta.out** se scrie un singur număr reprezentând numărul minim de deplasări necesar.

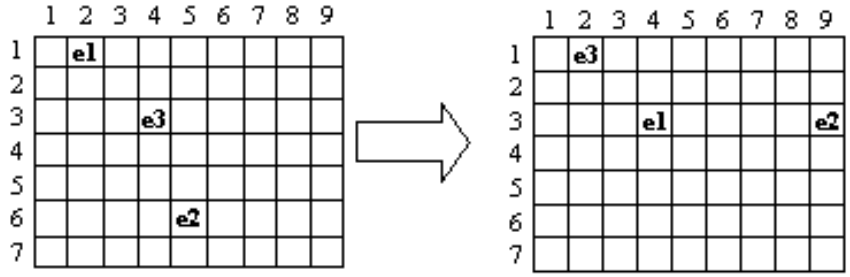
Restricţii şi precizări

− coordonatele poziţiilor iniţiale şi finale ale echipajelor sunt numere naturale din intervalul $[1, 250]$;

– poziţiile iniţiale ale celor p echipaje sunt distincte două câte două;
– poziţiile finale ale celor p echipaje sunt distincte două câte două.

Exemplu

poarta.in	poarta.out
3 1 2 3 4 6 5 3 9 3 4 1 2	4



Timp maxim de executare: 1 secundă/test

1.1.1 Indicaţii de rezolvare - descriere soluţie *

Fie `NrStationare` numărul echipajelor staţionare (care au poziţiile iniţiale şi finale egale) şi `NrCircuite` numărul circuitelor grafului orientat format astfel: nodurile sunt echipajele şi există arc de la echipajul i la echipajul j dacă şi numai dacă poziţia finală a echipajului i coincide cu poziţia iniţială a echipajului j.

Atunci `NrMinDeplasari=p+NrCircuite-NrStationare`.

1.1.2 Rezolvare detaliată

1.1.3 Codul sursă *

```
import java.io.*;
class Poarta
{
  static int p,nmd,nc=0,ns=0;
  static int[] xi,yi,xf,yf;
  static boolean[] ea; // EsteAnalizat deja
```

```
public static void main(String[] args) throws IOException
{
  StreamTokenizer st=new StreamTokenizer(
    new BufferedReader(new FileReader("poarta10.in")));

  st.nextToken(); p=(int)st.nval;
  xi=new int[p+1];
  yi=new int[p+1];
  xf=new int[p+1];
  yf=new int[p+1];
  ea=new boolean[p+1]; // implicit este false
  int i;
  for(i=1;i<=p;i++)
  {
    st.nextToken(); xi[i]=(int)st.nval;
    st.nextToken(); yi[i]=(int)st.nval;
    st.nextToken(); xf[i]=(int)st.nval;
    st.nextToken(); yf[i]=(int)st.nval;
  }

  for(i=1;i<=p;i++)
  {
    if(ea[i]) continue;
    if((xf[i]==xi[i])&&(yf[i]==yi[i])) { ea[i]=true; ns++;}
      else if(circuit(i)) nc++;
  }
  PrintWriter out=new PrintWriter(
    new BufferedWriter(new FileWriter("poarta.out")));
  nmd=p+nc-ns;
  System.out.println(p+" "+nc+" "+ns+" "+nmd);
  out.print(nmd);
  out.close();
}

static boolean circuit(int i)
{
  int j=succesor(i);
  while((j!=-1)&&(j!=i))
  {
    ea[j]=true;
    j=succesor(j);
  }
  if(j==i) return true; else return false;
}
```

```
  static int succesor(int j) // j --> k
  {
    int k;
    for(k=1;k<=p;k++)
      if((xf[j]==xi[k])&&(yf[j]==yi[k])) return k;
    return -1;
  }
}
```

1.2 Mouse

Un experiment urmăreşte comportarea unui şoricel pus într-o cutie dreptunghiulară, împărţită în $m \times n$ cămăruţe egale de formă pătrată. Fiecare cămăruţă conţine o anumită cantitate de hrană.

Şoricelul trebuie să pornească din colţul $(1, 1)$ al cutiei şi să ajungă în colţul opus, mâncând cât mai multă hrană. El poate trece dintr-o cameră în una alăturată (două camere sunt alăturate dacă au un perete comun), mănâncă toată hrana din cămăruţă atunci când intră şi nu intră niciodată într-o cameră fără hrană.

Cerinţă

Stabiliţi care este cantitatea maximă de hrană pe care o poate mânca şi traseul pe care îl poate urma pentru a culege această cantitate maximă.

Datele de intrare

Fişierul de intrare **mouse.in** conţine pe prima linie două numere m şi n reprezentând numărul de linii respectiv numărul de coloane ale cutiei, iar pe următoarele m linii cele $m \times n$ numere reprezentând cantitatea de hrană existentă în fiecare cămăruţă, câte n numere pe fiecare linie, separate prin spaţii. Toate valorile din fişier sunt numere naturale între 1 şi 100.

Datele de ieşire

În fişierul de ieşire **mouse.out** se vor scrie

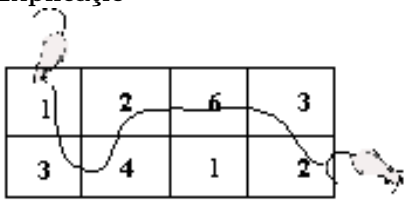
• pe prima linie două numere separate printr-un spaţiu: numărul de cămăruţe vizitate şi cantitatea de hrană maximă culeasă;

• pe următoarele linii un traseu posibil pentru cantitatea dată, sub formă de perechi de numere (linie coloană) începând cu 1 1 şi terminând cu m n.

Exemplu

mouse.in	mouse.out
2 4 1 2 6 3 3 4 1 2	7 21 1 1 2 1 2 2 1 2 1 3 1 4 2 4

Explicaţie



Timp maxim de executare: 1 secundă/test

1.2.1 Indicaţii de rezolvare - descriere soluţie *

Dacă m şi n sunt pare atunci numărul de cămăruţe vizitate este $mn - 1$ iar cantitatea de hrană maximă culeasă este suma cantităţilor de hrană din toate cămăruţele cu excepţia celei care are cea mai mică cantitate şi se află pe linia i şi coloana j şi $i + j$ este număr impar. Traseul este determinat de o parcurgere pe verticală sau orizontală şi ocolirea acelei cămăruţe.

Dacă m este impar atunci numărul de cămăruţe vizitate este mn iar cantitatea de hrană maximă culeasă este suma cantităţilor de hrană din toate cămăruţele. Traseul este determinat de o parcurgere pe orizontală.

Analog pentru situaţia în care n este impar.

1.2.2 Rezolvare detaliată

1.2.3 Codul sursă *

```
import java.io.*;
class Mouse
{
  static int m,n,imin,jmin,min,s;
  static int [][]a;
```

```
  static PrintWriter out;

  public static void main(String[] args) throws IOException
  {
    int i,j;
    StreamTokenizer st=new StreamTokenizer(
      new BufferedReader(new FileReader("mouse4.in")));
    out=new PrintWriter(new BufferedWriter(new FileWriter("mouse.out")));

    st.nextToken();m=(int)st.nval;
    st.nextToken();n=(int)st.nval;
    a=new int[m+1][n+1];

    for(i=1;i<=m;i++)
      for(j=1;j<=n;j++) {st.nextToken(); a[i][j]=(int)st.nval;}

    s=0;
    for(i=1;i<=m;i++)
      for(j=1;j<=n;j++) s=s+a[i][j];

    if(m%2==1) mimpar();
      else if(n%2==1) nimpar();
             else mnpare();
    out.close();
  }//main

  static void mimpar()
  {
    int i,j;
    out.println(m*n+" "+s);
    i=1;
    while(i+1<m)
    {
      for(j=1;j<=n;j++) out.println(i+" "+j);
      i++;
      for(j=n;j>=1;j--) out.println(i+" "+j);
      i++;
    }
    for(j=1;j<=n;j++) out.println(m+" "+j);
  }

  static void nimpar()
  {
    int i,j;
```

```
    j=1;
    out.println(m*n+" "+s);
    while(j+1<n)
    {
      for(i=1;i<=m;i++) out.println(i+" "+j);
      j++;
      for(i=m;i>=1;i--) out.println(i+" "+j);
      j++;
    }
    for(i=1;i<=m;i++) out.println(i+" "+n);
  }

  static void mnpare()
  {
    int i,j;
    imin=0;jmin=0;min=101;
    for(i=1;i<=m;i++)
      for(j=1;j<=n;j++)
        if((i+j)%2==1)
          if(a[i][j]<min) { min=a[i][j]; imin=i; jmin=j; }
    out.println((m*n-1)+" "+(s-a[imin][jmin]));

    j=1;
    while(j+1<jmin) // stanga
    {
      for(i=1;i<=m;i++) out.println(i+" "+j);
      j++;
      for(i=m;i>=1;i--) out.println(i+" "+j);
      j++;
    }

    i=1;
    while(i+1<imin) // sus
    {
      out.println(i+" " +j);
      out.println(i+" " +(j+1));
      out.println((i+1)+" " +(j+1));
      out.println((i+1)+" " +j);
      i=i+2;
    }

    out.println(i+" "+j); // patratel
    if((i==imin)&&(j+1==jmin)) out.println((i+1)+" " +j);
      else  out.println(i+" " +(j+1));
```

```
    out.println((i+1)+" " +(j+1));

    i=i+2;
    while (i<m) // jos
    {
      out.println(i+" " +(j+1));
      out.println(i+" " +j);
      out.println((i+1)+" " +j);
      out.println((i+1)+" " +(j+1));
      i=i+2;
    }

    j=j+2;
    while(j+1<=n) // dreapta
    {
      for(i=m;i>=1;i--) out.println(i+" "+j);
      j++;
      for(i=1;i<=m;i++) out.println(i+" "+j);
      j++;
    }
  }
}//class
```

Capitolul 2

OJI 2003 clasa a IX-a

2.1 Text

Vasile lucrează intens la un editor de texte. Un text este format din unul sau mai multe paragrafe. Orice paragraf se termină cu Enter şi oricare două cuvinte consecutive din acelaşi paragraf sunt separate prin spaţii (unul sau mai multe). În funcţie de modul de setare a paginii, numărul maxim de caractere care încap în pagină pe o linie este unic determinat (**Max**).

Funcţia pe care Vasile trebuie să o implementeze acum este alinierea în pagină a fiecărui paragraf din text la stânga şi la dreapta. Pentru aceasta el va trebui să împartă fiecare paragraf în linii separate de lungime **Max** (fiecare linie terminată cu Enter).

Împărţirea se realizează punând numărul maxim posibil de cuvinte pe fiecare linie, fără împărţirea cuvintelor în silabe.

Pentru aliniere stânga-dreapta, Vasile trebuie să repartizeze spaţii în mod **uniform** între cuvintele de pe fiecare linie, astfel încât ultimul caracter de pe linie să fie diferit de spaţiu, iar numărul total de caractere de pe linie să fie egal cu **Max**. Excepţie face numai ultima linie din paragraf, care rămâne aliniată la stânga (cuvintele fiind separate printr-un singur spaţiu, chiar dacă linia nu este plină).

În general, este puţin probabil ca alinierea să fie realizabilă prin plasarea aceluiaşi număr de spaţii între oricare două cuvinte consecutive de pe linie. Vasile consideră că este mai elegant ca, dacă între unele cuvinte consecutive trebuie plasat un spaţiu în plus faţă de alte perechi de cuvinte consecutive, acestea să fie plasate la începutul liniei.

Cerinţă

Scrieţi un program care să citească lungimea unei linii şi textul dat şi care să alinieze textul la stânga şi la dreapta.

Date de intrare

Fişierul de intrare **text.in** conţine pe prima linie **Max**, lungimea maximă a unui rând. Pe următoarele linii este scris textul.

Date de ieşire

Fişierul de ieşire **text.out** conţine textul aliniat stânga-dreapta.

Restricţii şi precizări

• $2 \leq$ **Max** ≤ 1000.

• Lungimea maximă a oricărui cuvânt din text este 25 caractere şi nu depăşeşte **Max**.

• Lungimea unui paragraf nu depăşeşte 1000 de caractere.

• Soluţia este unică.

Exemple

text.in	**text.out**
20 Vasile are multe bomboane bune.	Vasile are multe bomboane bune.

Explicaţie

Pe prima linie au fost plasate câte 3 spaţii între cuvintele consecutive.

text.in	**text.out**
20 Ana are mere. Ion are multe pere galbene?	Ana are mere. Ion are multe pere galbene?

Explicaţie

Între *Ion* şi *are* există 2 spaţii, între *are* şi *multe* - 2 spaţii, iar între *multe* şi *pere* - 1 spaţiu.

Observaţi că paragraful *Ana are mere.* (care are lungimea mai mică decât 20) a rămas aliniat la stânga, iar ultima linie din fiecare paragraf rămâne aliniată la stânga, cuvintele consecutive fiind separate printr-un singur spaţiu.

Timp maxim de executare: 1 secundă/test.

2.1.1 Indicaţii de rezolvare - descriere soluţie *

Fiecare paragraf se preia într-un vector de string-uri, elementele vectorului conţinând cuvintele din paragraf. Se parcurge acest vector, începând cu prima poziţie, determinând cel mai mare indice i_1 care permite plasarea cuvintelor de pe poziţiile $1, ..., i_1$ pe acelaşi rând. Se destribuie spaţiile disponibile, conform cerinţei problemei şi se afişează această linie. Se continuă prelucrarea vectorului începând cu poziţia $i_1 + 1$, şi aşa mai departe!

2.1.2 Rezolvare detaliată *

```
21
E cerul sus
Ca niste-ntinse brate
N-au crengile de ce sa se agate
```

```
0000:0000  32 31 0D 0A 45 20 63 65 72 75 6C 20 73 75 73 0D
0000:0010  0A 43 61 20 6E 69 73 74 65 2D 6E 74 69 6E 73 65
0000:0020  20 62 72 61 74 65 0D 0A 4E 2D 61 75 20 63 72 65
0000:0030  6E 67 69 6C 65 20 64 65 20 63 65 20 73 61 20 73
0000:0040  65 20 61 67 61 74 65 0D 0A 1A
```

Sfârşitul de fişier (EOF) este marcat prin `1A`.
Sfârşitul de linie (EOL) este marcat prin `0D0A`.

2.1.3 Codul sursă *

```
import java.io.*;
class Text
{
  static int Max, nc;
  static String[] s=new String[501]; // cuvintele din paragraf
  static int[] lgc=new int[501]; // numarul caracterelor cuvintelor
  static int[] nsp=new int[501]; // numarul spatiilor dupa cuvant
  static int cs=0, cd=0; // cs=cuvant stanga, cd=cuvant dreapta
  static int lglin=0;
  static PrintWriter out;

  public static void main(String[] args) throws IOException
  {
    out=new PrintWriter(new BufferedWriter(new FileWriter("text.out")));
    StreamTokenizer st=new StreamTokenizer(
                       new BufferedReader(new FileReader("text.in")));
    st.eolIsSignificant(true);

    st.nextToken(); Max=(int)st.nval;
    st.nextToken(); // preia EOL-ul existent dupa Max
    while(st.nextToken()!=StreamTokenizer.TT_EOF)
    {
      nc=0;
```

```
      do
      {
        s[++nc]=st.sval.toString();
      } while(st.nextToken()!=StreamTokenizer.TT_EOL);
      rezolva();
    }
    out.close();
  }

  static void rezolva() throws IOException
  {
    cs=0; // primul cuvant din linie  (din stanga)
    cd=0; // ultimul cuvant din linie (din stanga)
    while(cd<nc)
    {
      cs=cd+1;
      cd=cs;
      lglin=s[cs].length();
      while((cd+1<=nc)&&(lglin+1+s[cd+1].length()<=Max))
      {
        cd++;
        lglin=lglin+1+s[cd].length();
      }
      if(cd<nc) unRand(); else ultimulRand();
    }
  }

  static void unRand() throws IOException
  {
    int i,j;
    int ntsr; // ntsr=nr total de spatii ramase de distribuit
    int nsr; // nsr=nr spatii ramase de distribuit pentru primele cuvinte
    int nsf; // nr spatii de adaugat dupa fiecare cuvant cs ... cd-1

    ntsr=Max-lglin;
    if(cs!=cd)
    {
      nsf=ntsr/(cd-cs);
      nsr=ntsr%(cd-cs);
      for(i=cs;i<cd;i++) nsp[i]=1+nsf;
      for(i=1;i<=nsr;i++) nsp[cs+i-1]++;
      for(i=cs;i<=cd-1;i++)
      {
        out.print(s[i]);
```

```
        for(j=1;j<=nsp[i];j++) out.print(" ");
      }
      out.println(s[cd]);
    }
  }

  static void ultimulRand() throws IOException
  {
    int i;
    out.print(s[cs]);
    for(i=cs+1;i<=cd;i++) out.print(" "+s[i]);
    out.println();
  }
}
```

2.2 Numere

Gigel este un mare pasionat al cifrelor. Orice moment liber şi-l petrece jucându-se cu numere.

Jucându-se astfel, într-o zi a scris pe hârtie 10 numere distincte de câte două cifre şi a observat că printre acestea există două submulţimi disjuncte de sumă egală.

Desigur, Gigel a crezut că este o întâmplare şi a scris alte 10 numere distincte de câte două cifre şi spre surpriza lui, după un timp a găsit din nou două submulţimi disjuncte de sumă egală.

Cerinţă

Date 10 numere distincte de câte două cifre, determinaţi numărul de perechi de submulţimi **disjuncte** de sumă egală care se pot forma cu numere din cele date, precum şi una dintre aceste perechi pentru care suma numerelor din fiecare dintre cele două submulţimi este maximă.

Date de intrare

Fişierul de intrare **numere.in** conţine pe prima linie 10 numere naturale distincte separate prin câte un spaţiu x_1 x_2 ... x_{10}.

Date de ieşire

Fişierul de ieşire **numere.out** conţine trei linii. Pe prima linie se află numărul de perechi de submulţimi de sumă egală, precum şi suma maximă obţinută, separate printr-un spaţiu. Pe linia a doua se află elementele primei submulţimi, iar pe linia a treia se află elementele celei de a doua submulţimi, separate prin câte un spaţiu.

NrSol Smax	**NrSol** - numărul de perechi; **Smax** - suma maximă
x_1 ... x_k	elementele primei submulţimi
y_1 ... y_p	elementele celei de a doua submulţimi

Restricţii şi precizări

- $10 \leq x_i, y_i \leq 99$, pentru $1 \leq i \leq 10$.
- $1 \leq k, p \leq 9$.
- Ordinea submulţimilor în perechi nu contează.
- Perechea de submulţimi determinată nu este obligatoriu unică.

Exemplu

numere.in	**numere.out**
60 49 86 78 23 97 69 71 32 10	130 276 78 97 69 32 60 49 86 71 10

Explicaţie:
130 de soluţii; suma maximă este 276; s-au folosit 9 din cele 10 numere; prima submulţime are 4 elemente, a doua are 5 elemente.

Timp maxim de executare: 1 secundă/test

2.2.1 Indicaţii de rezolvare - descriere soluţie *

Numărul mic al numerelor (numai 10 numere distincte) permite generarea tuturor submulţimilor, verificarea condiţiilor din problemă pentru fiecare pereche de submulţimi şi determinarea informaţiilor cerute.

2.2.2 Rezolvare detaliată

2.2.3 Codul sursă *

```
import java.io.*;
class Numere
{
  static int[] x=new int[10];
  static PrintWriter out;

  public static void main(String[] args) throws IOException
  {
    int i, ia, ib, nsol=0, smax=-1, iamax=123,ibmax=123, sumaia=-1;
    long t1,t2;
```

```
    t1=System.currentTimeMillis();
    StreamTokenizer st=new StreamTokenizer(
                        new BufferedReader(new FileReader("numere.in")));
    out=new PrintWriter(new BufferedWriter(new FileWriter("numere.out")));
    for(i=0;i<10;i++) {st.nextToken(); x[i]=(int)st.nval;}
    for(ia=1;ia<=1022;ia++)
    for(ib=ia+1;ib<=1022;ib++) // fara ordine intre A si B
    if((ia&ib)==0)
    {
      sumaia=suma(ia);
      if(sumaia==suma(ib))
      {
        nsol++;
        if(sumaia>smax)
        {
          smax=sumaia;
          iamax=ia;
          ibmax=ib;
        }
      }
    }
    out.println(nsol+" "+smax);
    afis(iamax);
    afis(ibmax);
    out.close();
    t2=System.currentTimeMillis();
    System.out.println(t2-t1);
  }// main

  static int suma(int i)
  {
    int s=0,k=0;
    for(k=0;k<=9;k++) if( (i&(1<<k)) != 0 ) s+=x[k];
    return s;
  }

  static void afis(int i)
  {
    int k=0;
    while(i!=0)
    {
      if(i%2!=0) out.print(x[k]+" ");
      k++;
      i/=2;
```

```
    }
    out.println();
  }
}// class
```

Capitolul 3

OJI 2004 clasa a IX-a

3.1 Expresie

Se dă un şir de n numere naturale nenule $x_1, x_2, ..., x_n$ şi un număr natural m.

Cerinţă

Să se verifice dacă valoarea expresiei $\sqrt[m]{x_1...x_n}$ exte un număr natural.

În caz afirmativ să se afişeze acest număr descompus în factori primi.

Date de intrare

În fişierul **exp.in** se află pe prima linie m, pe linia a doua n, iar pe linia a treia numerele $x_1, x_2, ..., x_n$ separate între ele prin câte un spaţiu.

Date de ieşire

În fişierul **exp.out** se va scrie pe prima linie cifra 0, dacă valoarea expresiei nu este un număr natural, respectiv 1 dacă este un număr natural. Dacă valoarea expresiei este un număr natural pe următoarele linii se vor scrie perechi de forma p e (p este factor prim care apare în descompunere la puterea $e \geq 1$). Aceste perechi se vor scrie în ordine crescătoare după primul număr (adică p).

Restricţii şi precizări

- n - număr natural nenul < 5000
- x_i - număr natural nenul < 30000, $i \in 1, 2, ..., n$
- m - poate fi una din cifrele 2, 3, 4

Exemple

exp.in	exp.out	exp.in	exp.out
2 4 32 81 100 19	0	2 4 32 81 100 18	1 2 4 3 3 5 1

Timp maxim de execuţie: 1 secundă/test

3.1.1 Indicaţii de rezolvare - descriere soluţie *

Soluţie prezentată de Mihai Stroe, GInfo nr. 14/4

Rezolvarea problemei constă în descompunerea în factori primi a numerelor date şi prelucrarea acestor factori.

Concret, se poate păstra un vector P de la 1 la 30000, în care P_i este 0 dacă i nu este prim, respectiv puterea la care apare i în descompunerea în factori primi a produsului dacă i este prim.

Valorile din P nu pot depăşi $n \cdot log_2 30000$, deci vor fi mai mici sau egale cu $5000 \cdot 14 = 70.000$; astfel, elementele vectorului vor fi de tip *longint*.

Fiecare din cele n numere este descompus în factori primi. În momentul în care se determină un factor prim F al unui număr din cele date, se incrementează P_F cu puterea la care F apare în descompunerea numărului respectiv. Nu este necesară memorarea separată a descompunerii fiecărui număr.

În final, pentru fiecare element din P se verifică dacă este multiplu de m.

Dacă toate elementele îndeplinesc această condiţie, expresia este un număr natural şi se trece la afişare.

Se poate renunţa la vectorul de 30000 de elemente, păstrându-se în locul acestuia un vector în care se memorează numerele prime mai mici decât 30000 şi un vector care arată la ce puteri apar aceste numere în descompunere. Această abordare introduce în plus operaţii pentru a găsi indicele unui anumit număr prim; se poate folosi cu succes căutarea binară. Pe de altă parte, la descompunerea în factori primi se va testa numai împărţirea prin numere prime.

Analiza complexităţii

Descompunerea unui număr x în factori primi are ordinul de complexitate $O(\sqrt{x})$, dacă nu se foloseşte lista de numere prime.

Pasul de descompunere şi completare a vectorului P are deci ordinul de complexitate $O(n \cdot \sqrt{30000})$.

Citirea datelor, verificarea dacă expresia este un număr natural şi afişarea au ordinul de complexitate $O(n)$.

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(n)$, dacă facem abstracţie de constanta $\sqrt{30000}$.

3.1.2 Rezolvare detaliată

3.1.3 Codul sursă *

Prima variantă:

```
import java.io.*;
class Expresie1
{
  static int[] p=new int[30000];
  static int m,n;

  public static void main(String[] args) throws IOException
  {
    int i;
    StreamTokenizer st=new StreamTokenizer(
          new BufferedReader(new FileReader("expresie.in")));
    st.nextToken(); m=(int)st.nval;
    st.nextToken(); n=(int)st.nval;
    int[] x=new int[n+1];
    for(i=1;i<=n;i++) { st.nextToken(); x[i]=(int)st.nval; }
    for(i=1;i<=n;i++) descfact(x[i]);
    int ok=1;
    for (i=2;i<30000;i++)
    if (p[i]%m==0) p[i]=p[i]/m;
      else { ok=0; break; }
    PrintWriter out=new PrintWriter(
         new BufferedWriter(new FileWriter("expresie.out")));
    if(ok==0) out.println(0);
    else
    {
      out.println(1);
      for(i=2;i<30000;i++)
        if(p[i]!=0) out.println(i+" "+p[i]);
    }
    out.close();
  }

  static void descfact(int nr)
  {
    int d=2;
    while(nr%d==0)
```

```
    {
      p[d]++;
      nr=nr/d;
    }
    d=3;
    while(d<=nr)
    {
      while(nr%d==0) { p[d]++; nr=nr/d; }
      d=d+2;
    }
  }
}
```

A doua variantă:

```
import java.io.*;
class Expresie2 // pentru valori < 30.000 sunt 3245 prime 2...29.989
{ // sunt ~10% numere prime (+/- 5%)
  static final int valmax=30000; // valoare maxima pentru x_i
  static int m,n,nnp; // nnp=nr numere prime < valmax
  static int[] x;
  static int[] p=new int[3246]; // numere prime
  static int[] e=new int[3246]; // exponenti corespunzatori
  static boolean ok;

  public static void main(String[] args) throws IOException
  {
    int i,j;
    StreamTokenizer st=new StreamTokenizer(
                       new BufferedReader(
                       new FileReader("expresie.in")));
    PrintWriter out=new PrintWriter(
                    new BufferedWriter(
                    new FileWriter("expresie.out")));
    st.nextToken(); m=(int)st.nval;
    st.nextToken(); n=(int)st.nval;
    x=new int[n+1];
    for(i=1;i<=n;i++) { st.nextToken(); x[i]=(int)st.nval; }
    sol();
    if(!ok) out.println(0);
    else
    {
      out.println(1);
      for(i=1;i<=nnp;i++)
        if(e[i]>0) out.println(p[i]+" "+e[i]);
```

```
    }
    out.close();
  }// main

  static void sol()
  {
    int i,j;
    prime(); // afisv(p);
    for(i=1;i<=nnp;i++) e[i]=0; // era initializat implicit !!!
    for(i=1;i<=n;i++)
    {
      j=1; // pozitie in lista numerelor prime p[]
      while(x[i]!=1)
      {
        while(x[i]%p[j]==0) { e[j]++; x[i]/=p[j]; }
        j++;
      }
    }
    ok=true;
    for(i=1;i<=nnp;i++)
      if(e[i]%m==0) e[i]=e[i]/m; else {ok=false; break;}
  }

  static void prime()
  {
    int i,j;
    p[1]=2; p[2]=3; nnp=2;
    i=5;
    while(i<valmax)
    {
      if(estePrim(i)) p[++nnp]=i;
      i+=2;
    }
  }

  static boolean estePrim(int nr) // folosind lista numerelor prime !
  {
    int i=1;
    while((p[i]*p[i]<nr)&&(nr%p[i]!=0)) i++;
    if(p[i]*p[i]>nr) return true; return false;
  }
}
```

A treia variantă:

```
import java.io.*;
class Expresie3            // pentru "peste 5.000 de factori"
{
  static int m,n,nf=0,nfc;
  static int[] x;
  static int[] f={}, e={};
  static int[] fc=new int[6]; // 2*3*5*7*11*13=30030 > 30000 pentru x[i]
  static int[] ec=new int[6]; // exponenti curenti corespunzatori
  static boolean ok;

  public static void main(String[] args) throws IOException
  {
    int i,j;
    StreamTokenizer st=new StreamTokenizer(
                       new BufferedReader(
                       new FileReader("Expresie.in")));
    PrintWriter out=new PrintWriter(
                    new BufferedWriter(
                    new FileWriter("Expresie.out")));
    st.nextToken(); m=(int)st.nval;
    st.nextToken(); n=(int)st.nval;
    x=new int[n];
    for(i=0;i<n;i++) { st.nextToken(); x[i]=(int)st.nval; }
    sol();
    if(!ok) out.println(0);
    else
    {
      out.println(1);
      for(i=0;i<nf;i++) out.println(f[i]+" "+e[i]);
    }
    out.close();
  }// main

  static void sol()
  {
    int i;
    for(i=0;i<n;i++)
    {
      if(x[i]==1) continue;
      descfact(x[i]); interclasare();
    }
    ok=true;
    for(i=0;i<nf;i++)
    if(e[i]%m==0) e[i]=e[i]/m; else {ok=false; break;}
```

```
}

static void interclasare() // (f cu ff) SI (e cu ee)
{
  int i;
  if(nf==0)
  {
    int[] ff=new int[nfc], ee=new int[nfc];
    for(i=0;i<nfc;i++) {ff[i]=fc[i]; ee[i]=ec[i];}
    f=ff; e=ee; nf=nfc; return;
  }
  int j, nft=nf+nfc,n;
  int[] ff=new int[nft], ee=new int[nft];
  i=0; j=0; n=0; // primul indice in care incarc este 0=zero !!!
  while((i<nf)&&(j<nfc))
  {
    n++;
    if(f[i]<fc[j])
    {
      ff[n-1]=f[i];
      ee[n-1]=e[i];
      i++;
    }
    else if(f[i]>fc[j])
         {
           ff[n-1]=fc[j];
           ee[n-1]=ec[j];
           j++;
         }
         else
         {
           ff[n-1]=f[i];
           ee[n-1]=e[i]+ec[j];
           i++; j++;
         }
  }
  while(i<nf) {n++; ff[n-1]=f[i]; ee[n-1]=e[i];  i++;}
  while(j<nfc) {n++; ff[n-1]=fc[j]; ee[n-1]=ec[j]; j++;}
  if(n==nft) {f=ff; e=ee; nf=n;}
  else
  {
    int[] fff=new int[n], eee=new int[n];
    for(i=0;i<n;i++) {fff[i]=ff[i]; eee[i]=ee[i];}
    f=fff; e=eee; nf=n;}
```

```
  }

  static void descfact(int nr)
  {
    // if((nr==0)||(nr==1)) return nr;
    nfc=0;
    int d=2;
    if(nr%d==0)
    {
      nfc++; fc[nfc-1]=d; ec[nfc-1]=0;
      while(nr%d==0) {ec[nfc-1]++; nr=nr/d;}}
      d=3;
      while((d*d<=nr)&&(nr!=1))
      {
        if(nr%d==0)
        {
          nfc++;
          fc[nfc-1]=d;
          ec[nfc-1]=0;
          while(nr%d==0) {ec[nfc-1]++; nr=nr/d;}
        }
        d=d+2;
      }
      if(nr!=1) {nfc++; fc[nfc-1]=nr; ec[nfc-1]=1;}
  }// descfact
}//class
```

3.2 Reactivi

Într-un laborator de analize chimice se utilizează N reactivi.

Se ştie că, pentru a evita accidentele sau deprecierea reactivilor, aceştia trebuie să fie stocaţi în condiţii de mediu speciale. Mai exact, pentru fiecare reactiv x, se precizează intervalul de temperatură $[min_x, max_x]$ în care trebuie să se încadreze temperatura de stocare a acestuia.

Reactivii vor fi plasaţi în frigidere.

Orice frigider are un dispozitiv cu ajutorul căruia putem stabili temperatura (constantă) care va fi în interiorul acelui frigider (exprimată într-un număr întreg de grade Celsius).

Cerinţă

Scrieţi un program care să determine numărul minim de frigidere necesare pentru stocarea reactivilor chimici.

Date de intrare

Fişierul de intrare **react.in** conţine:

– pe prima linie numărul natural N, care reprezintă numărul de reactivi;

– pe fiecare dintre următoarele N linii se află $min\ max$ (două numere întregi separate printr-un spaţiu); numerele de pe linia $x + 1$ reprezintă temperatura minimă, respectiv temperatura maximă de stocare a reactivului x.

Date de ieşire

Fişierul de ieşire **react.out** va conţine o singură linie pe care este scris numărul minim de frigidere necesar.

Restricţii şi precizări

• $1 \leq N \leq 8000$

• $-100 \leq min_x \leq max_x \leq 100$ (numere întregi, reprezentând grade Celsius), pentru orice x de la 1 la N

• un frigider poate conţine un număr nelimitat de reactivi

Exemple

react.in	react.out	react.in	react.out	react.in	react.out
3 -10 10 - 2 5 20 50	2	4 2 5 5 7 10 20 30 40	3	5 -10 10 10 12 -20 10 7 10 7 8	2

Timp maxim de execuţie: 1 secundă/test

3.2.1 Indicaţii de rezolvare - descriere soluţie *

Soluţie prezentată de Mihai Stroe, GInfo nr. 14/4

Problema se poate rezolva prin metoda *greedy*.

O variantă mai explicită a enunţului este următoarea:

"Se consideră N intervale pe o axă. Să se aleagă un număr minim de puncte astfel încât fiecare interval să conţină cel puţin unul dintre punctele alese."

Facem o primă observaţie: pentru orice soluţie optimă există o soluţie cu acelaşi număr de puncte (frigidere), în care fiecare punct să fie sfârşitul unui interval. Aceasta se poate obţine mutând fiecare punct spre dreapta, până când ar ajunge la sfârşitul intervalului care se termină cel mai repede, dintre intervalele care îl conţin. Se observă că noua soluţie respectă restricţiile din enunţ.

În continuare ne vom concentra pe găsirea unei soluţii de acest tip.

Sortăm reactivii după sfârşitul intervalului. Pentru intervalul care se termină cel mai repede, alegem ultimul punct al său ca temperatură a unui frigider. Se observă că această alegere este cea mai bună, dintre toate alegerile unui punct

în intervalul respectiv, în sensul că mulţimea intervalelor care conţin punctul este mai mare (conform relaţiei de incluziune), decât mulţimea corespunzătoare oricărei alte alegeri. Acest fapt este adevărat, deoarece mutarea punctului mai la stânga nu duce la selectarea unor noi intervale.

Intervalele care conţin punctul respectiv sunt eliminate (reactivii corespunzători pot fi plasaţi într-un frigider), iar procesul continuă cu intervalele rămase, în acelaşi mod.

Analiza complexităţii

Notăm cu D numărul de temperaturi întregi din intervalul care conţine temperaturile din enunţ. Se observă că D este cel mult 201.

Citirea datelor de intrare are ordinul de complexitate $O(N)$.

Sortarea intervalelor după capătul din dreapta are ordinul de complexitate $O(N \cdot logN)$.

Urmează F paşi, unde F este numărul de frigidere selectate. Deoarece fiecare frigider este setat la o temperatură întreagă, $F \leq D$.

În cadrul unui pas, determinarea intervalului care se termină cel mai repede, pe baza vectorului sortat, are ordinul de complexitate $O(1)$. Eliminarea intervalelor care conţin un anumit punct (sfârşitul intervalului care se termină cel mai repede) are ordinul de complexitate $O(N)$.

Afişarea rezultatului are ordinul de complexitate $O(1)$.

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(N \cdot D + N \cdot logN)$; deoarece în general $D > logN$, considerăm ordinul de complexitate ca fiind $O(N \cdot D)$.

3.2.2 Rezolvare detaliată

3.2.3 Codul sursă *

```
import java.io.*;
class Reactivi
{
  static int n;        // n=nr reactivi
  static int ni=0;     // ni=nr interschimbari in quickSort
  static int nf=0;     // n=nr frigidere
  static int[] ngf;    // ng=nr grade in frigider
  static int[] x1,x2;  // limite inferioara/superioara

  public static void main(String[] args) throws IOException
  {
    int i,j;
```

```
    StreamTokenizer st=new StreamTokenizer(
          new BufferedReader(new FileReader("Reactivi.in")));
    PrintWriter out=new PrintWriter(
         new BufferedWriter(new FileWriter("Reactivi.out")));
    st.nextToken(); n=(int)st.nval;
    x1=new int[n+1];
    x2=new int[n+1];
    ngf=new int[n+1];
    for(i=1;i<=n;i++)
    {
      st.nextToken(); x1[i]=(int)st.nval;
      st.nextToken(); x2[i]=(int)st.nval;
    }
    sol();
    out.println(nf);
    out.close();
  }// main

  static void sol()
  {
    int i;
    quickSort(1,n);
    i=1; nf=1; ngf[nf]=x2[i];
    i++;
    while(i<n)
    {
      while((i<=n)&&(x1[i]<=ngf[nf])) i++;
      if(i<n) ngf[++nf]=x2[i++];
    }
  }

  static void quickSort(int p, int u)
  {
    int i,j,aux;
    i=p; j=u;
    while(i<j)
    {
      while((i<j)&&((x2[i]<x2[j])||
           ((x2[i]==x2[j])&&(x1[i]>=x1[j])))) i++;
      if(i!=j)
      {
        aux=x1[i]; x1[i]=x1[j]; x1[j]=aux;
        aux=x2[i]; x2[i]=x2[j]; x2[j]=aux;
      }
```

```
      while((i<j)&&((x2[i]<x2[j])||
           ((x2[i]==x2[j])&&(x1[i]>=x1[j])))) j--;
      if(i!=j)
      {
        aux=x1[i]; x1[i]=x1[j]; x1[j]=aux;
        aux=x2[i]; x2[i]=x2[j]; x2[j]=aux;
      }
    }
    if(p<i-1) quickSort(p,i-1);
    if(i+1<u) quickSort(i+1,u);
  }
}
```

Capitolul 4

OJI 2005 clasa a IX-a

4.1 Numere

Doru Popescu Anastasiu

Mircea este pasionat de programare. El a început să rezolve probleme din ce în ce mai grele. Astfel a ajuns la o problemă, care are ca date de intrare un tablou pătratic cu n linii şi n coloane, componente tabloului fiind toate numerele naturale distincte de la 1 la n^2. Pentru a verifica programul pe care l-a scris îi trebuie un fişier care să conţină tabloul respectiv. După ce a creat acest fişier, fratele său, pus pe şotii îi umblă în fişier şi îi schimbă câteva numere consecutive, cu numărul 0. Când se întoarce Mircea de la joacă constată cu stupoare că nu îi merge programul pentru testul respectiv. După câteva ore de depanare îşi dă seama că programul lui este corect şi că fişierul de intrare are probleme.

Cerinţă

Scrieţi un program care să-l ajute pe Mircea, găsindu-i cel mai mic şi cel mai mare dintre numerele consecutive schimbate de fratele său.

Date de intrare

În fişierul **numere.in** se dă pe prima linie n, iar pe următoarele n linii elementele tabloului, câte n elemente pe o linie, separate între ele prin câte un spaţiu, după modificările făcute de fratele lui Mircea.

Date de ieşire

În fişierul **numere.out** se va scrie pe un singur rând cu un singur spaţiu între ele numerele cerute (primul fiind cel mai mic).

Restricţii şi precizări

• $0 < n \leq 500$.

• Fratele lui Mircea schimbă cel puţin un număr în fişier.

• Numerele schimbate de fratele lui Mircea sunt mai mici sau cel mult egale cu 60000.

Exemplu

numere.in	**numere.out**	Explicaţie
3 5 0 7 0 0 1 6 9 8	2 4	În fişierul de intrare au fost înlocuite cu 0 numerele 2, 3, 4.

Timp maxim de execuţie: 1 secundă/test

4.1.1 Indicaţii de rezolvare - descriere soluţie *

Soluţia oficială

Se foloseşte un vector cu componente 0 sau 1, $x = (x_1, ..., x_m)$, unde m este 64000 dacă numărul de componente al tabloului (n^2) este mai mare decât 64000, respectiv $m = n^2$ în celălalt caz.

Iniţial vectorul x are toate componentele 0. Pe măsură ce se citesc numere v din fişier, componentele corespunzătoare din x se schimbă în 1 ($x_v := 1$).

După citirea numerelor din fişier se obţine în x o secvenţă de 0. Indicii corespunzători acestei secvenţe formează mulţimea de numere consecutive care au fost înlocuite cu 0 de fratele lui Mircea.

O altă modalitate de rezolvare constă în calculul sumei tuturor numerelor din tablou şi obţinerea astfel a sumei secvenţei de numere consecutive şterse de Mircea. Din păcate sumele sunt prea mari şi depăşesc tipurile predefinite. Dacă se folosesc implementări pe numere mari se obţine punctajul maxim, altfel doar jumătate din punctaj.

Soluţie prezentată în GInfo nr. 15/3

Pentru rezolvarea acestei probleme este suficient să utilizăm un şir de biţi care vor indica dacă un număr apare sau nu în fişier.

Vom avea nevoie întotdeauna de cel mult 250.000 de biţi, deci 31250 de octeţi.

Iniţial toţi biţii vor avea valoarea 0, iar pe măsură ce sunt citite numerele care formează matricea, valoarea bitului corespunzător unui număr citit devine 1.

Şirul de biţi va conţine în final o secvenţă de zerouri care va reprezenta soluţia problemei.

Există şi posibilitatea de a utiliza heap-ul pentru a păstra 250.000 de valori booleene sau întregi care vor permite şi ele determinarea secvenţei care lipseşte.

4.1.2 Rezolvare detaliată

4.1.3 Codul sursă *

```
import java.io.*;
class Numere
{
  static byte[] x=new byte[600001];
  static int n,min=0,max=0;

  public static void main(String[] args) throws IOException
  {
    int i,j;
    long t1,t2;
    t1=System.currentTimeMillis();
    StreamTokenizer st=new StreamTokenizer(
                       new BufferedReader(
                       new FileReader("numere.in")));
    st.nextToken(); n=(int)st.nval;
    for(i=1;i<=n*n;i++)
    {
      st.nextToken(); j=(int)st.nval;
      if(j<=60000) x[j]=1;
    }
    min=1;
    for(i=1;i<=60000;i++) if(x[i]==0) {min=i; break;}
    max=min;
    for(i=min+1;i<=60000;i++)
      if(x[i]==0) max++; else break;

    PrintWriter out=new PrintWriter(
                    new BufferedWriter(
                    new FileWriter("numere.out")));
    out.println(min+" "+max);
    out.close();
    t2=System.currentTimeMillis();
    System.out.println("Timp = "+(t2-t1));
  }
}
```

4.2 MaxD

Maria şi Adrian Niţă

Fiind elev în clasa a IX-a, George, îşi propune să studieze capitolul divizibilitate cât mai bine. Ajungând la numărul de divizori asociat unui număr natural, constată că sunt numere într-un interval dat, cu acelaşi număr de divizori.

De exemplu, în intervalul $[1, 10]$, 6, 8 şi 10 au acelaşi număr de divizori, egal

cu 4. De asemenea, 4 şi 9 au acelaşi număr de divizori, egal cu 3 etc.

Cerinţă

Scrieţi un program care pentru un interval dat determină care este cel mai mic număr din interval ce are număr maxim de divizori. Dacă sunt mai multe numere cu această proprietate se cere să se numere câte sunt.

Date de intrare

Fişierul de intrare **maxd.in** conţine pe prima linie două numere a şi b separate prin spaţiu ($a \leq b$) reprezentând extremităţile intervalului.

Date de ieşire

Fişierul de ieşire **maxd.out** va conţine pe prima linie trei numere separate prin câte un spaţiu *min nrdiv contor* cu semnificaţia:

min = cea mai mică valoare din interval care are număr maxim de divizori

nrdiv = numărul de divizori ai lui *min*

contor = câte numere din intervalul citit mai au acelaşi număr de divizori egal cu *nrdiv*

Restricţii şi precizări

- $1 \leq a \leq b \leq 1.000.000.000$
- $0 \leq b - a \leq 10.000$

Punctaj

Dacă aţi determinat corect *min*, obţineţi 50% din punctaj.

Dacă aţi determinat corect *nrdiv*, obţineţi 20% din punctaj

Dacă aţi determinat corect *contor*, obţineţi 30% din punctaj.

Exemple

maxd.in	**maxd.out**	Explicaţie
200 200	200 12 1	200 are 12 divizori iar in intervalul $[200, 200]$ există un singur număr cu această proprietate

maxd.in	**maxd.out**	Explicaţie
2 10	6 4 3	6 este cel mai mic număr din interval care are maxim de divizori egal cu 4 şi sunt 3 astfel de numere 6, 8, 10

Timp maxim de execuţie: 1 sec/test

4.2.1 Indicaţii de rezolvare - descriere soluţie *

Soluţia oficială

Pentru un număr natural n cuprins în intervalul $[a, b]$, să considerăm descompunerea în factori primi:

$$n = f_1^{e_1} * f_2^{e_2} * f_3^{e_3} * ... * f_k^{e_k}$$

Numărul de divizori ai lui n este dat de formula:

$$(e_1 + 1) * (e_2 + 1) * (e_3 + 1) * ... * (e_k + 1)$$

Se obţine un timp de execuţie mai bun dacă pentru factori (f_k) se realizează un tablou unidimensional format de numerele prime din domeniul *int*, iar n este verificat - legat de proprietatea de divizibilitate - doar pe valori numere prime din acest tablou.

Se determină pentru fiecare valoare x din intervalul $[a, b]$ numărul său de divizori, se reţine acea valoare ce are număr maxim de divizori; în caz de valori cu acelaşi număr maxim de divizori se incrementează contorul ce reţine numărul acestor valori.

Soluţie prezentată în GInfo nr. 15/3

Pentru fiecare număr în intervalul dat, vom determina numărul divizorilor acestuia.

Pentru aceasta vom determina descompunerea în factori primi a fiecărui număr.

Pentru un număr n, descompunerea în factori primi are forma:
$n = f_1^{e_1} \cdot f_2^{e_2} \cdot f_3^{e_3} \cdot ... \cdot f_k^{e_k}$,
iar numărul divizorilor va fi dat de valoarea
$(e_1 + 1)(e_2 + 1)(e_3 + 1)...(e_k + 1)$.

Pentru fiecare valoare x din intervalul $[a, b]$ se determină numărul său de divizori, se reţine acea valoare care are număr maxim de divizori; în cazul identificării unor valori cu acelaşi număr maxim de divizori, se incrementează contorul care reţine numărul acestor valori.

În cazul identificării unei valori cu număr mai mare de divizori, valoarea contorului devine 1 şi se actualizează variabila care conţine numărul cu cei mai mulţi divizori.

În final se va afişa prima valoare x care are cel mai mare număr de divizori, precum şi valoarea contorului.

4.2.2 Rezolvare detaliată

4.2.3 Codul sursă *

Varianta 1:

```
import java.io.*; //timp = 5600
class MaxD
{
  static int[] x;
  static int a,b;

  public static void main(String[] args) throws IOException
  {
```

```
    int i;
    long t1,t2;
    t1=System.currentTimeMillis();
    StreamTokenizer st=new StreamTokenizer(
          new BufferedReader(new FileReader("maxd.in")));
    st.nextToken(); a=(int)st.nval;
    st.nextToken(); b=(int)st.nval;
    x=new int[b-a+2];
    for(i=a;i<=b;i++) x[i-a+1]=descfact(i);
    int max=-1;
    int imax=-1;
    for(i=1;i<=b-a+1;i++)
      if(x[i]>max) { max=x[i]; imax=i; }
    int nrmax=0;
    for(i=1;i<=b-a+1;i++) if(x[i]==max) nrmax++;

    PrintWriter out=new PrintWriter(
         new BufferedWriter(new FileWriter("maxd.out")));
    out.println((imax+a-1)+" "+max+" "+nrmax);
    out.close();
    t2=System.currentTimeMillis();
    System.out.println("Timp = "+(t2-t1));
  }

  static int descfact(int nr)
  {
    int d;
    int nrd;
    int p=1;

    d=2;
    nrd=0;
    while(nr%d==0) { nrd++; nr=nr/d; }
    p=p*(nrd+1);

    d=3;
    while(d*d<=nr)
    {
      nrd=0;
      while(nr%d==0) { nrd++; nr=nr/d; }
      p=p*(nrd+1);
      d=d+2;
    }
    if(nr>1) p*=2;
```

```
    return p;
  }
}
```

Varianta 2:

```
import java.io.*;
class MaxD
{
  public static void main(String[] args) throws IOException
  {
    int i;
    int a,b;

    long t1,t2;
    t1=System.currentTimeMillis();

    StreamTokenizer st=new StreamTokenizer(
      new BufferedReader(new FileReader("maxd.in")));

    st.nextToken(); a=(int)st.nval;
    st.nextToken(); b=(int)st.nval;

    int max=-1;
    int imax=-1;
    int nrmax=0;
    int nd;

    for(i=a;i<=b;i++)
    {
      nd=nrDiv(i);
      if(nd>max) {max=nd; imax=i; nrmax=1;}
        else if(nd==max) nrmax++;
    }

    PrintWriter out=new PrintWriter(
      new BufferedWriter(new FileWriter("maxd.out")));
    out.println(imax+" "+max+" "+nrmax);
    out.close();
    t2=System.currentTimeMillis();
    System.out.println("Timp = "+(t2-t1));
  }// main(...)
```

```
  static int nrDiv(int nr)
  {
    int d;
    int nrd;
    int p=1;

    d=2;
    nrd=0;
    while(nr%d==0) { nrd++; nr=nr/d; }
    p=p*(nrd+1);

    d=3;
    while(d*d<=nr)
    {
      nrd=0;
      while(nr%d==0) { nrd++; nr=nr/d; }
      p=p*(nrd+1);

      d=d+2;
    }

    if(nr>1) p*=2;
    return p;
  }// nrDiv(...)
}// class
```

Capitolul 5

OJI 2006 clasa a IX-a

5.1 Flori

Cristina Bohm

Fetiţele din grupa mare de la grădiniţă culeg flori şi vor să împletească coroniţe pentru festivitatea de premiere. În grădină sunt mai multe tipuri de flori. Fiecare dintre cele n fetiţe culege un buchet având acelaşi număr de flori, însă nu neapărat de acelaşi tip. Pentru a împleti coroniţele fetiţele se împart în grupe. O fetiţă se poate ataşa unui grup numai dacă are cel puţin o floare de acelaşi tip cu cel puţin o altă fetiţă din grupul respectiv.

Cerinţă

Fiind dat un număr natural n reprezentând numărul fetiţelor şi numărul natural k reprezentând numărul de flori dintr-un buchet, să se determine grupele care se formează.

Date de intrare

Fişierul de intrare **flori.in** conţine pe prima linie, separate printr-un spaţiu, numerele naturale n şi k, reprezentând numărul de fetiţe şi respectiv numărul de flori din fiecare buchet. Fiecare dintre următoarele n linii conţine, pentru fiecare fetiţă, câte k valori separate prin câte un spaţiu reprezentând tipurile de flori culese.

Date de ieşire

Fişierul de ieşire **flori.out** va conţine pe fiecare linie câte o grupă formată din numerele de ordine ale fetiţelor separate prin câte un spaţiu, în ordine crescătoare, ca în exemplu.

Restricţii şi precizări

- $1 \leq n \leq 150$
- $1 \leq k \leq 100$
- Tipul unei flori este un număr întreg din intervalul $[0, 100]$.

• Într-o grupă numerele de ordine ale fetiţelor trebuie date în ordine strict crescătoare.

• În fişierul de ieşire grupele vor fi afişate în ordinea crescătoare a numărului de ordine al primei fetiţe din grupă.

Exemplu

flori.in	**flori.out**	Explicaţie
5 4 1 2 3 4 5 6 9 6 1 1 1 1 2 4 4 3 7 7 7 7	1 3 4 2 5	Fetiţele 1 şi 3 au cules amândouă flori de tipul 1, iar fetiţele 1 şi 4 au cules amândouă flori de tipurile 2,3 şi 4, deci toate cele trei fetiţe (1, 3, 4) se vor afla în aceeaşi grupă. Fetiţele 2 şi 5 vor forma fiecare câte o grupă deoarece nu au cules flori de acelaşi tip cu nici una dintre celelalte fetiţe.

Timp de rulare/test: 1 secundă

5.1.1 Indicaţii de rezolvare - descriere soluţie *

Soluţia comisiei

- citesc n - numărul de fetiţe şi k - numărul de flori dintr-un buchet

- construiesc matricea `a` definita astfel : pe linia i sunt tipurile distincte de flori ale fetiţei cu numărul de ordine i

- `a[i][0]` = numărul de elemente de pe linia i; acesta va deveni 0 dacă linia a fost reunită în altă linie

- vectorul `viz` are n elemente şi pe parcursul prelucrării , fetiţele care ajung în aceeaşi grupă vor avea aceeaşi valoare în vectorul `viz`: de exemplu, dacă fetiţa 3 ajunge în grupa în care e fetiţa 1 atunci `viz[3]=viz[1]`;

- inţial `viz[i]=i` însemnând că fiecare fetiţă e în grupă doar ea cu ea;

- apelul `irelj(i,j)` verifică dacă i e în relaţie cu j: caută pe linia i şi j un tip de floare comun fetiţelor i şi j

- funcţia `reuneste` face reuniunea mulţimilor de pe liniile i şi j în linia i; dacă s-a făcut o astfel de reuniune, scad i ($i - -$) şi astfel se rezolvă situaţia în care de exemplu `i rel j, not ( i rel k) , j rel k`; executând `i--`, k va ajunge tot în grupă cu i; altfel k ar ajunge în altă grupă

- afişarea grupelor presupune selectarea din vectorul `viz` a poziţiilor care au aceeaşi valoare: toate poziţiile i care au `viz[i]=1` (de exemplu) sunt în prima grupă; pun pe 0 poziţiile afişate pentru a nu le mai relua o dată.

5.1.2 Rezolvare detaliată

5.1.3 Codul sursă *

Variantă iterativă:

```
import java.io.*;
class Flori1
{
  static int n,k;
  static int[][] a=new int[151][101];
  static int[] gf=new int[151];
  static int[] fgc=new int[101];

  public static void main(String[] args) throws IOException
  {
    int i,j,ii,ng,ok,gasit;
    StreamTokenizer st=new StreamTokenizer(
      new BufferedReader(new FileReader("flori.in")));
    PrintWriter out=new PrintWriter(
      new BufferedWriter(new FileWriter("flori.out")));

    st.nextToken(); n=(int)st.nval;
    st.nextToken(); k=(int)st.nval;

    for(i=1;i<=n;i++)
      for(j=1;j<=k;j++) { st.nextToken(); a[i][j]=(int)st.nval; }

    ng=0;
    for(i=1;i<=n;i++)
    {
      if(gf[i]!=0) continue;
      ng++;
      for(j=0;j<=100;j++) fgc[j]=0;
      for(j=1;j<=k;j++) fgc[a[i][j]]=1;

      ok=1;
      while(ok==1)
      {
        ok=0;
        for(ii=i+1;ii<=n;ii++)
        {
```

```
          if(gf[ii]!=0) continue;
          gasit=0;
          for(j=1;j<=k;j++) if(fgc[a[ii][j]]==1)
          {
            gasit=1;
            break;
          }

          if(gasit==1)
          {
            for(j=1;j<=k;j++) fgc[a[ii][j]]=1;
            ok=1;
            gf[ii]=ng;
          }
        }//for ii
      }//while

      out.print(i+" ");
      for(j=1;j<=n;j++) if(gf[j]==ng) out.print(j+" ");
      out.println();
    }//for i

    out.close();
  }// main
}// class
```

Variantă recursivă:

```
import java.io.*;
class Flori2
{
  static int n,k,ng;
  static char[][] a=new char[151][101];
  static int[] gf=new int[151];

  public static void main(String[] args) throws IOException
  {
    int i,j,fij;
    StreamTokenizer st=new StreamTokenizer(
      new BufferedReader(new FileReader("flori.in")));
    PrintWriter out=new PrintWriter(
      new BufferedWriter(new FileWriter("flori.out")));
```

```
    st.nextToken(); n=(int)st.nval;
    st.nextToken(); k=(int)st.nval;

    for(i=1;i<=n;i++)
      for(j=1;j<=k;j++) { st.nextToken(); fij=(int)st.nval; a[i][fij]=1;}

    ng=0;
    for(i=1;i<=n;i++)
    {
      if(gf[i]!=0) continue;
      ng++;
      fata(i);
    }

    for(i=1;i<=ng;i++)
    {
      for(j=1;j<=n;j++) if(gf[j]==i) out.print(j+" ");
      out.println();
    }

    out.close();
  }// main

  static void fata(int i)
  {
    int j,ii;

    gf[i]=ng;
    for(j=0;j<=100;j++)
    {
      if(a[i][j]==0) continue;
      for(ii=1;ii<=n;ii++)
      {
        if(ii==i) continue;
        if(a[ii][j]==1)
        if(gf[ii]==0) fata(ii);
      }
    }
  }// fata(...)
}// class
```

5.2 Pluton

Marinel Serban

În timpul acţiunii "Furtună în deşert" din cauza unei furtuni de nisip, n soldaţi s-au rătăcit de plutoanele lor. După trecerea furtunii se pune problema regrupării acestora pe plutoane. Pentru aceasta se folosesc plăcuţele de identificare pe care soldaţii le poartă la gât. Pe aceste plăcuţe sunt scrise numere care pot identifica fiecare soldat şi plutonul din care acesta face parte. Astfel, soldaţii din acelaşi pluton au numărul de identificare format din aceleaşi cifre, dispuse în altă ordine şi numerele de identificare sunt unice. De exemplu, numerele de identificare 78003433, 83043073, 33347008 indică faptul că cei trei soldaţi care le poartă fac parte din acelaşi pluton.

Cerinţă

Fiind date cele n numere de pe plăcuţele de identificare, să se regrupeze cei n soldaţi pe plutoane, indicându-se numărul de plutoane găsite (un pluton refăcut trebuie să aibă minimum un soldat), numărul de soldaţi din cel mai numeros pluton, numărul de plutoane care au acest număr maxim de soldaţi precum şi componenţa unui astfel de pluton (cu număr maxim de soldaţi regrupaţi).

Date de intrare

Fişierul de intrare **pluton.in** conţine pe prima linie numărul n de soldaţi recuperaţi, iar pe fiecare dintre următoarele n linii câte un număr de identificare a celor n soldaţi.

Date de ieşire

Fişierul de ieşire **pluton.out** va conţine pe prima linie numărul de plutoane refăcute. Linia a doua va conţine numărul de soldaţi din cel mai numeros pluton refăcut. Linia a treia va conţine numărul de plutoane care au numărul maxim de soldaţi recuperaţi. Linia a patra va conţine componenţa unui astfel de pluton, cu număr maxim de soldaţi recuperaţi, numerele de identificare ale soldaţilor din componenţă fiind scrise unul după altul separate prin câte un spaţiu.

Restricţii şi precizări

- $0 < n \leq 4000$
- $0 <$ număr de identificare $< 2.000.000.000$

Observaţii

Deoarece linia a patra conţine numerele de identificare ale soldaţilor unuia dintre plutoanele cu un număr maxim de soldaţi, pot exista mai multe soluţii corecte. Se poate alege oricare dintre acestea.

Se acordă punctaje parţiale astfel: pentru valoarea corectă de pe prima linie se acordă 30% din punctaj; pentru valorile corecte de pe prima şi a doua linie se acordă 50% din punctaj, pentru valorile corecte de pe prima, a doua şi a treia linie se acordă 70% din punctaj, iar pentru rezolvarea corectă a tuturor cerinţelor se acordă punctajul integral aferent testului.

Exemplu

pluton.in	**pluton.out**	Explicaţie
10	6	Au fost recuperaţi soldaţi din 6 plutoane
1223	3	distincte, cei mai mulţi soldaţi recuperaţi
123	2	dintr-un pluton fiind în număr de 3.
666	321 312 123	
321		Există 2 plutoane cu număr maxim de
7890		soldaţi recuperaţi (3), unul dintre ele
2213		fiind format din soldaţii cu numerele
312		321 312 123.
655		
1000		De remarcat că şi soluţia
1322		1223 2213 1322 este corectă.

Timp de rulare/test: 1 secundă

5.2.1 Indicaţii de rezolvare - descriere soluţie *

Soluţia comisiei

Soluţia 1:

– în timpul citirii creez un nou vector care conţine pe poziţiile corespunzătoare numerele de identificare din vectorul iniţial în ordinea descrescatoare a cifrelor

– în etapa a doua se parcurge vectorul nou format grupând toate numerele de identificare identice; după formarea unui grup (pluton) se determină mărimea acestuia reţinâdu-se acesta dacă e cel mai numeros găsit până în acel moment sau contorizându-l dacă numărul de soldaţi determinat este egal cu cel maxim determinat anterior.

Soluţia 2:

– citesc numerele de identificare în vectorul `a[]`

– construiesc 2 vectori ajutători

- vectorul `b[]` care va conţine numărul de cifre al fiecărui element
- vectorul `c[]` care va conţine numărul de cifre distincte a fiecărui element

– ordonez cei trei vectori crescător după numărul de cifre distincte (după `c[]`) şi după numărul de cifre, deci cheia de sortare va fi un număr construit după formula `c[i]*10+b[i]`

– formez şi număr plutoanele; plutoanele le voi reţine pe linii distincte a matricei `G[MAX][2]`

- în fiecare linie, elementul `G[i][0]` va conţine numărul de elemente din pluton

- elementul `G[i][1]` va conţine reprezentantul plutonului, primul care apare în `a[]`
- repet până când toate elementele din `a` au fost verificate
  - reţinem primul element nepus încă din pluton cu caracteristicile lui
  - verific elementele cu aceleaşi caracteristici să facă parte din acelaşi pluton cu primul element din pluton pe care l-am reţinut
  - testez dacă are aceleaşi cifre
    - dacă nu are aceleaşi cifre trec mai departe
    - altfel îl numar (încă face parte din acelaşi pluton)

– detectez numărul maxim de elemente ale unui pluton şi reţin maximul

– afişare cerinţele 1 2 3 folosind valorile aflate

– la cerinţa 4

  - caut in a[] reprezentantul unui pluton numeros
  - afişez cele `maxe` elemente - în vectorul sortat elementele cu aceleaşi caracteristici (b şi c) sunt unul după altul, dar mai trebuie verificat să aibă caracteristicile reprezentantului (ex. 1212 şi 3434 au aceleaşi caracteristici (4,2), (4,2), dar nu fac parte din acelaşi pluton)

Soluţia 3:

– se utilizeaza notiunea de lista: dintr-o lista fac parte toti membrii unui pluton

– se construieste un vector ajutator care retine pentru fiecare element elementul care il urmeaza in lista

– pe parcursul formarii listelor se determina lista cea mai numeroasa precum si numarul de liste de acest tip

– afisarea se face utilizand informatiile din vetorul urm

5.2.2 Rezolvare detaliată

5.2.3 Codul sursă *

Solutie "incorectă" pentru Borland C++ 3.1 dar care ia 100 puncte !!!

```
import java.io.*;
class Pluton1
{
  static int n,np;
  static long[] ni=new long[4001];
  static long[] nid=new long[4001];
  static int[] nip=new int[4001];
  static int[] z=new int[4001];
  static int[] fc=new int[10];
  static int[] x=new int[11];

  static long cifre(long nr)  // 1230456789 --> 9.876.543.210 !!!
  {
    int i,j,k,nc=0;
    long nrcd=0;              // nr cu cifre descrescatoare

  for(i=0;i<=9;i++) fc[i]=0;
  for(i=0;i<=10;i++) x[i]=0;

  while(nr!=0) { fc[(int)(nr%10)]++; nr=nr/10; nc++; }

  k=0;
  for(i=9;i>=0;i--)
    if(fc[i]!=0)
      for(j=1;j<=fc[i];j++) { k++; x[k]=i; }

    for(i=1;i<=nc;i++) nrcd=nrcd*10+x[i];
    return nrcd;
  }// cifre(...)

  public static void main(String[] args) throws IOException
  {
    int i,j;
    int max,npmax,pmax;

    StreamTokenizer st=new StreamTokenizer(
      new BufferedReader(new FileReader("9-pluton.in")));
    PrintWriter out=new PrintWriter(
      new BufferedWriter(new FileWriter("pluton.out")));

    st.nextToken(); n=(int)st.nval;
    for(i=1;i<=n;i++) {st.nextToken(); ni[i]=(int)st.nval;}

    for(i=1;i<=n;i++) nid[i]=cifre(ni[i]);
```

```
    np=0;
    for(i=1;i<=n;i++)
    {
      if(nip[i]!=0) continue;
      np++;
      nip[i]=np;
      for(j=i+1;j<=n;j++)
        if(nip[j]==0)
          if(nid[j]==nid[i]) nip[j]=np;
    }

    out.println(np);

    for(i=1;i<=np;i++)
    for(j=1;j<=n;j++) if(nip[j]==i) z[i]++;

    max=0;
    npmax=0;
    pmax=0;

    for(i=1;i<=np;i++) if(z[i]>max) {max=z[i];pmax=i;}
    out.println(max);

    for(i=1;i<=np;i++) if(z[i]==max) npmax++;
    out.println(npmax);

    for(i=1;i<=n;i++) if(nip[i]==pmax) out.print(ni[i]+" ");

    out.println();
    out.close();
  }// main
}// class
```

Soluţie "corectă" şi pentru Borland C++ 3.1

```
import java.io.*;
class Pluton2
{
  static int n,np;
  static int[] ni=new int[4001];
  static int[] nid1=new int[4001];
  static int[] nid2=new int[4001];
```

```
  static int[] nip=new int[4001];
  static int[] z=new int[4001];
  static int[] fc=new int[10];
  static int[] x=new int[11];

  static void nrcd(int nr,int i0)
  {
    int i,j,k,nc;
    int nrcd1, nrcd2;   // nr cu cifre descrescatoare = |nrcd1|nrcd2|

    for(i=0;i<=9;i++) fc[i]=0;
    for(i=0;i<=10;i++) x[i]=0;

    nc=0;
    while(nr!=0) { fc[(int)(nr%10)]++; nr=nr/10; nc++; }

    k=0;
    for(i=0;i<=9;i++)
      if(fc[i]!=0)
        for(j=1;j<=fc[i];j++) { k++; x[k]=i; }

    nrcd1=0;
    for(i=nc;i>=6;i--) nrcd1=nrcd1*10+x[i];

    nrcd2=0;
    for(i=5;i>=1;i--)  nrcd2=nrcd2*10+x[i];

    nid1[i0]=nrcd1;
    nid2[i0]=nrcd2;
  }// cifre(...)

  public static void main(String[] args) throws IOException
  {
    int i,j;
    int max,npmax,pmax;

    StreamTokenizer st=new StreamTokenizer(
      new BufferedReader(new FileReader("pluton.in")));
    PrintWriter out=new PrintWriter(
      new BufferedWriter(new FileWriter("pluton.out")));

    st.nextToken(); n=(int)st.nval;
    for(i=1;i<=n;i++) {st.nextToken(); ni[i]=(int)st.nval;}
```

```
    for(i=1;i<=n;i++) nrcd(ni[i],i);

    np=0;
    for(i=1;i<=n;i++)
    {
      if(nip[i]!=0) continue;
      np++;
      nip[i]=np;
      for(j=i+1;j<=n;j++)
      if(nip[j]==0)
        if((nid1[j]==nid1[i])&&(nid2[j]==nid2[i])) nip[j]=np;
    }

    out.println(np);

    for(i=1;i<=np;i++)
    for(j=1;j<=n;j++) if(nip[j]==i) z[i]++;

    max=0;
    npmax=0;
    pmax=0;

    for(i=1;i<=np;i++) if(z[i]>max) {max=z[i];pmax=i;}
    out.println(max);

    for(i=1;i<=np;i++) if(z[i]==max) npmax++;
    out.println(npmax);

    for(i=1;i<=n;i++) if(nip[i]==pmax) out.print(ni[i]+" ");

    out.println();
    out.close();
  }// main
}// class
```

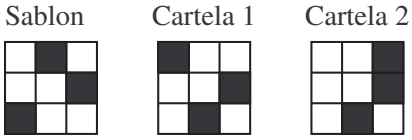
Capitolul 6

OJI 2007 clasa a IX-a

6.1 Cartele - OJI 2007

În sediul unei firme se intră doar cu ajutorul cartelelor magnetice. De câte ori se schimbă codurile de acces, cartelele trebuie formatate. Formatarea presupune imprimarea unui model prin magnetizare. Dispozitivul în care se introduc cartelele, numit cititor de cartele, verifică acest model. Toate cartelele au aceleaşi dimensiuni, suprafaţa pătrată şi grosimea neglijabilă. Cele două feţe plane ale unei cartele se împart fiecare în $N \times N$ celule pătrate, identice ca dimensiuni. Prin formatare unele celule, marcate cu negru în exemplu, se magnetizează permiţând radiaţiei infraroşii să treacă dintr-o parte în cealaltă a cartelei. În interiorul cititorului de cartele se iluminează uniform una dintre feţele cartelei. De cealaltă parte fasciculele de lumină care străbat cartela sunt analizate electronic. Pentru a permite accesul în clădire modelul imprimat pe cartelă trebuie să coincidă exact cu modelul şablonului care memorează codul de intrare. Prin fanta dispozitivului nu se pot introduce mai multe cartele deodată. Cartela se poate introduce prin fantă cu oricare dintre muchii spre deschizătura fantei şi cu oricare dintre cele două feţe orientate către şablon. După introducere cartela se dispune în plan paralel cu şablonul, lipit de acesta, astfel încât cele patru colţuri ale cartelei se suprapun exact cu colţurile şablonului. Modelele imprimate pe cele două feţe ale unei cartele sunt identice. Unei celule magnetizate îi corespunde pe faţa opusă tot o celulă magnetizată, iar unei celule nemagnetizate îi corespunde una nemagnetizată. O celulă magnetizată este transparentă pentru radiaţia infraroşie indiferent de faţa care se iluminează.

Un angajat al firmei are mai multe cartele. Pe unele dintre acestea a fost imprimat noul cod de intrare, iar pe altele sunt coduri mai vechi. Pentru a afla care sunt cartelele care-i permit accesul în sediul firmei angajatul este nevoit să le verifice pe toate, introducîndu-le pe rând, în toate modurile pe care le consideră necesare, în fanta cititorului de cartele.



Cerinţă

Scrieţi un program care determină care dintre cartele permite accesul în sediul firmei.

Date de intrare

Fişierul de intrare **cartele.in** conţine pe prima linie două numere naturale N şi C despărţite printr-un spaţiu. N este dimensiunea tablourilor care reprezintă modelul şablon şi modelele cartelelelor. C reprezintă numărul de cartele. Urmează $C + 1$ blocuri de câte N linii fiecare. Primul bloc de N linii codifică şablonul. Următoarele C blocuri de câte N linii codifică fiecare câte o cartelă. Pe fiecare linie sunt câte N valori întregi, despărţite printr-un singur spaţiu. Celulelor magnetizate le corespunde valoarea 1, iar celorlalte, valoarea 0.

Date de ieşire

În fişierul de ieşire **cartele.out** se vor scrie C linii, câte o valoare pe linie. Pe linia i se va scrie valoarea 1 dacă cartela i permite accesul în clădire şi valoarea 0 în caz contrar.

Restricţii şi precizări

$1 < N, C \leq 50$

Exemplu

cartele.in	**cartele.out**	**Explicaţii**
3 2 0 1 0 0 0 1 1 0 0 1 0 0 0 0 1 0 1 0 0 0 1 0 0 1 0 1 0	1 0	Datele de intrare corespund situaţiei din figură. Cartela 1 se potriveşte perfect şablonului, dacă se roteşte în sens trigonometric cu 90 de grade. Cartela 2 nu se potriveşte şablonului, indiferent de modul în care se introduce în fantă.

Timp maxim de execuţie/test: 1 secundă

6.1.1 Indicaţii de rezolvare - descriere soluţie *

Pentru fiecare cartelă, se compară element cu element, matricea care reprezintă sablonul, cu următoarele tablouri:

1. Cartela
2. Cartela rotită cu 90 grade
3. Cartela rotită cu 180 grade
4. Cartela rotită cu 270 grade

Dacă nu s-a gasit o coincidenţă, se întoarce cartela, printr-o operaţie de oglindire faţă de linia $i = n/2$, (sau faţă de coloana $j = n/2$), după care se compară şablonul cu următoarele tablouri:

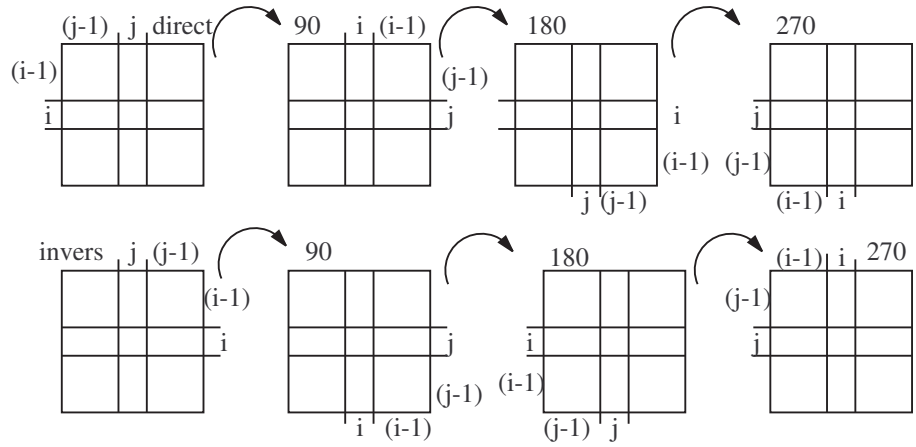
5. Cartela oglindită
6. Cartela oglindită rotită cu 90 grade
7. Cartela oglindită rotită cu 180 grade
8. Cartela oglindită rotită cu 270 grade

Rotirile se pot face în sens trigonometric sau orar.

Dacă s-a găsit o coincidenţă la oricare dintre paşii de mai sus, se opreşte căutarea, se afisează 1 şi se trece la prelucrarea următoarei cartele.

Dacă nici după pasul 8 nu s-a gasit o potrivire exactă, se afişează 0 şi se trece la prelucrarea următoarei cartele.

6.1.2 Rezolvare detaliată



6.1.3 Codul sursă *

Varianta 1:

```
import java.io.*;
```

```
class cartele
{
  static final int DIM=51;

  static StreamTokenizer st;
  static PrintWriter out;

  static int[][] a=new int[DIM][DIM];  // sablonul
  static int[][] b=new int[DIM][DIM];  // cartela
  static int[][] aux=new int[DIM][DIM];// auxiliar

  static int n, c;

  public static void main(String[] args) throws IOException
  {
    st=new StreamTokenizer(new BufferedReader(new FileReader("cartele.in")));
    out=new PrintWriter(new BufferedWriter(new FileWriter("cartele.out")));

    rezolva();
    out.close();
  }// main(...)

  static void rezolva() throws IOException
  {
    int i, j, k, r;
    boolean identice;

    st.nextToken(); n=(int)st.nval;
    st.nextToken(); c=(int)st.nval;

    for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)
      {
        st.nextToken(); a[i][j]=(int)st.nval;
      }

    for(k=1;k<=c;k++)
    {
      identice=true;
      for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        {
          st.nextToken(); b[i][j]=(int)st.nval;
          if(b[i][j]!=a[i][j]) identice=false;
```

```
        }

      for(int f=1;f<=2&&!identice;f++)  // pentru fata 1 si 2
      {
        for(r=1;r<=4&&!identice;r++)  // la a patra rotatie se revine la matricea initiala
        {
          roteste();    // cu 90 in sens trigonometric
          if(egale()) identice=true;
        }
        if(!identice) inverseaza();
      }
      if(identice) out.println(1); else  out.println(0);
    }// for k
  }// rezolva(...)

  static boolean egale()
  {
    for(int i=1;i<=n;i++)
      for(int j=1; j<=n; j++)
        if(a[i][j]!=b[i][j] ) return false;
    return true;
  }// egale(...)

  static void inverseaza()
  {
    int i, j, temp;
    for(i=1;i<=n/2;i++)
      for(j=1;j<=n;j++) { temp=b[i][j]; b[i][j]=b[n-i+1][j]; b[n-i+1][j]=temp; }
  } // inverseaza(...)

  static void roteste()
  {
    int i, j;
    for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)  aux[n-j+1][i]=b[i][j];

    for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)  b[i][j]=aux[i][j];
  }// roteste(...)
}// class
```

Varianta 2:

```
import java.io.*;
```

```
class cartele
{
  static final int DIM=51;

  static StreamTokenizer st;
  static PrintWriter out;

  static int[][] a=new int[DIM][DIM];  // sablonul
  static int[][] b=new int[DIM][DIM];  // cartela

  static int n, c;

  public static void main(String[] args) throws IOException
  {
    int i, j, k;
    boolean ok;

    st=new StreamTokenizer(new BufferedReader(new FileReader("cartele.in")));
    out=new PrintWriter(new BufferedWriter(new FileWriter("cartele.out")));

    st.nextToken(); n=(int)st.nval;
    st.nextToken(); c=(int)st.nval;

    for(i=1;i<=n;i++) // citesc sablonul
      for(j=1;j<=n;j++)
      {
        st.nextToken(); a[i][j]=(int)st.nval;
      }

    for(k=1;k<=c;k++)
    {
      for(i=1;i<=n;i++) // citesc cartela k
        for(j=1;j<=n;j++)
        {
          st.nextToken(); b[i][j]=(int)st.nval;
        }

      ok=true;
      for(i=1;i<=n&&ok;i++) // direct
        for(j=1;j<=n&&ok;j++)
          if(a[i][j]!=b[i][j]) ok=false;
      if(ok) {out.println(1); continue;} // cu alta cartela

      ok=true;
```

```
  for(i=1;i<=n&&ok;i++) // rotit cu 90 (ceas!)
    for(j=1;j<=n&&ok;j++)
      if(a[i][j]!=b[j][n-(i-1)]) ok=false;
  if(ok) {out.println(1); continue;} // cu alta cartela

  ok=true;
  for(i=1;i<=n&&ok;i++) // rotit cu 180 (ceas!)
    for(j=1;j<=n&&ok;j++)
      if(a[i][j]!=b[n-(i-1)][n-(j-1)]) ok=false;
  if(ok) {out.println(1); continue;} // cu alta cartela

  ok=true;
  for(i=1;i<=n&&ok;i++) // rotit cu 270 (ceas!) <==> 90 trig
    for(j=1;j<=n&&ok;j++)
      if(a[i][j]!=b[n-(j-1)][i]) ok=false;
  if(ok) {out.println(1); continue;} // cu alta cartela

  ok=true;
  for(i=1;i<=n&&ok;i++) // invers + direct
    for(j=1;j<=n&&ok;j++)
      if(a[i][j]!=b[i][n-(j-1)]) ok=false;
  if(ok) {out.println(1); continue;} // cu alta cartela

  ok=true;
  for(i=1;i<=n&&ok;i++) // invers + rotit 90 (ceas!)
    for(j=1;j<=n&&ok;j++)
      if(a[i][j]!=b[n-(j-1)][n-(i-1)]) ok=false;
  if(ok) {out.println(1); continue;} // cu alta cartela

  ok=true;
  for(i=1;i<=n&&ok;i++) // invers + rotit 180 (ceas!)
    for(j=1;j<=n&&ok;j++)
      if(a[i][j]!=b[n-(i-1)][j]) ok=false;
  if(ok) {out.println(1); continue;} // cu alta cartela

  ok=true;
  for(i=1;i<=n&&ok;i++) // invers + rotit cu 270 (ceas!) <==> 90 trig
    for(j=1;j<=n&&ok;j++)
      if(a[i][j]!=b[j][i]) ok=false;
  if(ok) {out.println(1); continue;} // cu alta cartela

  out.println(0); // nu s-a potrivit
} // for k
out.close();
```

```
  }// main
}// class
```

Varianta 3:

```
import java.io.*;
class cartele
{
  static final int DIM=51;

  static StreamTokenizer st;
  static PrintWriter out;

  static int[][] a=new int[DIM][DIM];  // sablonul
  static int[][] b=new int[DIM][DIM];  // cartela

  static int n, c;

  public static void main(String[] args) throws IOException
  {
    int i, j, k;

    st=new StreamTokenizer(new BufferedReader(new FileReader("1.in")));
    out=new PrintWriter(new BufferedWriter(new FileWriter("cartele.out")));

    st.nextToken(); n=(int)st.nval;
    st.nextToken(); c=(int)st.nval;

    for(i=1;i<=n;i++) // citesc sablonul
      for(j=1;j<=n;j++)
      {
        st.nextToken(); a[i][j]=(int)st.nval;
      }

    for(k=1;k<=c;k++)
    {
      for(i=1;i<=n;i++) // citesc cartela k
        for(j=1;j<=n;j++)
        {
          st.nextToken(); b[i][j]=(int)st.nval;
        }

      if(egale(1,0,0,0, 0,1,0,0)) {out.println(1); continue;} // direct
      if(egale(0,1,0,0, 0,0,1,0)) {out.println(1); continue;} // rotit cu 90 (ceas!)
```

```
      if(egale(0,0,1,0, 0,0,0,1)) {out.println(1); continue;} // rotit cu 180 (ceas!)
      if(egale(0,0,0,1, 1,0,0,0)) {out.println(1); continue;} // rotit cu 270 (ceas!)
      if(egale(1,0,0,0, 0,0,0,1)) {out.println(1); continue;} // invers + direct
      if(egale(0,0,0,1, 0,0,1,0)) {out.println(1); continue;} // invers + rotit 90 (ceas!)
      if(egale(0,0,1,0, 0,1,0,0)) {out.println(1); continue;} // invers + rotit 180 (ceas!)
      if(egale(0,1,0,0, 1,0,0,0)) {out.println(1); continue;} // invers + rotit cu 270 (ceas!)

      out.println(0); // nu s-a potrivit
    } // for k
    out.close();
  }// main

  static boolean egale(int i1,int j1, int ni1, int nj1, int i2, int j2, int ni2, int nj2)
  {
    int i,j;
    boolean ok=true;
    for(i=1;i<=n&&ok;i++)
      for(j=1;j<=n&&ok;j++)
        if(a[i][j]
           !=
           b[i*i1+j*j1+(n-i+1)*ni1+(n-j+1)*nj1][i*i2+j*j2+(n-i+1)*ni2+(n-j+1)*nj2])
             ok=false;
    return ok;
  }// egale(...)
}// class
```

6.2 Paritate - OJI 2007

În vederea asigurării unei transmiteri cât mai exacte a informaţiilor pe reţea, transmiterea se efectuează caracter cu caracter, fiecare caracter fiind dat prin codul său ASCII, adică o grupă de 8 biţi (octet). Pentru fiecare 8 biţi transmişi se calculează un bit de paritate care are valoarea 0 (dacă codul ASCII al caracterului conţine un număr par de cifre binare 1) sau 1 (în caz contrar).

Deoarece în problema noastră se transmit numai caractere ASCII standard, cu codul ASCII din intervalul [32,127], codul lor ASCII are bitul 7 (primul bit din stînga) egal cu 0. Pe această poziţie va fi pus bitul de paritate, economisind astfel câte un bit pentru fiecare caracter transmis. De exemplu, dacă mesajul care trebuie trasmis conţine caracterele "Paritate", succesiunea de biţi transmisă va fi:

01010000 11100001 01110010 01101001 01110100 11100001 01110100 01100101

În plus, pe lângă caracterele amintite, în mesaj mai poate să apară un caracter special care indică trecerea la începutul unui nou rând. Acest caracter are

codul ASCII 10.

Cerinţă

Să se scrie un program care să verifice dacă un text a fost sau nu transmis corect.

Date de intrare

Fişierul de intrare **paritate.in** are pe prima linie o succesiune de caractere '0' şi '1' care reprezintă mesajul transmis. Între caractere nu există spaţii. Linia se termină cu caracterul marcaj de sfârşit de linie (newline).

Date de ieşire

Fişierul de ieşire **paritate.out** are pe prima linie mesajul DA dacă textul a fost transmis corect sau NU în caz contrar. În cazul în care mesajul de pe prima linie este DA liniile următoare vor conţine textul transmis în clar. În cazul în care mesajul de pe prima linie este NU linia următoare va conţine numerele de ordine ale caracterelor care nu au fost transmise corect, în ordine strict crescătoare, separate prin câte un spaţiu.

Restricţii şi precizări

• Cei 8 biţi ai codului ASCII a unui caracter se numerotează de la 0 la 7, de la dreapta la stânga, cel mai din stânga bit fiind bitul 7 iar cel mai din dreapta bitul 0.

• Textul transmis are cel mult 60000 caractere.

• Numărul de caractere '0' şi '1' din prima linie a fişierului de intrare este multiplu de 8.

• Codurile ASCII ale caracterelor din text aparţin mulţimii $\{10, 32 - 127\}$, codul 10 însemnând trecerea la începutul unui rând nou.

• Nici o linie din fişierul de iec sire nu va avea mai mult de 255 caractere.

• Caracterele din text sunt numerotate începând de la 0.

• mesajele DA/NU din prima linie a fişierului de ieşire se scriu cu majuscule.

Exemplul 1:

paritate.in
0101000011100001011100100110100101110100111000010111010001100101

paritate.out	**Explicaţie**
DA Paritate	Toate codurile sunt

Exemplul 2:

paritate.in
1101000011100001111100100110100101110100111000010111010011100101

paritate.out	Explicaţie
NU 0 2 7	Primul caracter a fost transmis ca succesiunea de biţi 11010000 ceea ce înseamnă că fără bitul de paritate ar fi trebuit să existe un număr impar de cifre 1, ceea ce este fals. Deci caracterul nu a fosttransmis corect. Acelaşi lucru se verifică şi pentru caracterele cu numerele de ordine 2 şi 7.

Exemplul 3:

paritate.in
0100000111111010011010010000101001100101000010100110101001101111011010001

paritate.out	Explicaţie
DA Azi e joi	Toate codurile sunt corecte. În text există două caractere cu cod ASCII 10

Timp maxim de execuţie/test: 1 secundă

6.2.1 Indicaţii de rezolvare - descriere soluţie *

Se utilizează un tablou de caractere care va conţine:

- caracterul corect transmis sau

- caracterul #0 în cazul în care transmisia nu s-a efectual corect

În acelaşi timp variabila Eroare va conţine ultima poziţie a unui cod eronat sau 0 dacă nu sunt erori la transmiterea mesajului.

Deoarece suntem asiguraţi de faptul că numărul de biţi 0/1 transmişi este multiplu de 8, nu mai fac această verificare şi tratez fiecare grupă de 8 biţi astfel:

- citesc primul caracter separat (este bitul de paritate)

- îl transform în cifra 0/1

- citesc pe rând ceilalţi 7 biţi si formez codul ASCII corect numărând în acelaşi timp biţii egali cu 1

- dacă bitul de paritate este corect (adică am un număr par de cifre 1) pun pe poziţia corespunzătoare din tablou caracterul al cărui cod îl am - în caz contrar pun pe poziţia respectivă valoarea #0 şi reţin în variabila Eroare poziţia caracterului eronat

După terminarea acestui proces nu am decât să verific variabila Eroare:

- în cazul în care are valoarea 0 (transmisie fără eroare), afişez 'DA' în prima linie a fişierului de ieşire, apoi parcurg vectorul caracter cu caracter şi îl scriu în fişierul de ieşire, având grijă ca în cazul întâlnirii caracterului #10 (cod de linie nouă) să trec la o nouă linie

- în cazul în care are o valoare ¿0 (transmisie cu erori) afisez 'NU' în prima linie a fişierului de ieşire, apoi parcurg vectorul caracter cu caracter şi, în cazul întâlnirii valorii #0 (caracter eronat) afişez indicele respectiv.

6.2.2 Rezolvare detaliată

6.2.3 Codul sursă *

Varianta 1: Versiunea este o prelucrare a variantei oficiale; comentariile din sursa au rămas nemodificate pentru că sunt un exemplu bun!

```
import java.io.*;
class paritate
{
  static final int MAX=60000;
  static int[] a=new int[MAX];   // 0=eroare

  public static void main(String[] args) throws IOException
  {
    int c;
    int i,j,k;
    int BitP, Cod, Bit, Nr1;
    int Eroare;  // ultima pozitie a unui cod eronat sau 0 daca nu sunt erori

    //BufferedReader br0=new BufferedReader(new InputStreamReader(System.in));
    BufferedReader br=new BufferedReader(new FileReader("paritate.in"));
    PrintWriter out=new PrintWriter(new BufferedWriter(new FileWriter("paritate.out")));

    i=-1;
    Eroare=0;
    c=0;
    String s=br.readLine();
    //System.out.println("s = "+s);
    k=0;
    while(k<s.length()-1)
    {
      c=s.charAt(k);
      //System.out.println(k+" : "+c);
      //br0.readLine();
      i++;                      // pozitie caracter
      BitP=c-'0';               // bitul de paritate
      Cod=0;                    // aici formez codul
      Nr1=0;                    // cati de 1
      for(j=1;j<=7;j++)         // citesc ceilalti 7 biti
      {
        c=s.charAt(++k);        // citesc bit
        //System.out.println(k+" : "+c);
```

```
        //br0.readLine();
        Bit=c-'0';
        if(Bit==1) Nr1++;      // daca e 1 il numar
        Cod=Cod*2+Bit;         // formez codul
      }

      if((Nr1+BitP)%2==0)      // daca cod corect
        a[i]=Cod;              // pun caracterul in vector
      else                     // altfel
      {
        a[i]=1;                // pun 1
        Eroare=i;              // si retin pozitia
      }
      ++k;
    }// while

    if(Eroare==0)              // daca nu sunt erori
    {
      out.println("DA");       // scrie DA si
      for(j=0;j<=i;j++)        // afiseaza cele i+1 caractere
        if(a[j]==10)           // avand grija la caracterul cu codul 10
          out.println();
        else                   // altfel
        out.print((char)a[j]); // scrie caracterul
    }
    else                       // eroare!!!
    {
      out.println("NU");       // scrie NU si
      for(j=0;j<Eroare;j++)
        if(a[j]==1)            // cauta erorile - cod 01
          out.print(j+" ");    // si afiseaza pozitia lor
      out.println(Eroare);     // afiseaza pozitia ultimei erori
    }

    out.close();
  }// main
}// class
```

Capitolul 7

ONI 2000 clasa a IX-a

7.1 Algoritm

prof. Roxana Tâmplaru, Liceul "Ştefan Odobleja", Craiova

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

Pentru descrierea structurilor se utilizează următoarele cuvinte cheie:

- pentru atribuire	ATRIB	
- pentru alternativă	DACA	
	ATUNCI	
	ALTFEL	unde ALTFEL poate lipsi
- pentru selecţie	ALEGE	
	CAZ	
	CAZ_IMPLICIT	unde CAZ_IMPLICIT poate lipsi
- pentru repetitive	1) CAT_TIMP	
	2) REPETA	
	PANA_CAND	
	3) PENTRU	
- pentru compusă	INCEPUT	
	SFARSIT	

Nu se face diferenţă între literele mari şi literele mici. Un algoritm începe cu cuvântul cheie INCEPUT, se termină cu SFARSIT şi nu conţine structuri vide. De asemenea o structură compusă începe cu cuvântul cheie INCEPUT şi se termină cu SFARSIT.

Cerinţă

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

Date de intrare:

Din fişierul text ALGOR.IN se citeşte algoritmul.

Fişierul conţine pe fiecare linie câte un singur cuvânt cheie.

Nu există linii vide.

Algoritmul respectă principiile programării structurate şi este scris corect.

Date de ieşire:

În fişierul ALGOR.OUT se va scrie pe prima linie numărul minim de date de test necesare pentru verificarea algoritmului. Verificarea se bazează pe principiul "cutiei transparente", ceea ce înseamnă că testele se compun astfel încât să fie posibilă executarea algoritmului pe toate ramurile posibile.

De exemplu, în cazul unei structuri repetitive CAT_TIMP care conţine în corpul său un singur ATRIB, un test vizează o execuţie fără să se intre în corpul structurii, altul pentru a trece cel puţin o dată şi prin corpul acestuia.

În mod similar se tratează şi structura PENTRU.

Restricţii şi precizări

• După cuvintele cheie

ATUNCI, ALTFEL,
CAZ, CAZ_IMPLICIT,
CAT_TIMP, REPETA, PENTRU

trebuie să existe obligatoriu o structură de atribuire, alternativă, de decizie, repetitivă sau compusă.

Exemplul 1:

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

Exemplul 2:

ALGOR.IN	ALGOR.OUT	OBS.
INCEPUT ATRIB REPETA inceput atrib atrib SFARSIT pana_cand SFARSIT	1	REPETA se execută cel puţin o dată

Exemplul 3:

ALGOR.IN	ALGOR.OUT	OBS.
INCEPUT ATRIB ALEGE CAZ ATRIB CAZ INCEPUT ATRIB ATRIB SFARSIT SFARSIT	3	- se execută ATRIB de la primul CAZ - se execută ATRIB de la al doilea CAZ - nu se execută nici primul CAZ, nici al doilea

Exemplul 4:

ALGOR.IN	ALGOR.OUT	OBS.
INCEPUT ATRIB ALEGE CAZ ATRIB CAZ ATRIB CAZ_IMPLICIT ATRIB SFARSIT	3	- se execută ATRIB de la primul CAZ - se execută ATRIB de la al doilea CAZ - se execută ATRIB de la CAZ_IMPLICIT

Exemplul 5:

ALGOR.IN	ALGOR.OUT	OBS.
INCEPUT atrib DACA ATUNCI ATRIB ALTFEL ATRIB pentru atrib SFARSIT	4	- se execută ATUNCI şi PENTRU - se execută ATUNCI şi nu se execută PENTRU - se execută ALTFEL şi PENTRU - se execută ALTFEL şi nu se execută PENTRU În total 4 teste.

Timp maxim de execuţie pe test: 1 secundă

7.1.1 Indicaţii de rezolvare - descriere soluţie *

Metoda de rezolvare este *Divide et Impera*. Problema se descompune în instrucţiuni elementare folosind recursivitatea.

7.1.2 Rezolvare detaliată *

```
import java.io.*;
class Algoritm1
{
  static String atom;
  static String fin="algor.in";
  static String fout="algor.out";
  static StreamTokenizer st;
  static BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
  static String pauza;

  public static void main(String[] args) throws IOException
  {
    st=new StreamTokenizer(
    new BufferedReader( new FileReader(fin)));
    st.wordChars((int)'_',(int)'_');

    PrintWriter out=new PrintWriter(
                    new BufferedWriter(new FileWriter(fout)));

    readAtom();
    out.println(nrTeste()); // "inceput" bloc
    out.close();
```

```
}// main

static int nrTeste() throws IOException
{
  int nr, nrTesteAtunci, nrTesteAltfel, nrTesteCaz;
  int nrBlocuriDeschise; // inceput
  boolean implicit;

  String atomr=new String(atom);
  System.out.println(" ***> "+atomr);

  if(atom.equals("inceput")) // inceput bloc
  {
    nr=1; // initializare pentru produsul: nr=nr*nrTeste();
    nrBlocuriDeschise=1;
    readAtom();
    while(nrBlocuriDeschise!=0)
    {
      if(atom.equals("inceput"))      {nrBlocuriDeschise++; readAtom();}
      else if(atom.equals("sfarsit")) {nrBlocuriDeschise--; readAtom();}
      else nr=nr*nrTeste();
    }
  }
  else if(atom.equals("atrib")) { readAtom(); nr=1; }
  else if(atom.equals("daca"))
  {
    readAtom(); // citeste "atunci"
    readAtom(); nrTesteAtunci=nrTeste();
    nrTesteAltfel=1;
    if(atom.equals("altfel")) {readAtom(); nrTesteAltfel=nrTeste();}
    nr=nrTesteAtunci+nrTesteAltfel;
  }
  else if(atom.equals("alege"))
  {
    implicit=false;
    nr=0;
    readAtom();
    while(atom.equals("caz")||atom.equals("caz_implicit"))
    {
      if(atom.equals("caz_implicit")) implicit = true;
      readAtom(); nrTesteCaz=nrTeste();
      nr=nr+nrTesteCaz;
    }
    if(!implicit) nr++;
```

```
    }
    else if(atom.equals("cat_timp")) {readAtom(); nr=nrTeste()+1;}
    else if(atom.equals("repeta"))
    {
      readAtom(); nr=nrTeste();
      readAtom(); // citeste ce urmeaza dupa "pana_cand"
    }
    else if(atom.equals("pentru")) {readAtom(); nr=nrTeste()+1;}
    else nr=1; // la eof

    System.out.println(" <*** "+atomr);
    return nr;
  }

  static void readAtom() throws IOException
  {
    if(st.nextToken()==StreamTokenizer.TT_EOF) atom="eof";
    else atom=st.sval.toString().toLowerCase();

    System.out.println("readAtom() "+atom);
    pauza=br.readLine();
  }
}// class
```

7.1.3 Codul sursă *

```
import java.io.*;
class Algoritm
{
  static String atom;
  static String fin="algor.in";
  static String fout="algor.out";
  static StreamTokenizer st;

  public static void main(String[] args) throws IOException
  {
    st=new StreamTokenizer(
       new BufferedReader(new FileReader(fin)));
    st.wordChars((int)'_',(int)'_');

    PrintWriter out=new PrintWriter(
                    new BufferedWriter(new FileWriter(fout)));
```

```
    readAtom(); // citeste "inceput" bloc program
    out.println(nrTeste());
    out.close();
  }// main

  static int nrTeste() throws IOException
  {
    int nr, nrTesteAtunci, nrTesteAltfel, nrTesteCaz;
    int nrBlocuriDeschise; // inceput
    boolean implicit;

    if(atom.equals("inceput")) // inceput bloc
    {
      nr=1; // initializare pentru produsul: nr=nr*nrTeste();
      nrBlocuriDeschise=1;
      readAtom();
      while(nrBlocuriDeschise!=0)
      {
        if(atom.equals("inceput"))  {nrBlocuriDeschise++; readAtom();}
        else if(atom.equals("sfarsit")) {nrBlocuriDeschise--; readAtom();}
        else nr=nr*nrTeste();
      }
    }
    else if(atom.equals("atrib")) { readAtom(); nr=1; }
    else if(atom.equals("daca"))
    {
      readAtom(); // citeste "atunci"
      readAtom(); nrTesteAtunci=nrTeste();
      nrTesteAltfel=1;
      if(atom.equals("altfel")) {readAtom(); nrTesteAltfel=nrTeste();}
      nr=nrTesteAtunci+nrTesteAltfel;
    }
    else if(atom.equals("alege"))
    {
      implicit=false;
      nr=0;
      readAtom();
      while(atom.equals("caz")||atom.equals("caz_implicit"))
      {
        if(atom.equals("caz_implicit")) implicit = true;
        readAtom(); nrTesteCaz=nrTeste();
        nr=nr+nrTesteCaz;
      }
```

```
      if(!implicit) nr++;
    }
    else if(atom.equals("cat_timp")) {readAtom(); nr=nrTeste()+1;}
    else if(atom.equals("repeta"))
    {
      readAtom(); nr=nrTeste();
      readAtom(); // citeste ce urmeaza dupa "pana_cand"
    }
    else if(atom.equals("pentru")) {readAtom(); nr=nrTeste()+1;}
    else nr=1; // pentru "eof"

    return nr;
  }

  static void readAtom() throws IOException
  {
    if(st.nextToken()==StreamTokenizer.TT_EOF) atom="eof";
    else atom=st.sval.toString().toLowerCase();
  }
}// class
```

7.2 Cod de identificare

prof. Eugen Ionescu, Liceul "Tiberiu Popoviciu", Cluj-Napoca

Pentru a concura cu numărul de serie de la procesoarele Intel Pentium III, Advanced Micro Devices (AMD) a stabilit un sistem de identificare pentru noile procesoare cu numele de cod Thunderbird. Fiecare firmă distribuitoare primeşte o mulţime de litere (de exemplu: $\{a, m, x\}$) din care va trebui să-şi formeze codurile proprii de identificare.

Firmelor li se impune exact de câte ori trebuie să apară fiecare literă în aceste coduri. De exemplu, o firmă trebuie să formeze identificatori care să conţină exact 3 litere a, 2 litere m şi 1 literă x.

Cerinţă

Scrieţi un program care, cunoscând un anumit cod dat, determină următorul cod corect în ordine lexicografică, dacă există un astfel de cod următor.

Date de intrare

Singura linie a fişierului de intrare conţine un cod.

Date de ieşire

Fişierul de ieşire COD.OUT va conţine o singură linie pe care se va afla codul următor; dacă nu există un astfel de cod, atunci în fişier se va scrie "Este ultimul cod."

Restricţii şi precizări

• Codurile sunt formate din cel mult 100 de litere mici ale alfabetului latin.

Exemple:

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

Timp de execuţie: 1 secundă/test.

7.2.1 Indicaţii de rezolvare - descriere soluţie *

Se determină cel mai mare indice i pentru care `cod[i]<cod[i+1]`. Caracterele de pe ultimile poziţii din cod, incepând cu poziţia i inclusiv, se scriu în noul cod în ordine crescătoare.

7.2.2 Rezolvare detaliată *

O primă încercare ca antrenament cu string-uri!

```
import java.io.*;
class Cod1
{
  static String cod;
  static int n;
  static int[] nrl=new int[26];

  public static void main(String[] args) throws IOException
  {
    int i;
    BufferedReader br=new BufferedReader(new FileReader("cod.in"));
    PrintWriter out=new PrintWriter(
                    new BufferedWriter(new FileWriter("cod.out")));
    cod=br.readLine();
    n=cod.length();

    System.out.println(cod);
    for(i=0;i<n;i++) System.out.print(cod.charAt(i)+"\t ");
    System.out.println();
    for(i=0;i<n;i++) System.out.print((byte)cod.charAt(i)+"\t ");
    System.out.println();
    for(i=0;i<n;i++) System.out.print(((byte)cod.charAt(i)-'a')+"\t ");
    System.out.println();
    for(i=0;i<n;i++) nrl[(byte)cod.charAt(i)-'a']++;
```

```
    System.out.println();
    for(i=0;i<26;i++)
    if(nrl[i]>0) System.out.println(i+"\t"+(char)(i+'a')+"\t"+nrl[i]);
    out.close();
  }
}
```

Totuşi, merită lucrat cu vector de caractere!

```
import java.io.*;
class Cod2
{
  static char[] cod;
  static int n;
  static int[] nrl1=new int[26];
  static int[] nrl2=new int[26]; // pentru a evita o sortare !

  public static void main(String[] args) throws IOException
  {
    int i,j,k,kk;
    BufferedReader br=new BufferedReader(new FileReader("cod.in"));
    PrintWriter out=new PrintWriter(
                    new BufferedWriter(new FileWriter("cod.out")));
    cod=br.readLine().toCharArray();
    n=cod.length;

    for(k=0;k<n;k++) System.out.print(cod[k]);
    System.out.println();

    for(i=0;i<n;i++) nrl1[(byte)cod[i]-'a']++;
    for(i=0;i<26;i++) nrl2[i]=nrl1[i];

    i=n-2; // cod[0] ... cod[i] cod[i+1] ... cod[n-2] cod[n-1]
    while((i>=0)&&(cod[i]>=cod[i+1]))
    {
      j=(byte)cod[i+1]-'a';
      nrl2[j]--;
      i--;
    }
    j=(byte)cod[i+1]-'a';
    nrl2[j]--;

    for(k=0;k<i;k++) System.out.print(cod[k]);
    System.out.println();
```

```
    if(i<0) out.println("Este ultimul cod.");
    else
    {
      j=(byte)cod[i]-'a'; // j = "codul" caracterului cod[i]
      nrl2[j]--; // caractere de pus la sfarsit!

      for(k=0;k<26;k++)
      if(nrl2[k]!=nrl1[k])
      System.out.println((char)(k+'a')+" "+nrl2[k]+" "+nrl1[k]);

      for(k=j+1;k<26;k++)  // caut primul caracter > cod[i]
      if(nrl2[k]<nrl1[k])  // si il pun pe pozitia i
      {
        cod[i]=(char)(k+'a');
        nrl2[k]++;
        break;
      }

    for(k=0;k<=i;k++) System.out.print(cod[k]);
    System.out.println();

    for(k=0;k<26;k++)
      if(nrl2[k]!=nrl1[k])
        System.out.println((char)(k+'a')+" "+nrl2[k]+" "+nrl1[k]);

    i++;
    for(k=0;k<26;k++)
      if(nrl2[k]<nrl1[k]) // poate lipsi !
        for(j=nrl2[k];j<nrl1[k];j++)
        {
          cod[i]=(char)(k+'a');
          for(kk=0;kk<=i;kk++) System.out.print(cod[kk]);
          System.out.println();
          i++;
        }
      for(k=0;k<n;k++) out.print(cod[k]);
      out.println();
    }
    out.close();
  }
}
```

Eliminând mesajele de urmărire a execuţiei obţinem versiunea finală.

7.2.3 Codul sursă *

```
import java.io.*;
class Cod3
{
  static char[] cod;
  static int n;
  static int[] nrl1=new int[26];
  static int[] nrl2=new int[26]; // pentru a evita o sortare !

  public static void main(String[] args) throws IOException
  {
    int i,j,k,kk;
    BufferedReader br=new BufferedReader(new FileReader("cod.in"));
    PrintWriter out=new PrintWriter(
                    new BufferedWriter(new FileWriter("cod.out")));
    cod=br.readLine().toCharArray();
    n=cod.length;

    for(i=0;i<n;i++) nrl1[(byte)cod[i]-'a']++;
    for(i=0;i<26;i++) nrl2[i]=nrl1[i];

    i=n-2; // cod[0] ... cod[i] cod[i+1] ... cod[n-2] cod[n-1]
    while((i>=0)&&(cod[i]>=cod[i+1]))
    {
      j=(byte)cod[i+1]-'a';
      nrl2[j]--;
      i--;
    }
    j=(byte)cod[i+1]-'a'; // trebuie si el!
    nrl2[j]--;

    if(i<0) out.println("Este ultimul cod.");
    else
    {
      j=(byte)cod[i]-'a'; // j = "codul" caracterului cod[i]
      nrl2[j]--; // caractere de pus la sfarsit!

      for(k=j+1;k<26;k++)  // caut primul caracter > cod[i]
      if(nrl2[k]<nrl1[k])  // si il pun pe pozitia i
      {
        cod[i]=(char)(k+'a');
        nrl2[k]++;
```

```
        break;
      }

      i++;
      for(k=0;k<26;k++)
        if(nrl2[k]<nrl1[k]) // poate lipsi !
          for(j=nrl2[k];j<nrl1[k];j++)
          {
            cod[i]=(char)(k+'a');
            i++;
          }
      for(k=0;k<n;k++) out.print(cod[k]);
      out.println();
    }
    out.close();
  }
}
```

7.3 Comoara

Mihai Stroe, student, Universitatea Politehnica, Bucureşti

Cei K membri ai unui grup de căutători de comori se află într-un complex dreptunghiular format din camere pătrate de latură 1.

În mod normal toate camerele ar trebui să fie deschise, dar o parte dintre ele sunt închise şi pot fi deschise doar cu ajutorul unor cartele speciale. Astfel, fiecare cameră şi fiecare cartelă are asociat un număr între 0 şi 20; o cameră cu numărul asociat 0 este deschisă de la început, în timp ce una cu numărul asociat diferit de 0 este iniţial închisă, poate fi deschisă din interior sau din exterior dintr-o cameră vecină cu o cartelă cu numărul corespunzător şi va rămâne deschisă ulterior.

Cartelele cu numărul asociat 0 nu au nici o întrebuinţare practică. Un număr poate fi asociat mai multor cartele, respectiv camere. Dintr-o cameră se poate trece în alta numai dacă ele sunt vecine şi ambele sunt deschise. Două camere se consideră vecine dacă au un perete (deci o latură) în comun.

În fiecare cameră se află comori cu o valoare între 0 şi 10000. De asemenea, în fiecare cameră se află exact o cartelă de acces (cu numărul asociat între 0 şi 20); un căutător de comori poate ridica şi folosi cartelele din camerele prin care trece, dar nu le poate da unui alt căutător decât dacă respectivul se află în aceeaşi cameră cu el.

Cerinţă

Cunoscându-se configuraţia complexului şi poziţiile iniţiale ale căutătorilor de comori, să se determine valoarea maximă a comorilor adunate de aceştia.

Datele de intrare

se citesc din fişierul COMOARA.IN care are următorul format:
– pe prima linie dimensiunile complexului, m şi n
– pe următoarele m linii numerele asociate fiecărei camere
– pe următoarele m linii valorile comorilor din fiecare cameră
– pe următoarele m linii numerele asociate cartelelor din fiecare cameră
– pe următoarea linie numărul K al membrilor grupului
– pe următoarele K linii poziţiile iniţiale ale membrilor grupului în complex

Datele de ieşire:
În fişierul COMOARA.OUT se va afişa valoarea totală a comorilor care pot fi strânse de cei K membri ai grupului.

Restricţii şi precizări
• $m, n \leq 40$, $k \leq 10$

Exemple

COMOARA.IN	COMOARA.OUT
3 3 0 0 11 14 0 10 19 0 0 5162 4331 1390 5230 1955 9796 5507 6210 1021 0 0 0 19 0 0 0 0 0 2 3 2 2 1	23909

Observaţie: al doilea căutator de comori nu poate pătrunde în camera $(3, 1)$, deşi are cartela corespunzătoare, pentru că nu poate părăsi camera $(2, 1)$ în care este plasat iniţial.

Timp maxim de execuţie pe test: 1 secundă

7.3.1 Indicaţii de rezolvare - descriere soluţie *

Se foloseşte un algoritm de tip *fill* (umplere) pentru fiecare căutător de comori, în mod repetat, până când nu mai apar îmbunătăţiri ale soluţiei. Pe parcurs, unele camere devin şi rămân deschise. Pentru fiecare căutător de comori se păstrează informaţii referitoare la cheile pe care le are la un moment dat (în momentul în care poate pătrunde într-o cameră, el ia cheia existentă în acea cameră).

7.3.2 Rezolvare detaliată

7.3.3 Codul sursă *

```
import java.io.*;
class Comoara
{
  static int m,n;
  static int[][] codCamera;       // cod camera
  static int[][] valc;            // valoare comoara in camera
  static int[][] cheiaDinCamera;  // cheia din camera
  static int k;                   // nr persoane in grup
  static int[] lin;               // linia initiala persoana
  static int[] col;               // coloana initiala persoana

  static boolean[][] traseu;
  static boolean[][][] amaifost;
  static boolean[][] arecheia;    // ce chei are
  static int s;                   // suma comorilor
  static boolean gata;            // =true daca nu apar imbunatatiri

  static PrintWriter out;
  static StreamTokenizer st;

  public static void main(String[] args) throws IOException
  {
    citire();
    rezolvare();
    afisare();
  }

  static void citire() throws IOException
  {
    int i,j;
    st=new StreamTokenizer(
       new BufferedReader(new FileReader("comoara.in")));
    st.nextToken(); m=(int)st.nval;
    st.nextToken(); n=(int)st.nval;
    codCamera=new int[m+1][n+1];
    valc=new int[m+1][n+1];
    cheiaDinCamera=new int[m+1][n+1];
```

```
    traseu=new boolean[m+1][n+1];

    for(i=1;i<=m;i++)
      for(j=1;j<=n;j++) {st.nextToken(); codCamera[i][j]=(int)st.nval;}
    for(i=1;i<=m;i++)
      for(j=1;j<=n;j++) {st.nextToken(); valc[i][j]=(int)st.nval;}
    for(i=1;i<=m;i++)
      for(j=1;j<=n;j++) {st.nextToken(); cheiaDinCamera[i][j]=(int)st.nval;}

    st.nextToken(); k=(int)st.nval;
    arecheia=new boolean[k+1][21];
    lin=new int[k+1];
    col=new int[k+1];
    amaifost=new boolean[k+1][m+1][n+1];

    for(i=1;i<=k;i++)
    {
      st.nextToken(); lin[i]=(int)st.nval;
      st.nextToken(); col[i]=(int)st.nval;
    }
  }

  static void rezolvare()
  {
    int i;
    s=0;
    for(i=1;i<=k;i++) arecheia[i][0]=true;
    for(i=1;i<=k;i++) amaifost[i][lin[i]][col[i]]=true;

    gata=false;
    while(!gata) alteCautari();
  }

  static void alteCautari()
  {
    int i;
    gata=true;
    for(i=1;i<=k;i++)
    {
      curatTraseu();
      cauta(i,lin[i],col[i]);
    }
  }
```

```
  static void curatTraseu()
  {
    int i,j;
    for(i=1;i<=m;i++) for(j=1;j<=n;j++) traseu[i][j]=false;
  }

  static void cauta(int x, int i, int j)        // alg "fill" (de umplere)
  {
    if((i<1)||(i>m)||(j<1)||(j>n)) return;
    if(traseu[i][j]) return;                    // a mai trecut pe aici
    if((!amaifost[x][i][j])&&                   // nu a mai fost aici
       (!arecheia[x][codCamera[i][j]])) return; // nu are cheia de intrare aici

    traseu[i][j]=true;
    if(!amaifost[x][i][j]) gata=false;          // face o imbunatatire
    amaifost[x][i][j]=true;

    arecheia[x][cheiaDinCamera[i][j]]=true;     // ia cheia din camera
    s=s+valc[i][j];                             // ia valorile
    valc[i][j]=0;                               // raman valori zero

    if(arecheia[x][codCamera[i][j]])            // daca are cheia camerei
    {
      codCamera[i][j]=0;                        // camera ramane deschisa
      cauta(x,i-1,j);
      cauta(x,i+1,j);
      cauta(x,i,j-1);
      cauta(x,i,j+1);
    }
  }

  static void afisare() throws IOException
  {
    out=new PrintWriter(
        new BufferedWriter(new FileWriter("comoara.out")));
    out.println(s);
    out.close();
  }
}
```

7.4 Cuburi

prof. Ovidiu Domşa, Colegiul "Horea, Cloşca şi Crişan", Alba Iulia

Un joc nou este format din n cuburi identice, numerotate de la 1 la n. Fiecare cub are toate cele şase feţe adezive (cu "scai").

Jocul constă în realizarea unui obiect din toate cele n cuburi.

Obiectul iniţial este cubul 1. Un obiect se obţine din obiectul anterior prin alipirea unui nou cub la un cub aflat în obiect, pe una din feţele acestuia.

Pentru alipirea unui cub nou, se extrage un bileţel cu numărul cubului la care se va alipi acesta şi se aruncă un zar care va indica unde va fi lipit cubul nou (sus, jos, la stânga, la dreapta, în faţă sau în spate, poziţii codificate respectiv cu 1, 2, 3, 4, 5, 6, ale cubului din obiect).

Cerinţă

a) Dându-se o succesiune de alipire a cuburilor, verificaţi dacă aceasta este validă

b) Dacă succesiunea este validă, precizaţi dacă obiectul obţinut are forma unui paralelipiped plin, specificând dimesiunile acestuia.

Date de intrare

Fisierul CUBURI.IN conţine pe prima linie valoarea lui n.

Pe următoarele $n-1$ linii, se află o succesiune de aşezare a cuburilor, pentru fiecare cub specificând:

număr_cub biletel zar

valori separate prin câte un spatiu.

Date de ieşire

Fişierul de iesire CUBURI.OUT va conţine pe două linii rezultatele de la punctul *a*) şi respectiv *b*) după cum urmează:

a) Dacă succesiunea este validă, prima linie va contine valoarea 0. În caz contrar, prima linie va contine numărul cubului ce nu poate fi alipit, numărul cubului şi numărul feţei la care nu se poate alipi, precum si codul de eroare:

1 pentru alipire la cub inexistent;

2 pentru alipire pe o faţă ocupată.

b) Dacă obiectul nu este paralelipiped plin, sau dacă succesiunea nu este validă, a doua linie a fişierului va conţine valoarea 0.

Dacă obiectul este paralelipiped plin, a doua linie a fişierului va conţine dimensiunile acestuia, măsurate în număr de cuburi, în ordinea:

număr de cuburi pe directiile sus-jos, stânga-dreapta, faţă-spate

separate prin câte un spatiu.

Restricţii şi precizări

- $2 \leq n \leq 1000$;
- pe o dimensiune se alipesc cel mult 10 cuburi.

Exemple

CUBURI.IN	CUBURI.OUT	CUBURI.IN	CUBURI.OUT
6 2 1 1 3 2 4 5 1 4 6 3 1 4 6 3	0 3 2 1	4 2 1 1 4 2 1 3 2 4	0 0

CUBURI.IN	CUBURI.OUT	CUBURI.IN	CUBURI.OUT
8 2 1 1 5 1 4 6 3 6 7 6 2 3 2 4 4 2 6 8 7 3	6 3 6 1 0	8 2 1 1 4 2 6 3 2 4 6 3 6 7 6 2 5 1 6 8 7 3	8 7 3 2 0

Timp maxim de execuţie: 1 secundă/test

7.4.1 Indicaţii de rezolvare - descriere soluţie *

Identificăm poziţia cubului în obiect prin coordonatele sale din (stanga, jos, faţă). De asemenea, pentru că *pe o dimensiune se alipesc cel mult* 10 *cuburi*, folosim un tablou tridimensional pentru a reţine numărul de ordine al cubului care este plasat în oricare punct al spaţiului în care poate să fie construit corpul respectiv.

7.4.2 Rezolvare detaliată

7.4.3 Codul sursă *

```
import java.io.*;
class Cuburi
{
  static int n;
  static int[][][] ecubul=new int[21][21][21]; // aici e cubul ...
  static int[] x;    // x[i]=coordonata x a cubului i
  static int[] y;    // y[i]=coordonata y a cubului i
  static int[] z;    // z[i]=coordonata z a cubului i
  static boolean[] eplasat;
  static int minx, maxx, miny, maxy, minz, maxz;  // coordonate obiect
```

```
  static final int sus=1, jos=2, stanga=3, dreapta=4, fata=5, spate=6;

  public static void main(String[] args) throws IOException
  {
    int i;
    int cubnou, cubvechi, fatacubvechi;
    boolean ok=true;

    StreamTokenizer st=new StreamTokenizer(
                       new BufferedReader(new FileReader("cuburi.in")));
    PrintWriter out=new PrintWriter(
                    new BufferedWriter(new FileWriter("cuburi.out")));

    st.nextToken(); n=(int)st.nval;
    x=new int[n+1];
    y=new int[n+1];
    z=new int[n+1];
    eplasat=new boolean[n+1];

    x[1]=11; y[1]=11; z[1]=11;  // coordonatele cubului 1
    ecubul[x[1]][y[1]][z[1]]=1; // aici e cubul 1
    eplasat[1]=true;            // cubul 1 este plasat in obiect
    minx=maxx=x[1];
    miny=maxy=y[1];
    minz=maxz=z[1];

    for(i=2;i<=n;i++)
    {
      st.nextToken(); cubnou=(int)st.nval;
      st.nextToken(); cubvechi=(int)st.nval;
      st.nextToken(); fatacubvechi=(int)st.nval;
      if(!eplasat[cubvechi])         // alipire la cub inexisent
      {
        out.println(cubnou+" "+cubvechi+" "+fatacubvechi+" 1");
        out.println("0");           // succesiunea nu e valida
        ok=false;
        break;                      // iese din for ==> nu mai pun "else" !
      }

      x[cubnou]=x[cubvechi];
      y[cubnou]=y[cubvechi];
      z[cubnou]=z[cubvechi];
      switch(fatacubvechi)
      {
```

```
        case sus:     z[cubnou]++; if(z[cubnou]>maxz) maxz++; break;
        case jos:     z[cubnou]--; if(z[cubnou]<minz) minz--; break;
        case stanga:  x[cubnou]--; if(x[cubnou]<minx) minx--; break;
        case dreapta: x[cubnou]++; if(x[cubnou]>maxx) maxx++; break;
        case fata:    y[cubnou]++; if(y[cubnou]>maxy) maxy++; break;
        case spate:   y[cubnou]--; if(y[cubnou]<miny) miny--; break;
        default:      System.out.println("Date de intrare eronate!");
      }
      if(ecubul[x[cubnou]][y[cubnou]][z[cubnou]]!=0)// fata ocupata
      {
        out.println(cubnou+" "+cubvechi+" "+fatacubvechi+" 2");
        out.println("0");   // succesiunea nu e valida
        ok=false;
        break;              // iese din for ==> nu mai pun "else" !
      }
      ecubul[x[cubnou]][y[cubnou]][z[cubnou]]=cubnou;
      eplasat[cubnou]=true;
    }
    if(ok)
      if((maxx-minx+1)*(maxy-miny+1)*(maxz-minz+1)==n)
      {
        out.println("0");
        out.println((maxz-minz+1)+" "+(maxx-minx+1)+" "+(maxy-miny+1));
      }
      else
      {
        out.println("0");
        out.println("0");
      }
    out.close();
  }
}
```

7.5 Fibo

prof. Marinel Şerban, Liceul de Informatică, Iaşi

Considerăm şirul lui Fibonacci: $0, 1, 1, 2, 3, 5, 8, 13, 21, ...$

Dat fiind un număr natural ($n \in \mathbb{N}$), scrieţi acest număr sub formă de sumă de elemente neconsecutive din şirul Fibonacci, fiecare element putând să apară cel mult o dată, astfel încât numărul de termeni ai sumei să fie minim.

Date de intrare

Din fişierul FIB.IN se citeşte de pe prima linie numărul natural n.

Datele de ieşire

În fişierul de ieşire FIB.OUT se vor afişa termeni ai şirului Fibonacci, câte unul pe linie, a căror sumă este n.

Restricţii şi precizări

• n poate avea maxim 80 de cifre

Exemple:

FIB.IN	FIB.OUT	FIB.IN	FIB.OUT
20	2 5 13	8	8

Timp maxim de execuţie pe test: 1 secundă

7.5.1 Indicaţii de rezolvare - descriere soluţie *

Se determină cel mai mare număr din şirul Fibonacci care este mai mic sau egal cu numărul dat. Acesta este primul număr din soluţie. Se determină cel mai mare număr din şirul Fibonacci care este mai mic sau egal cu diferenţa dintre numărul dat şi primul număr din soluţie. Acesta este al doilea număr din soluţie. Se repetă acest algoritm până când numărul rămas devine zero. Se folosesc operaţiile de adunare, scădere şi comparare cu numere mari.

7.5.2 Rezolvare detaliată

7.5.3 Codul sursă *

```
import java.io.*;
class Fibo
{
  static int nf=385; // 81 cifre Fibo
  static int[][] f=new int[nf+1][1];
  static int[] x;
  static PrintWriter out;

  public static void main(String[] args) throws IOException
  {
    int k,i,iy,y;
    BufferedReader br=new BufferedReader(new FileReader("fib.in"));
    out=new PrintWriter(new BufferedWriter(new FileWriter("fib.out")));
```

```
    char[] a=br.readLine().toCharArray();
    int nx=a.length;
    x=new int[nx];

   for(i=0;i<a.length;i++) x[nx-1-i]=a[i]-'0';

    f[0]=nr2v(0);
    f[1]=nr2v(1);
    f[2]=nr2v(1);
    for(k=3;k<=nf;k++) f[k]=suma(f[k-1],f[k-2]);

    while(compar(x,nr2v(0))>0)
    {
      iy=maxFibo(x);
      afisv(f[iy]);
      x=scade(x,f[iy]);
    }
    out.close();
  }

  static int maxFibo(int[] nr)
  {
    int k;
    for(k=1;k<=nf;k++) if (compar(f[k],nr)>0) break;
    return k-1;
  }

  static int compar(int[] a, int[] b) //-1, 0, 1 ... a < = > b
  {
    int na=a.length;
    int nb=b.length;
    if(na>nb) return 1; else if(na<nb) return -1;

    int i=na-1;
    while((i>=0)&&(a[i]==b[i])) i--;
    if(i==-1) return 0;
      else if(a[i]>b[i]) return 1; else return -1;
  }

  static int[] scade(int[] x,int[] y) // z=x-y unde x>=y
  {
    int nx=x.length;
    int ny=y.length;
    int nz=nx;
```

```
    int t,i;
    int[] z=new int[nz];
    int[] yy=new int[nz];
    for(i=0;i<ny;i++) yy[i]=y[i];
    t=0;
    for(i=0;i<nz;i++)
    {
      z[i]=x[i]-yy[i]-t;
      if(z[i]<0) {z[i]+=10; t=1;} else t=0;
    }
    if(z[nz-1]!=0) return z;
    else
    {
      int[] zz=new int[nz-1];
      for(i=0;i<nz-1;i++) zz[i]=z[i];
      return zz;
    }
  }

  static int[] suma(int[] x,int[] y)
  {
    int nx=x.length;
    int ny=y.length;
    int nz;
    if(nx>ny) nz=nx+1; else nz=ny+1;
    int t,i;
    int[] z=new int[nz];
    int[] xx=new int[nz];
    int[] yy=new int[nz];
    for(i=0;i<nx;i++) xx[i]=x[i];
    for(i=0;i<ny;i++) yy[i]=y[i];
    t=0;
    for(i=0;i<nz;i++)
    {
      z[i]=xx[i]+yy[i]+t;
      t=z[i]/10;
      z[i]=z[i]%10;
    }
    if(z[nz-1]!=0) return z;
    else
    {
      int[] zz=new int[nz-1];
      for(i=0;i<nz-1;i++) zz[i]=z[i];
      return zz;
```

```
    }
  }

  static int[] nr2v(int nr)
  {
    int nrr=nr,nc=0,i;
    while(nr!=0) { nc++; nr=nr/10; }
    int[] nrv=new int[nc];
    nr=nrr;
    for(i=0;i<nc;i++) { nrv[i]=nr%10; nr=nr/10; }
    return nrv;
  }

  static void afisv(int[] x)
  {
    int i;
    for(i=x.length-1;i>=0;i--) out.print(x[i]);
    out.println();
  }
}
```

7.6 Kommando

prof. Marinel Şerban, Liceul de Informatică, Iaşi

Într-o clădire cu h etaje sunt deţinuţi, la parter, câţiva prizonieri de către T terorişti. Fiecare etaj al clădirii are $m \times n$ camere identice. Fiecare cameră are un cod numeric (nu neapărat unic) exprimat printr-un număr din intervalul $0 - 255$.

O trupă de komando, formată din K specialişti în luptele antitero, trebuie să elibereze prizonierii. Trupa de komando este paraşutată pe clădire şi încearcă să ajungă la parter. Se cunoaşte locul (x, y) unde fiecare membru al trupei a aterizat pe acoperiş.

Greutatea fiecărui membru al trupei este exprimată în unităţi din intervalul $1 - 255$. Un membru al trupei poate trece "în jos" printr-o cameră a clădirii doar dacă "greutatea lui trece prin camera respectivă", conform următoarei definiţii.

Definiţie: Spunem că "a trece prin b" ($a >> b$) dacă, în reprezentare binară, numărul de cifre 1 a lui a este mai mic sau egal cu numărul de cifre 1 a lui b şi cifrele 1 ale lui a sunt comune cu unele cifre 1 ale lui b.

Exemplu: "44 trece prin 174" ($44 >> 174$) deoarece

$$\begin{aligned} 44 &= 00101100 \\ 174 &= 10101110 \end{aligned}$$

Pentru detectarea unei camere prin care să poată trece, un membru al trupei

de komando se poate, eventual, deplasa cu un "pas" în cele 8 direcţii alăturate poziţiei curente în care a ajuns prin aterizare sau trecerea "în jos". Prin "pas"-ul respectiv se ajunge la una din cele 8 camere vecine. Prizonierii pot fi eliberaţi doar dacă la parter ajung minim T membri ai trupei de komando.

Cerinţă:

Să se determine dacă prizonierii pot fi eliberaţi sau nu, precum şi numărul de membri ai trupei de komando care pot să ajungă la parter.

Date de intrare

Fişierul text KOMMANDO.IN are structura următoare:

• pe prima linie valorile m, n, h, K, T despărţite prin câte un spaţiu, cu semnificaţiile descrise mai sus;

• următoarele h linii reprezintă codurile celor $m \times n$ camere ale unui etaj, despărţite prin câte un spaţiu;

• ultimele K linii ale fişierului conţin greutatea şi coordonatele x şi y a poziţiei de aterizare pe acoperiş ale celor K membri ai trupei de komando, pentru fiecare pe câte o linie, despărţite prin câte un spaţiu:

m	n	h	K	T							
c_{111}	c_{112}	...	c_{11n}	c_{121}	c_{122}	...	c_{12n}	...	c_{1m1}	...	c_{1mn}
...											
c_{h11}	c_{h12}	...	c_{h1n}	c_{h21}	c_{h22}	...	c_{h2n}	...	c_{hm1}	...	c_{hmn}
G_1	x_1	y_1									
...											
G_K	x_K	y_K									

Datele de ieşire:

Fişierul text KOMMANDO.OUT cu structura:

• DA sau NU - pe prima linie;

• numărul de membri ai trupei de komando ajunşi la parter - pe linia a doua.

Restricţii şi precizări

• $2 \leq m, n, h \leq 35 \qquad 1 \leq x_i \leq m \qquad 1 \leq y_i \leq n$

• $1 \leq T, K, G_i \leq 255$

• $0 \leq c_{ijk} \leq 255$

• Toate valorile sunt întregi.

Exemplu:

KOMMANDO.IN	KOMMANDO.OUT
5 5 5 3 2 0 0 0 0 0 0 0 33 0 0 0 0 0 2 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 44 2 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 11 0 0 0 0 0 0 2 22 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 66 2 0 0 0 0 0 7 0 15 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 2 2 3 3 3 4 4	DA 2

Timp maxim de execuţie pe test: 5 secunde

7.6.1 Indicaţii de rezolvare - descriere soluţie *

În fişierul de intrare etajele sunt de sus în jos (primul etaj este lângă acoperiş)!

Pentru fiecare *soldat* se face o parcurgere *în adâncime* (folosind recursivitatea) sau o parcurgere *în lăţime* (eventual folosind o structură de tip *coadă*) pentru un algoritm de tip *fill* (umplere).

7.6.2 Rezolvare detaliată

7.6.3 Codul sursă *

```
import java.io.*; // 35*35*35=42875
class Kommando1 // parcurgere in adancime
{
  static int m,n,h,K,T;
  static int[][][] codCamera;  // cod camera
  static int[] xk;             // x acoperis soldat k
  static int[] yk;             // y acoperis soldat k
  static int[] G;              // greutatea
  static int ns=0;             // nr soldati ajunsi la parter
  static boolean aajunslaparter;

  static PrintWriter out;
  static StreamTokenizer st;

  public static void main(String[] args) throws IOException
  {
```

```
    citire();
    rezolvare();
    afisare();
  }

  static void citire() throws IOException
  {
    int x,y,etaj,k;
    st=new StreamTokenizer(
       new BufferedReader(new FileReader("0.in")));
    st.nextToken(); m=(int)st.nval;
    st.nextToken(); n=(int)st.nval;
    st.nextToken(); h=(int)st.nval;
    st.nextToken(); K=(int)st.nval;
    st.nextToken(); T=(int)st.nval;

    codCamera=new int[h+1][m+1][n+1];
    xk=new int[K+1];
    yk=new int[K+1];
    G=new int[K+1];

    for(etaj=1;etaj<=h;etaj++)
      for(x=1;x<=m;x++)
        for(y=1;y<=n;y++)
        {
          st.nextToken();
          codCamera[etaj][x][y]=(int)st.nval;
        }

    for(k=1;k<=K;k++)
    {
      st.nextToken(); G[k]=(int)st.nval;
      st.nextToken(); xk[k]=(int)st.nval;
      st.nextToken(); yk[k]=(int)st.nval;
    }
  }

  static void rezolvare()  throws IOException
  {
    int soldat;
    for(soldat=1;soldat<=K;soldat++)
    {
      System.out.println(soldat);
      aajunslaparter=false;
```

```
      coboara(soldat,1,xk[soldat],yk[soldat]);
      if(aajunslaparter) ns++;
    }
  }

  static void coboara(int soldat, int etaj, int x, int y)
  {
    System.out.println(soldat+" "+etaj+" "+x+" "+y);
    if((x<1)||(x>m)||(y<1)||(y>n)) return;
    if(aajunslaparter) return;
    if(etaj==h) {aajunslaparter=true; return;}
    int i,j;
    for(i=-1;i<=1;i++)
      for(j=-1;j<=1;j++)
        if((x+i>=1)&&(x+i<=m)&&(y+j>=1)&&(y+j<=n))
          if(trece(G[soldat],codCamera[etaj][x+i][y+j]))
            coboara(soldat, etaj+1, x+i, y+j);
  }

  static boolean trece(int a, int b) // a trece prin b ?
  {
    int k;
    for(k=0;k<=7;k++)
      if(((1<<k)&a)!=0) // pozitia k in a este 1
        if(((1<<k)&b)==0) // pozitia k in b este 0
          return false;
    return true;
  }

  static void afisare() throws IOException
  {
    out=new PrintWriter(
        new BufferedWriter(new FileWriter("kommando.out")));
    if(ns>=T) out.println("DA"); else  out.println("NU");
    out.println(ns);
    out.close();
  }
}

import java.io.*; // optimizare prin inregistrarea traseului
class Kommando2 // totusi, test 10 ==> 13.5 sec ???
{
  static int m,n,h,K,T;
  static int[][][] codCamera;     // cod camera
```

```
  static int[] xk;                // x acoperis soldat k
  static int[] yk;                // y acoperis soldat k
  static int[] G;                 // greutatea
  static int ns=0;                // nr soldati ajunsi la parter
  static boolean aajunslaparter;
  static boolean[][][] traseu;    // traseul soldatului curent

  static PrintWriter out;
  static StreamTokenizer st;

  public static void main(String[] args) throws IOException
  {
    long t1,t2;
    t1=System.currentTimeMillis();

    citire();
    rezolvare();
    afisare();

    t2=System.currentTimeMillis();
    System.out.println("Timp = "+(t2-t1)+" ms");
  }

  static void citire() throws IOException
  {
    int x,y,etaj,k;
    st=new StreamTokenizer(
       new BufferedReader(new FileReader("10.in")));
    st.nextToken(); m=(int)st.nval;
    st.nextToken(); n=(int)st.nval;
    st.nextToken(); h=(int)st.nval;
    st.nextToken(); K=(int)st.nval;
    st.nextToken(); T=(int)st.nval;

    codCamera=new int[h+1][m+1][n+1];
    traseu=new boolean[h+1][m+1][n+1];

    xk=new int[K+1];
    yk=new int[K+1];
    G=new int[K+1];

    for(etaj=1;etaj<=h;etaj++)
      for(x=1;x<=m;x++)
        for(y=1;y<=n;y++)
```

```
        {
          st.nextToken();
          codCamera[etaj][x][y]=(int)st.nval;
        }

    for(k=1;k<=K;k++)
    {
      st.nextToken(); G[k]=(int)st.nval;
      st.nextToken(); xk[k]=(int)st.nval;
      st.nextToken(); yk[k]=(int)st.nval;
    }
  }

  static void rezolvare()  throws IOException
  {
    int soldat;
    for(soldat=1;soldat<=K;soldat++)
    {
      curatTraseu();
      aajunslaparter=false;
      coboara(soldat,1,xk[soldat],yk[soldat]);
      if(aajunslaparter) ns++;
    }
  }

  static void curatTraseu()
  {
    int etaj, x, y;
    for(etaj=1;etaj<=h;etaj++)
      for(x=1;x<=m;x++)
        for(y=1;y<=n;y++)
          traseu[etaj][x][y]=false;
  }

  static void coboara(int soldat, int etaj, int x, int y)
  {
    if((x<1)||(x>m)||(y<1)||(y>n)) return;
    if(aajunslaparter) return;
    if(etaj==h) {aajunslaparter=true; return;}
    if(traseu[etaj][x][y]) return; // a mai trecut pe aici
    traseu[etaj][x][y]=true;

    int i,j;
    for(i=-1;i<=1;i++)
```

```
      for(j=-1;j<=1;j++)
        if((x+i>=1)&&(x+i<=m)&&(y+j>=1)&&(y+j<=n))
          if(trece(G[soldat],codCamera[etaj][x+i][y+j]))
            coboara(soldat, etaj+1, x+i, y+j);
  }

  static boolean trece(int a, int b) // a trece prin b ?
  {
    int k;
    for(k=0;k<=7;k++)
      if(((1<<k)&a)!=0) // pozitia k in a este 1
        if(((1<<k)&b)==0) // pozitia k in b este 0
          return false;
    return true;
  }

  static void afisare() throws IOException
  {
    out=new PrintWriter(
        new BufferedWriter(new FileWriter("kommando.out")));
    if(ns>=T) out.println("DA"); else  out.println("NU");
    out.println(ns);
    out.close();
  }
}

import java.io.*; // parcurgere in latime
class Kommando3 // totusi, test 10 ==> 10.5 sec ???
{
  static int m,n,h,K,T;
  static int[][][] codCamera;     // cod camera
  static int[] xk;                // x acoperis soldat k
  static int[] yk;                // y acoperis soldat k
  static int[] G;                 // greutatea
  static int ns=0;                // nr soldati ajunsi la parter
  static boolean aajunslaparter;
  static boolean[][][] ok;        // soldat poate trece prin camera

  static PrintWriter out;
  static StreamTokenizer st;

  public static void main(String[] args) throws IOException
  {
    long t1,t2;
```

```
    t1=System.currentTimeMillis();

    citire();
    rezolvare();
    afisare();

    t2=System.currentTimeMillis();
    System.out.println("Timp = "+(t2-t1)+" ms");
  }

  static void citire() throws IOException
  {
    int x,y,etaj,k;
    st=new StreamTokenizer(
       new BufferedReader(new FileReader("0.in")));
    st.nextToken(); m=(int)st.nval;
    st.nextToken(); n=(int)st.nval;
    st.nextToken(); h=(int)st.nval;
    st.nextToken(); K=(int)st.nval;
    st.nextToken(); T=(int)st.nval;

    codCamera=new int[h+1][m+1][n+1];
    ok=new boolean[h+1][m+1][n+1];

    xk=new int[K+1];
    yk=new int[K+1];
    G=new int[K+1];

    for(etaj=1;etaj<=h;etaj++)
      for(x=1;x<=m;x++)
        for(y=1;y<=n;y++)
        {
          st.nextToken();
          codCamera[etaj][x][y]=(int)st.nval;
        }

    for(k=1;k<=K;k++)
    {
      st.nextToken(); G[k]=(int)st.nval;
      st.nextToken(); xk[k]=(int)st.nval;
      st.nextToken(); yk[k]=(int)st.nval;
    }
  }
```

```
static void rezolvare()  throws IOException
{
  int soldat,etaj,x,y;
  for(soldat=1;soldat<=K;soldat++)
  {
    System.out.println(soldat);
    curatTraseu();
    aajunslaparter=false;
    etaj=1;
    x=xk[soldat];
    y=yk[soldat];
    coboara(soldat,etaj,x,y); // coboara din (x,y) si vecini
    for(etaj=2;etaj<=h;etaj++)
      for(x=1;x<=m;x++)
        for(y=1;y<=n;y++)
          if(ok[etaj][x][y]) coboara(soldat,etaj,x,y);
    if(aajunslaparter) ns++;
  }
}

static void coboara(int soldat, int etaj, int x, int y)
{
  if(etaj==h) {aajunslaparter=true; return;}
  int i,j;
  for(i=-1;i<=1;i++)
    for(j=-1;j<=1;j++)
      if((x+i>=1)&&(x+i<=m)&&(y+j>=1)&&(y+j<=n))
        if(trece(G[soldat],codCamera[etaj][x+i][y+j]))
          ok[etaj+1][x+i][y+j]=true;
}

static boolean trece(int a, int b) // a trece prin b ?
{
  int k;
  for(k=0;k<=7;k++)
    if(((1<<k)&a)!=0) // pozitia k in a este 1
      if(((1<<k)&b)==0) // pozitia k in b este 0
        return false;
  return true;
}

static void curatTraseu()
{
  int etaj, x, y;
```

```
    for(etaj=1;etaj<=h;etaj++)
      for(x=1;x<=m;x++)
        for(y=1;y<=n;y++)
          ok[etaj][x][y]=false;
  }

  static void afisare() throws IOException
  {
    out=new PrintWriter(
        new BufferedWriter(new FileWriter("kommando.out")));
    if(ns>=T) out.println("DA"); else  out.println("NU");
    out.println(ns);
    out.close();
  }
}

import java.io.*; // parcurgere in latime
class Kommando4   // test 10 ==> 4.9 sec
{                 // se poate si mai bine folosind traseu="coada" !!!
  static int m,n,h,K,T;
  static int[][][] codCamera; // cod camera
  static int[] xk; // x acoperis soldat k
  static int[] yk; // y acoperis soldat k
  static int[] G; // greutatea
  static int ns=0; // nr soldati ajunsi la parter
  static boolean aajunslaparter;
  static boolean[][][] ok; // soldat poate trece prin camera
  static boolean[][][] traseu;// soldat a mai fost aici

  static PrintWriter out;
  static StreamTokenizer st;

  public static void main(String[] args) throws IOException
  {
    long t1,t2;
    t1=System.currentTimeMillis();

    citire();
    rezolvare();
    afisare();

    t2=System.currentTimeMillis();
    System.out.println("Timp = "+(t2-t1)+" ms");
  }
```

```
  static void citire() throws IOException
  {
    int x,y,etaj,k;
    st=new StreamTokenizer(
       new BufferedReader(new FileReader("10.in")));
    st.nextToken(); m=(int)st.nval;
    st.nextToken(); n=(int)st.nval;
    st.nextToken(); h=(int)st.nval;
    st.nextToken(); K=(int)st.nval;
    st.nextToken(); T=(int)st.nval;

    codCamera=new int[h+1][m+1][n+1];
    ok=new boolean[h+1][m+1][n+1];
    traseu=new boolean[h+1][m+1][n+1];

    xk=new int[K+1];
    yk=new int[K+1];
    G=new int[K+1];

    for(etaj=1;etaj<=h;etaj++)
    for(x=1;x<=m;x++)
    for(y=1;y<=n;y++)
    {
      st.nextToken();
      codCamera[etaj][x][y]=(int)st.nval;
    }

    for(k=1;k<=K;k++)
    {
      st.nextToken(); G[k]=(int)st.nval;
      st.nextToken(); xk[k]=(int)st.nval;
      st.nextToken(); yk[k]=(int)st.nval;
    }
  }

  static void rezolvare()  throws IOException
  {
    int soldat,etaj,x,y;
    for(soldat=1;soldat<=K;soldat++)
    {
      curatTraseu();
      aajunslaparter=false;
      etaj=0;
```

```
      x=xk[soldat];
      y=yk[soldat];
      coboara(soldat,etaj,x,y);
      for(etaj=1;etaj<=h;etaj++)
        for(x=1;x<=m;x++)
          for(y=1;y<=n;y++)
            if(ok[etaj][x][y]) coboara(soldat,etaj,x,y);
      if(aajunslaparter) ns++;
    }
  }

  static void coboara(int soldat, int etaj, int x, int y)
  {
    if(etaj==h) {aajunslaparter=true; return;}
    int i,j;
    for(i=-1;i<=1;i++)
      for(j=-1;j<=1;j++)
        if((x+i>=1)&&(x+i<=m)&&(y+j>=1)&&(y+j<=n))
          if(!traseu[etaj+1][x+i][y+j])
          {
            traseu[etaj+1][x+i][y+j]=true;
            if(trece(G[soldat],codCamera[etaj+1][x+i][y+j]))
              ok[etaj+1][x+i][y+j]=true;
          }
  }

  static boolean trece(int a, int b) // a trece prin b ?
  {
    int k;
    for(k=0;k<=7;k++)
      if(((1<<k)&a)!=0) // pozitia k in a este 1
        if(((1<<k)&b)==0) // pozitia k in b este 0
          return false;
    return true;
  }

  static void curatTraseu()
  {
    int etaj, x, y;
    for(etaj=0;etaj<=h;etaj++)
      for(x=1;x<=m;x++)
        for(y=1;y<=n;y++)
        {
          ok[etaj][x][y]=false;
```

```
          traseu[etaj][x][y]=false;
        }
  }

  static void afisare() throws IOException
  {
    out=new PrintWriter(
        new BufferedWriter(new FileWriter("kommando.out")));
    if(ns>=T) out.println("DA"); else  out.println("NU");
    out.println(ns);
    out.close();
  }
}
```

Capitolul 8

ONI 2001 clasa a IX-a

8.1 Ferma

prof. Maria Niţă şi prof. Adrian Niţă, Oradea

Un fermier are un teren care are forma unui tablou dreptunghiular lung de M unităţi şi lat de N unităţi. Pe teren sunt plantaţi din loc în loc copaci, pe care fermierul nu doreşte să-i taie. Dorind să-şi supravegheze cultura, fermierul realizează un mic robot de formă pătrată având latura de 3 unităţi pe care îl poate teleghida prin fermă, parcurgând unitate cu unitate o anumită suprafaţă.

Robotul se poate mişca pe verticală şi pe orizontală dar, nu poate trece peste copaci, nu îi poate distruge, nu se poate roti şi are nevoie pentru mişcare de o suprafaţă corespunzătoare dimensiunii lui.

Cerinţă

Ajutaţi-l pe fermier să determine suprafaţa maximă pe care o poate urmări, folosind acest sistem.

Date de intrare

Fişier de intrare: FERMA.IN

• Linia 1: N M - două numere naturale nenule, separate pritr-un spaţiu, reprezentând numărul de linii (N), respectiv numărul de coloane (M);

• Liniile 2..N+1: $C_1C_2...C_M$ - aceste N linii codifică ferma; fiecare linie conţine câte M caractere (fără să fie separate prin spaţii) având semnificaţia:

'.' - teren liber;

'+' - locul în care este plantat un copac;

'R' - centrul robotului.

Date de ieşire

Fişier de ieşire: FERMA.OUT

• Liniile 1..N: $C_1C_2...C_M$ - aceste N linii codifică modul în care fermierul poate să-şi utilizeze robotul pe terenul său; fiecare linie conţine câte M caractere (fără să fie separate prin spaţii) având semnificaţia:

'.' - teren neacoperit de robot;

'*' - teren ce poate fi verificat de robot;

'+' - loc în care a rămas copacul.

Restricţii

$1 \leq N, M \leq 50$

Exemplu

FERMA.IN

12 11

.	.	.	.	.	.	.	.	.	.	.
.	.	.	+	.	.	.	.	.	+	.
.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.
.	.	.	+	.	.	.	.	.	.	.
.	+	.	.	.	R	.	.	.	.	.
.	.	.	.	.	.	.	.	.	+	.
.	.	+	.	.	.	.	.	.	.	+
.	.	.	.	.	.	+	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	+	.	.	.	.

FERMA.OUT

.	.	.	.	*	*	*	*	*	.	.
.	.	.	+	*	*	*	*	*	+	.
.	.	*	*	*	*	*	*	*	*	*
.	.	*	*	*	*	*	*	*	*	*
.	+	*	*	*	*	*	*	*	*	*
.	.	.	+	*	*	*	*	*	*	*
.	+	.	*	*	*	*	*	*	*	*
.	.	.	*	*	*	*	*	*	+	.
.	.	+	*	*	*	*	*	*	.	+
*	*	*	*	*	*	+	.	.	.	.
*	*	*	*	*	*	.	.	.	.	.
*	*	*	*	*	*	+	.	.	.	.

Timp maxim de executare/test: 3 secunde

8.1.1 Indicaţii de rezolvare - descriere soluţie *

Se foloseşte un algoritm de tip *fill* (umplere) folosind recursivitatea, plecând din poziţia iniţială a robotului şi ţinând cont de restricţiile problemei.

8.1.2 Rezolvare detaliată *

```
import java.io.*;
class Ferma1
{
  static int n,m;
  static int[][] a=new int[51][51]; // ferma
  static PrintWriter out;

  public static void main(String[] args) throws IOException
  {
    int i,j,k;
    int ic,sc;
    out=new PrintWriter(new BufferedWriter(new FileWriter("ferma.out")));
    BufferedReader br=new BufferedReader(new FileReader("ferma.in"));
    StreamTokenizer st=new StreamTokenizer(br);

    st.nextToken(); n=(int)st.nval;
    st.nextToken(); m=(int)st.nval;
    System.out.println("n="+n+" m="+m);

    br.readLine();// citeste LF adica 0A adica 10
    for(i=1;i<=n;i++)
```

```
    {
      for(j=1;j<=m;j++)
      {
        a[i][j]=br.read();
        System.out.print(a[i][j]+" ");
      }
      br.readLine(); // citeste CR LF adica 0D 0A adica 13 10
      System.out.println();
    }

    System.out.println();
    for(i=1;i<=n;i++)
    {
      for(j=1;j<=m;j++) System.out.print((char) a[i][j]+" ");
      System.out.println();
    }
    out.close();
  }
}
```

8.1.3 Codul sursă *

```
import java.io.*;
class Ferma2
{
  static int n,m;
  static int[][] a=new int[51][51]; // ferma codificata
  static PrintWriter out;
  static final int liber=(int)'.',copac=(int)'+',robot=(int)'*';

  public static void main(String[] args) throws IOException
  {
    int i,j,ir=0,jr=0;
    int ic,sc;
    out=new PrintWriter(new BufferedWriter(new FileWriter("ferma.out")));
    BufferedReader br=new BufferedReader(new FileReader("ferma.in"));
    StreamTokenizer st=new StreamTokenizer(br);

    st.nextToken(); n=(int)st.nval;
    st.nextToken(); m=(int)st.nval;

    br.readLine();// citeste LF = 0x0A =10
```

```
    for(i=1;i<=n;i++)
    {
      for(j=1;j<=m;j++) a[i][j]=br.read();
      br.readLine(); // citeste CR LF adica 0D 0A adica 13 10
    }

    for(i=1;i<=n;i++)
      for(j=1;j<=m;j++)
        if(a[i][j]==(int)'R') {ir=i; jr=j; break;};
    for(i=-1;i<=1;i++)
      for(j=-1;j<=1;j++) a[ir+i][jr+j]=robot;

    deplasareDin(ir,jr);
    afism();
    out.close();
  }

  static void deplasareDin(int ir, int jr)
  {
    if(ir-2>=1) // sus
    if((a[ir-2][jr-1]!=copac)&&(a[ir-2][jr]!=copac)&&(a[ir-2][jr+1]!=copac))
    if((a[ir-2][jr-1]==liber)||(a[ir-2][jr]==liber)||(a[ir-2][jr+1]==liber))
    {
      a[ir-2][jr-1]=a[ir-2][jr]=a[ir-2][jr+1]=robot;
      deplasareDin(ir-1,jr);
    }

    if(ir+2<=n) // jos
    if((a[ir+2][jr-1]!=copac)&&(a[ir+2][jr]!=copac)&&(a[ir+2][jr+1]!=copac))
    if((a[ir+2][jr-1]==liber)||(a[ir+2][jr]==liber)||(a[ir+2][jr+1]==liber))
    {
      a[ir+2][jr-1]=a[ir+2][jr]=a[ir+2][jr+1]=robot;
      deplasareDin(ir+1,jr);
    }

    if(jr-2>=1) // stanga
    if((a[ir-1][jr-2]!=copac)&&(a[ir][jr-2]!=copac)&&(a[ir+1][jr-2]!=copac))
    if((a[ir-1][jr-2]==liber)||(a[ir][jr-2]==liber)||(a[ir+1][jr-2]==liber))
    {
      a[ir-1][jr-2]=a[ir][jr-2]=a[ir+1][jr-2]=robot;
      deplasareDin(ir,jr-1);
    }

    if(jr+2<=m) // dreapta
```

```
    if((a[ir-1][jr+2]!=copac)&&(a[ir][jr+2]!=copac)&&(a[ir+1][jr+2]!=copac))
    if((a[ir-1][jr+2]==liber)||(a[ir][jr+2]==liber)||(a[ir+1][jr+2]==liber))
    {
      a[ir-1][jr+2]=a[ir][jr+2]=a[ir+1][jr+2]=robot;
      deplasareDin(ir,jr+1);
    }
  }

  static void afism()
  {
    int i,j;
    for(i=1;i<=n;i++)
    {
      for(j=1;j<=m;j++) out.print((char) a[i][j]);
      out.println();
    }
    out.println();
  }
}
```

8.2 Fracţii

prof. Ovidiu Domşa, Alba Iulia

O proprietate interesantă a fracţiilor ireductibile este că orice fracţie se poate obţine după următoarele reguli:

• pe primul nivel se află fracţia 1/1;

• pe al doilea nivel, în stânga fracţiei 1/1 de pe primul nivel, plasăm fracţia 1/2 iar în dreapta ei fracţia 2/1;

nivelul 1: 1/1
nivelul 2: 1/2 2/1

• pe fiecare nivel k se plasează sub fiecare fracţie i/j de pe nivelul de deasupra, fracţia $i/(i+j)$ în stânga, iar fracţia $(i+j)/j$ în dreapta.

nivelul 1: 1/1
nivelul 2: 1/2 2/1
nivelul 3: 1/3 3/2 2/3 3/1

Cerinţă

Dându-se o fracţie oarecare prin numărătorul şi numitorul său, determinaţi numărul nivelului pe care se află fracţia sau o fracţie echivalentă (având aceeaşi valoare) cu aceasta.

Date de intrare

Fişier de intrare: FRACTII.IN

• Linia 1: N M - două numere naturale nenule, separate printr-un spaţiu, reprezentând numărătorul şi numitorul unei fracţii.

Date de ieşire
Fişier de ieşire: FRACTII.OUT
• Linia 1: *niv* - număr natural nenul, reprezentând numărul nivelului care corespunde fracţiei.

Restricţii
$1 < N, M \leq 2.000.000.000$

Exemple

FRACTII.IN	FRACTII.OUT	FRACTII.IN	FRACTII.OUT
13 8	6	12 8	3

Timp maxim de execuţie: 1 secundă/test

8.2.1 Indicaţii de rezolvare - descriere soluţie *

Se aduce fracţia la o fracţie echivalentă ireductibilă. Nu se coboară în arbore ci se urcă din poziţia fracţiei echivalente până la fracţia 1/1 din vârful arborelui de fracţii.

8.2.2 Rezolvare detaliată

8.2.3 Codul sursă *

```
import java.io.*;
class Fractii
{
  public static void main(String[] args) throws IOException
  {
    int n,m,k,d;
    StreamTokenizer st=new StreamTokenizer(new BufferedReader(
                       new FileReader("fractii.in")));
    PrintWriter out=new PrintWriter(new BufferedWriter(
                    new FileWriter("fractii.out")));
    st.nextToken();n=(int)st.nval;
    st.nextToken();m=(int)st.nval;
    k=0;
    d=cmmdc(n,m);
    n=n/d;
```

```
    m=m/d;
    while((n!=1)||(m!=1))
    {
      k++;
      if(n>m) n=n-m; else m=m-n;
    }
    k++;
    out.println(k);
    out.close();
  }

  static int cmmdc(int a, int b)
  {
    int d,i,c,r;
    if(a>b) {d=a; i=b;} else {d=b; i=a;}
    while(i!=0) {c=d/i; r=d%i; d=i; i=r;}
    return d;
  }
}
```

8.3 Tablou

prof. Rodica Pintea, Bucureşti

Generaţi un tablou bidimensional cu proprietăţile:

• conţine N linii şi N coloane;

• elementele sale sunt numere naturale nenule;

• suma elementelor este egală cu numărul natural nenul S;

• pe nici o linie şi pe nici o coloană nu există două elemente identice;

• diferenţa dintre cel mai mare şi cel mai mic element ale tabloului este minimă.

Date de intrare

Fişier de intrare: TABLOU.IN

• Linia 1: N S - două numere naturale nenule, separate printr-un spaţiu, reprezentând numărul de linii şi de coloane ale tabloului, respectiv valoarea sumei tuturor elementelor din tablou;

Date de ieşire

Fişier de ieşire: TABLOU.OUT

• Linia 1..N: $nr_{11}\ nr_{12}$ ... nr_{1N}
$nr_{21}\ nr_{22}$ ... nr_{2N}
.....
$nr_{N1}\ nr_{N2}$ ... nr_{NN}

pe aceste N linii se vor scrie elementele tabloului, câte o linie din tablou pe o linie din fişier; elementele se vor separa prin câte un spaţiu.

Restricţii şi precizări

- $1 < N \leq 100$
- $0 < S \leq 2^{31}$
- Dacă problema nu are soluţie, în fişierul de ieşire se va scrie cifra 0.
- Dacă problema are mai multe soluţii, în fişier se va scrie una singură.

Exemplu

TABLOU.IN	TABLOU.OUT
3 51	4 6 7 7 4 5 5 7 6

Timp maxim de execuţie: 1 sec/test

8.3.1 Indicaţii de rezolvare - descriere soluţie *

Se determină cel mai mare număr natural x cu proprietatea $n(x + x + 1 + ... + x + n - 1) < s$ şi se plasează cele n numere $x, x + 1, ..., x + n - 1$ pe linii, permutându-le circular de la o linie la alta. Astfel, pe fiecare linie şi fiecare coloană vor fi numere diferite. Se măresc cu câte o unitate, dacă este cazul, cele mai mari numere din matrice până când suma tuturor elementelor matricei este egală cu s.

8.3.2 Rezolvare detaliată *

```
import java.io.*;          // max --> [i][j] cu (i+j)%n==n-1
class Tablou1              // min --> [i][j] cu (i+j)%n==0
{                          // (i+j)%n=k ==> j=?=(n-i+k-1)%n
  static int n,s,x,sp,p;   // p=puncte ramase < n*n
  static int[][] a;

  public static void main(String[] args) throws IOException
  {
    int i,j,k;
    StreamTokenizer st=new StreamTokenizer(new BufferedReader(
                       new FileReader("tablou.in")));
    PrintWriter out=new PrintWriter(new BufferedWriter(
                    new FileWriter("tablou.out")));
    st.nextToken();n=(int)st.nval;
    st.nextToken();s=(int)st.nval;
    a=new int[n][n];
```

```
    x=(2*s-n*n*n+n*n)/(2*n*n);
    System.out.println("x="+x);
    sp=0;
    for(i=0;i<n;i++)
      for(j=0;j<n;j++)
      {
        a[i][j]=x+(i+j)%n;
        sp+=a[i][j];
      }
    p=s-sp;
    System.out.println("sp="+sp+" s="+s+" p="+p);

    System.out.println(); afism();
    k=x+n-1;
    while(p>0)
    {
      i=0;
      while((p>0)&&(i<n)) // k=valoarea pe care o maresc cu 1
      {
        j=(n-i+k-1)%n;
        System.out.println("k="+k+" i="+i+" j="+j+" p="+p);
        a[i][j]++;
        i++; p--;
      }
      System.out.println(); afism();
      k--;
    }
    System.out.println(); afism();
    out.close();
  }

  static void afism()
  {
    int i,j;
    for(i=0;i<n;i++)
    {
      for(j=0;j<n;j++) System.out.print(a[i][j]+" ");
      System.out.println();
    }
  }
}
```

8.3.3 Codul sursă *

```
import java.io.*;         // max --> [i][j] cu (i+j)%n==n-1
class Tablou2             // min --> [i][j] cu (i+j)%n==0
{                         // (i+j)%n=k ==> j=?=(n-i+k-1)%n
  static int n,s,x,sp,p;  // p=puncte ramase < n*n
  static int[][] a;

  public static void main(String[] args) throws IOException
  {
    int i,j,k;
    StreamTokenizer st=new StreamTokenizer(new BufferedReader(
                       new FileReader("tablou.in")));
    PrintWriter out=new PrintWriter(new BufferedWriter(
                    new FileWriter("tablou.out")));
    st.nextToken();n=(int)st.nval;
    st.nextToken();s=(int)st.nval;
    a=new int[n][n];

    x=(2*s-n*n*n+n*n)/(2*n*n);
    sp=0;
    for(i=0;i<n;i++)
    for(j=0;j<n;j++)
    {
      a[i][j]=x+(i+j)%n;
      sp+=a[i][j];
    }
    p=s-sp;
    k=x+n-1;
    while(p>0)
    {
      i=0;
      while((p>0)&&(i<n)) // k=valoarea pe care o maresc cu 1
      {
        j=(n-i+k-1)%n;
        a[i][j]++;
        i++; p--;
      }
      k--;
    }
    for(i=0;i<n;i++)
    {
      for(j=0;j<n;j++) out.print(a[i][j]+" ");
```

```
      out.println();
    }
    out.close();
  }
}
```

8.4 Competiţie dificilă

Angel Proorocu, Bucureşti

La o competiţie au participat N concurenţi. Fiecare dintre ei a primit un număr de concurs astfel încât să nu existe concurenţi cu acelaşi număr.

Numerele de concurs aparţin mulţimii $\{1, 2, ..., N\}$.

Din păcate, clasamentul final a fost pierdut, iar comisia îşi poate aduce aminte doar câteva relaţii între unii participanţi (de genul "participantul cu numărul 3 a ieşit înaintea celui cu numărul 5").

Cerinţă

Şeful comisiei are nevoie de un clasament final şi vă cere să-l ajutaţi determinând primul clasament în ordine lexicografică ce respectă relaţiile pe care şi le aminteşte comisia.

Date de intrare

Fişier de intrare: COMPET.IN

• Linia 1: N M - două numere naturale nenule, reprezentând numărul concurenţilor, respectiv numărul relaţiilor pe care şi le aminteşte comisia;

• Liniile 2..M+1: i j - pe fiecare din aceste M linii se află câte două numere naturale nenule i şi j, având semnificaţia: concurentul cu numărul de concurs i a fost în clasament înaintea concurentului cu numărul de concurs j.

Date de ieşire

Fişier de ieşire: COMPET.OUT

• Linia 1: nr_1 nr_2 ... nr_N - pe această linie se va scrie clasamentul sub forma unui şir de numere naturale nenule, separate prin câte un spaţiu, reprezentând numerele de concurs ale concurenţilor, în ordine de la primul clasat la ultimul.

Restricţii şi precizări

• $1 < N \leq 1000$

• se garantează corectitudinea datelor de intrare şi faptul că există totdeauna o soluţie.

Exemple

COMPET.IN	COMPET.OUT	COMPET.IN	COMPET.OUT
3 1 1 2	1 2 3	4 2 2 1 3 4	2 1 3 4

Timp maxim de execuţie: 1 secundă/test

8.4.1 Indicaţii de rezolvare - descriere soluţie *

Pentru fiecare concurent se determină numărul concurenţilor care se găsesc în clasamentul final în faţa lor. Dintre concurenţii care au în clasamentul final în faţa lor 0 (zero) concurenţi se alege cel cu numărul de concurs cel mai mic. Acesta este primul concurent din soluţie. Tuturor concurenţilor care îl au în faţa lor pe concurentul plasat în soluţie li se scade câte o unitate din numărul concurenţilor pe care îi au în faţa lor. Se repetă acest algoritm până când toţi concurenţii au fost plasaţi în soluţie.

8.4.2 Rezolvare detaliată

8.4.3 Codul sursă *

```
import java.io.*;
class competitie
{
static int n,m;
static int[] a;
static int[] b;
static int[] c;
static int[] inainte;
static boolean[] finalizat;

public static void main(String[] args) throws IOException
{
int i,j,igasit=-1;
StreamTokenizer st=new StreamTokenizer(new BufferedReader(
new FileReader("compet.in")));
PrintWriter out=new PrintWriter(new BufferedWriter(
new FileWriter("compet.out")));
st.nextToken();n=(int)st.nval;
st.nextToken();m=(int)st.nval;
a=new int[m+1];
b=new int[n+1];
c=new int[n+1]; // solutia
inainte=new int[n+1];
finalizat=new boolean[n+1];
```

```
for(i=1;i<=m;i++)
{
st.nextToken();a[i]=(int)st.nval; // a[i] < b[i]
st.nextToken();b[i]=(int)st.nval;
}

for(i=1;i<=m;i++) inainte[b[i]]++;
for(j=1;j<=n;j++)
{
for(i=1;i<=n;i++)
if((!finalizat[i])&&(inainte[i]==0))
{
finalizat[i]=true;
c[j]=i;
igasit=i;
break;
}
for(i=1;i<=m;i++)
if(a[i]==igasit) inainte[b[i]]--;
}

for(i=1;i<=n;i++) out.print(c[i]+" ");
out.close();
}
}
```

8.5 Cuvinte

prof. Maria Niţă şi prof. Adrian Niţă, Oradea

Se consideră o listă având un număr cunoscut de cuvinte. Din acestă listă s-au ales două cuvinte oarecare. Se doreşte transformarea primului cuvânt în cel de-al doilea, trecând prin cuvinte intermediare, existente în lista dată. Trecerea dintr-un cuvânt în altul se poate face folosind următoarele operaţii: schimbarea, adăugarea sau ştergerea unui singur caracter.

Cerinţă

Dându-se o listă de cuvinte şi două cuvinte din aceasta, găsiţi cea mai scurtă secvenţă de operaţii care transformă primul cuvânt în cel de-al doilea folosind operaţiile permise.

Date de intrare

Fişierul de intrare: CUVINTE.IN

• Linia 1: N P Q - trei numere naturale nenule, reprezentând numărul cuvintelor din listă (N), poziţia primului cuvânt în listă (P), respectiv poziţia celui

de-al doilea cuvânt în listă (Q);

• Liniile 2..N+1: *cuvânt* - aceste N linii conţin fiecare câte un cuvânt, aparţinând listei.

Date de ieşire

Fişier de ieşire: CUVINTE.OUT

• Linia 1: M - număr natural, reprezentând numărul minim de paşi necesari pentru a ajunge de la primul cuvânt la al doilea;

• Liniile 2..M+2: $cuvânt_i$ - pe aceste linii apar în ordine cuvintele dintr-o secvenţă ce respectă cerinţa problemei (câte un cuvânt pe linie), inclusiv primul cuvânt al secvenţei.

Restricţii şi precizări

• $2 \leq N \leq 100$

• $1 \leq M, P, Q \leq 100$

• numărul maxim de caractere dintr-un cuvânt este 100;

• dacă nu există soluţie, în fişierul de ieşire se va scrie numărul 0 (zero);

• dacă sunt mai multe secvenţe de lungime minimă, în fişier se va scrie una singură.

Exemple

CUVINTE.IN	CUVINTE.OUT	CUVINTE.IN	CUVINTE.OUT
7 1 5	2	7 1 6	0
car	car	car	
cer	mar	cer	
cerc	mare	cerc	
mar		mar	
mare		mare	
rosu		rosu	
inrosit		inrosit	

Timp maxim de execuţie: 2 secunde/test

8.5.1 Indicaţii de rezolvare - descriere soluţie *

Folosim un tablou bidimensional $(a[i][j])_{1 \leq i,j \leq n}$ cu următoarea semnificaţie: $a[i][j] = 1$ dacă se poate transforma cuvântul i în cuvântul j, şi $a[i][j] = 0$ altfel. Se parcurge în lăţime graful neorientat asociat, plecând de la primul cuvânt, până când se întâlneşte al doilea cuvânt sau nu mai sunt cuvinte care pot fi atinse prin această parcurgere.

8.5.2 Rezolvare detaliată

8.5.3 Codul sursă *

```
import java.io.*;
class Cuvinte
{
  static int n,p,q;
  static String[] cuvant=new String[101]; // cuvintele
  static int[] coada=new int[101];
  static int[] analizat=new int[101];
  static int[] d=new int[101]; // distante catre p
  static int[] pr=new int[101]; // predecesori
  static int[][] a=new int[101][101]; // matrice de adiacenta
  static PrintWriter out;

  public static void main(String[] args) throws IOException
  {
    int i,j,k;
    int ic,sc;
    out=new PrintWriter(new BufferedWriter(new FileWriter("cuvinte.out")));
    StreamTokenizer st=new StreamTokenizer(
                       new BufferedReader(new FileReader("cuvinte.in")));
    st.nextToken(); n=(int)st.nval;
    st.nextToken(); p=(int)st.nval;
    st.nextToken(); q=(int)st.nval;
    for(i=1;i<=n;i++) { st.nextToken();  cuvant[i]=st.sval.toString(); }
    for(i=1;i<n;i++)
      for(j=i+1;j<=n;j++)
        a[i][j]=a[j][i]=trece(cuvant[i],cuvant[j]);

    // drum minim intre p si q
    ic=0; sc=1; coada[ic]=p; // ic==sc ==> coada vida!
    analizat[p]=1;
    while(ic!=sc)
    {
      i=coada[ic]; ic++;
      for(j=1;j<=n;j++)
      {
        if((analizat[j]==0)&&(a[i][j]==1))
        {
```

```
          coada[sc]=j; sc++;
          analizat[j]=1;
          d[j]=d[i]+1;
          pr[j]=i;
          if(j==q) break;
        }
      }
      if(analizat[q]==1) break;
    }
    out.println(d[q]);
    drum(q);
    out.close();
  }

  static void drum(int j)
  {
    if(pr[j]!=0) drum(pr[j]);
    out.println(cuvant[j]);
  }

  static int trece(String x, String y) // lg egale sau diferenta=1
  {
    int k,dif;
    if(Math.abs(x.length()-y.length())>1) return 0;
    String xy;
    if(x.length()<y.length()) {xy=x; x=y; y=xy;} // lg.x >= lg.y

    int nx=x.length(), ny=y.length();
    if(nx==ny)
    {
      dif=0;
      for(k=0;k<nx;k++) if(x.charAt(k)!=y.charAt(k)) dif++;
      if(dif==1) return 1; else return 0;
    }
    else // nx>ny
    {
      k=0;
      while((k<ny)&&(x.charAt(k)==y.charAt(k))) k++;
      if(k==ny) return 1;
      else
      {
        k++;
        while((k<ny)&&(x.charAt(k)==y.charAt(k-1))) k++;
        if(k==ny) return 1; else return 0;
```

```
      }
    }
  }
}
```

8.6 Grup

prof. Emanuela Cerchez şi prof. Marinel Şerban, Iaşi

Administratorul reţelei cu N calculatoare de la SRI împarte, din motive strategice, aceste calculatoare în mai multe grupuri. De fapt, important este doar numărul de grupuri şi numărul de calculatoare din fiecare grup, aşa că împărţirea este descrisă prin şirul numerelor de calculatoare din fiecare grup, ordonat crescător.

Periodic, el procedează la o nouă împărţire pe grupe a calculatoarelor.

Dintre toate împărţirile posibile ale calculatoarelor în grupuri putem alege ca următoare împărţire doar aceea a cărei descriere precede sau succede lexicografic imediat împărţirii curente.

Notă:

Spunem că şirul $x_1\ x_2\ ...\ x_p$ precede lexicografic şirul $y_1\ y_2\ ...\ y_k$ dacă:

a) există un indice j, astfel încât $x_i = y_i$ pentru toţi indicii $i < j$ şi $x_j < y_j$ sau

b) $p < k$ şi $x_i = y_i$ pentru toţi indicii $i \leq p$

Exemple

a) 3 7 2 5 precede lexicografic 3 7 4 1 6 2

b) 1 2 3 precede lexicografic 2

Cerinţă

Dându-se o împărţire în grupe a celor N calculatoare, determinaţi cele două variante candidate pentru împărţirea următoare.

Date de intrare

Fişier de intrare: GRUP.IN

• Linia 1: $N\ k$ - numere naturale nenule, reprezentând numărul total (N) al calculatoarelor din reţea şi numărul (k) de grupe.

• Linia 2: $g_1\ g_2\ \ldots\ g_k$ - numărul calculatoarelor din fiecare grupă.

Date de ieşire

Fişier de ieşire: GRUP.OUT

• p - numărul de grupe din împărţirea care precede lexicografic imediat împărţirea dată;

• $h_1\ h_2\ ...\ h_p$ - numărul de calculatoare din cele p grupe ale împărţirii precedente;

• u - numărul de grupe din împărţirea care succede lexicografic imediat împărţirea dată;

• $t_1\ t_2\ ...\ t_u$ - numărul de calculatoare din cele u grupe ale împărţirii următoare;

Restricţii şi precizări
• $2 \leq N \leq 1000$
• $g_1 + g_2 + ... + g_k = h_1 + h_2 + ... + h_p = t_1 + t_2 + ... + t_u = N$
• $1 \leq g_1 \leq g_2 \leq ... \leq g_k$; $1 \leq h_1 \leq h_2 \leq ... \leq h_p$; $1 \leq t_1 \leq t_2 \leq ... \leq t_u$;
• $1 < k < N$
• $1 \leq p, u \leq N$

Exemplu

GRUP.IN	GRUP.OUT
14 3	3
2 6 6	2 5 7
	2
	2 12

Timp maxim de execuţie: 1 secundă/test

8.6.1 Indicaţii de rezolvare - descriere soluţie *

Fie $g[i]$ numărul calculatoarelor din grupul i ($1 \leq i \leq k$.

Pentru secvenţa precedentă (lexicografic):

Dacă $g[k-1] <= g[k]/2$ atunci secvenţa are $k+1$ grupe. Grupul $g[k]$ se împarte în două grupuri (prima jumătate se plasează pe poziţia k iar a doua jumătate se plasează pe poziţia $k+1$ (dacă $g[k]$ este impar, grupul de pe poziţia $k+1$ va avea cu un calculator mai mult decât grupul de pe poziţia k).

Dacă $g[k-1] > g[k]/2$ atunci se determină cel mai mare indice i astfel încât $g[i]-1 \geq g[i-1]$. Secvenţa va avea $i+1$ grupe. Primele $i-1$ grupe de calculatoare rămân neschimbate. Pe poziţia i plasăm $g[i]-1$ calculatoare iar pe poziţia $i+1$ restul calculatoarelor.

Pentru secvenţa următoare (lexicografic):

Dacă $g[k-1]+1 > g[k]-1$ atunci secvenţa are $k-1$ grupe. Primele $k-2$ grupe rămân neschimbate iar pe poziţia $k-1$ se pun împreună calculatoarele de pe poziţiile $k-1$ şi k (din secvenţa iniţială).

Dacă $g[k-1]+1 \leq g[k]-1$ atunci primele $k-2$ grupuri rămân neschimbate iar calculatoarele din grupurile $k-1$ şi k (din secvenţa iniţială) se vor distribui în grupuri de $g[k-1]+1$ calculatoare (cu excepţia ultimului grup, eventual, care va avea cel puţin $g[k-1]+1$ calculatoare dar nu mai mult de $2g[k-1]+1$ calculatoare).

8.6.2 Rezolvare detaliată

8.6.3 Codul sursă *

```
import java.io.*;
class Grup
{
  static int n,k;
  static int[] g;

  public static void main(String[] args) throws IOException
  {
    int i,j,s,nge;
    StreamTokenizer st=new StreamTokenizer(
                       new BufferedReader(new FileReader("grup.in")));
    PrintWriter out=new PrintWriter(
                    new BufferedWriter(new FileWriter("grup.out")));
    st.nextToken(); n=(int)st.nval;
    st.nextToken(); k=(int)st.nval;
    g=new int[k+1];
    for(i=1;i<=k;i++) { st.nextToken(); g[i]=(int)st.nval; }

    if(g[k-1]<=g[k]/2)
    {
      out.println(k+1); // nr grupe din precedenta
      for(j=1;j<=k-1;j++) out.print(g[j]+" ");
      out.println((g[k]/2)+" "+(g[k]+1)/2);
    }
    else
    {
      s=g[k];
      i=k-1;
      while(g[i]-1<g[i-1]) {s+=g[i]; i--;}
      out.println(i+1); // nr grupe din precedenta
      for(j=1;j<i;j++) out.print(g[j]+" ");
      out.println((g[i]-1)+" "+(s+1));
    }

    if(g[k-1]+1<=g[k]-1)
    {
      nge=(g[k]-1)/(g[k-1]+1);// nr grupe egale la sfarsit
      out.println(k+nge-1); // nr grupe din urmatoarea
      for(j=1;j<=k-2;j++) out.print(g[j]+" ");
      for(j=1;j<=nge;j++) out.print((g[k-1]+1)+" ");
      out.println(g[k-1]+1+(g[k]-1)%(g[k-1]+1));
    }
    else
    {
```

```
      out.println(k-1); // nr grupe din urmatoarea
      for(j=1;j<=k-2;j++) out.print(g[j]+" ");
      out.println((g[k-1]+g[k]));
    }
    out.close();
  }
}
```

Capitolul 9

ONI 2002 clasa a IX-a

9.1 Pentagon

lect. univ. Ovidiu Domşa, Alba Iulia

În urma bombardamentelor din 11 septembrie 2001, clădirea Pentagonului a suferit daune la unul din pereţii clădirii. Imaginea codificată a peretelui avariat se reprezintă sub forma unei matrice cu m linii şi n coloane, ca în figura de mai jos:

1110000111	unde	1 reprezintă zid intact
1100001111		0 zid avariat
1000000011		
1111101111		
1110000111		

Sumele alocate de Bin Laden pentru refacerea Pentagonului vor fi donate celor care vor ajuta americanii să refacă această clădire prin plasarea, pe verticală, a unor blocuri de înălţimi k, $k = 1, ..., m$, şi lăţime 1, în locurile avariate.

Cerinţă:

Pentru o structură dată a unui perete din clădirea Pentagonului, determinaţi numărul minim al blocurilor, de înălţimi $k = 1$, $k = 2$, ..., $k = m$, necesare refacerii clădirii.

Date de intrare:

Fişierul de intrare PENTAGON.IN conţine pe prima linie dimensiunile m şi n ale peretelui clădirii, iar pe următoarele m linii câte o secvenţă de caractere 1 sau 0 de lungime n.

Date de ieşire:

Fişierul PENTAGON.OUT va conţine pe câte o linie, ordonate crescător după k, secvenţe:

k nr

unde k este înalţimea blocului,

iar nr este numărul de blocuri de înălţime k, separate prin câte un spaţiu.

Restricţii şi precizări

- $1 \leq m, n \leq 200$
- nu se vor afişa blocurile de înălţime k, a căror număr este zero (0).

Exemplu

PENTAGON.IN	PENTAGON.OUT
5 10	1 7
1110000111	2 1
1100001111	3 2
1000000011	5 1
1111101111	
1110000111	

Timp maxim de execuţie: 1 secundă/test

9.1.1 Indicaţii de rezolvare - descriere soluţie *

Soluţie prezentată în GInfo 12/6

Rezolvarea acestei probleme este foarte simplă, ea neimplicând decât parcurgerea pe coloane a unei matrice, contorizarea unor secvenţe de zerouri şi păstrarea unei statistici referitoare la aceste secvenţe.

Pentru a determina blocurile necesare, vom determina blocurile pe fiecare coloană (datorită faptului că lăţimea unui bloc este întotdeauna 1 şi toate blocurile sunt dispuse pe verticală, un bloc poate ocupa o singură coloană).

Pentru a determina blocurile de pe o coloană va trebui să determinăm secvenţele de zerouri de pe coloana respectivă. Vom lua în considerare doar secvenţele de lungime maximă pentru a minimiza numărul total de blocuri. De exemplu, dacă avem şase zerouri consecutive, am putea folosi un bloc de lungime 6, dar şi două

blocuri de lungime 5 şi 1, 4 şi 2 etc. Evident, este obligatoriu să folosim un singur bloc pentru ca numărul total al blocurilor utilizate să fie minim.

Aşadar, pentru fiecare coloană vom determina lungimile secvenţelor de zerouri. O secvenţă de zerouri poate începe fie pe prima linie a coloanei, fie în momentul în care întâlnim o linie pe care se află un element cu valoarea 0 în timp ce pe linia anterioară se află un element cu valoarea 1. Secvenţa se va termina fie la terminarea parcurgerii coloanei (se ajunge pe ultima linie a acesteia), fie în momentul în care întâlnim o linie pe care se află un element cu valoarea 1 în timp ce pe linia anterioară se află un element cu valoarea 0.

În momentul detectării terminării unei secvenţe (presupunem că lungimea acesteia este x), numărul blocurilor de lungime x este incrementat. Pentru păstrarea numărului de blocuri se utilizează un şir a, unde a_x indică numărul blocurilor de lungime x.

La sfârşit, vom afişa statistica cerută pe baza datelor păstrate în şirul a. Vor fi afişate toate perechile de forma i a_i care respectă condiţia $a_i \neq 0$.

Analiza complexităţii

Datorită faptului că numărul de elemente care compun o secvenţă de zerouri poate fi determinat pe măsură ce este parcursă secvenţa, întregul algoritm constă într-o singură traversare a matricei. Ordinul de complexitate al unei astfel de traversări este $O(m \cdot n)$, unde m reprezintă numărul de linii ale matricei, iar n reprezintă numărul de coloane.

Pentru citirea datelor, matricea este parcursă o singură dată, deci ordinul de complexitate al acestei operaţii este tot $O(m \cdot n)$.

Pentru afişarea datelor se parcurge o singură dată şirul a; acesta nu poate conţine mai mult de m elemente deoarece nu pot fi folosite blocuri mai înalte decât înălţimea zidului. Aşadar, scrierea soluţiei are ordinul de complexitate $O(m)$.

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(m \cdot n) + O(m \cdot n) + O(m) = O(m \cdot n)$.

9.1.2 Rezolvare detaliată

9.1.3 Codul sursă *

```
import java.io.*;
class Pentagon
{
  static int m,n;
  static String[] a;
  static int[] f;
```

```
  public static void main(String[]args) throws IOException
  {
    int i,j,k;
    BufferedReader br=new BufferedReader(new FileReader("pentagon.in"));
    StreamTokenizer st=new StreamTokenizer(br);

    st.nextToken();m=(int)st.nval;
    st.nextToken();n=(int)st.nval;

    a=new String[m];
    f=new int[n+1]; // frecventa lungimilor blocurilor

    br.readLine(); // altfel nu merge !!!

    for(i=0;i<m;i++) a[i]=br.readLine();

    for(j=0;j<n;j++)
    {
      i=0;
      while(i<m)
      {
        while((i<m)&&(a[i].charAt(j)=='1')) i++;
        k=0;
        while((i<m)&&(a[i].charAt(j)=='0')) {k++; i++;}
        f[k]++;
      }
    }

    PrintWriter out=new PrintWriter(
                    new BufferedWriter(new FileWriter("pentagon.out")));
    for(k=1;k<=n;k++) if(f[k]>0) out.println(k+" "+f[k]);
    out.close();
  }
}
```

9.2 Pod

prof. Marinel Şerban, Iaşi

Între două maluri ale unei văi adânci s-a construit un pod suspendat format din N bucăţi de scândură, legate cu liane.

Vom considera că scândurile sunt numerotate de la 1 la N, începând de pe malul pe care ne aflăm.

În timp unele bucăţi de scândură s-au deteriorat, iar altele chiar au dispărut.

Pentru traversarea podului se ştie că:

– se pot face paşi doar de lungime 1, 2 sau 3;

– scândurile deteriorate sunt nesigure, deci pe ele şi de pe ele se pot face doar paşi de lungime 1.

– evident, nu se poate păşi pe o scândură care lipseşte.

Cerinţă:

Scrieţi un program care să determine numărul de modalităţi de traversare a podului (mai exact, de a ajunge pe celălalt mal), precum şi o soluţie de traversare, dacă o astfel de soluţie există.

Date de intrare:

Fişierul de intrare POD.IN are structura:

POD.IN	Semnificaţie
N	Numărul total de scânduri
$k\ s_1\ s_2\ ...\ s_k$	Numărul de scânduri lipsă şi numerele lor de ordine
$h\ d_1\ d_2\ ...\ d_h$	Numărul de scânduri deteriorate şi numerele lor de ordine

Date de ieşire:

Fişierul de ieşire POD.OUT va conţine pe prima linie valoarea -1 dacă nu este posibil să traversăm podul, respectiv numărul de posibilităţi de a traversa podul, dacă aceasta este posibil.

În cazul în care există soluţii, pe cea de a doua linie va fi afişată o astfel de soluţie, prin indicarea, în ordine, a scândurilor pe care se păşeşte, sub forma:

POD.OUT	Semnificaţie
Nr	Numărul total de posibilităţi
$p_1\ p_2\ ...\ p_m$	Soluţia determinată, prin indicarea în ordine a scândurilor pe care se păşeşte

Restricţii şi precizări:

- $3 \leq N \leq 300$
- $0 \leq k, h \leq N$
- $\{s_1, s_2, ..., s_k\} \subseteq \{2, ...N\}$,
- $\{d_1, d_2, ..., d_h\} \subseteq \{1, 2, ...N\}$;
- $\{s_1, s_2, ..., s_k\} \cap \{d_1, d_2, ..., d_h\} = \emptyset$
- Nr are cel mult 80 de cifre.

Exemple:

pod.in	pod.out	pod.in	pod.out	pod.in	pod.out
5	24	10	48	6	-1
0	3	2 2 7	3 6 8	2 2 4	
0		1 5		1 3	

Timp maxim de execuţie: 1 secundă/test

9.2.1 Indicaţii de rezolvare - descriere soluţie *

Soluţie prezentată în GInfo 12/6

Rezolvarea problemei se bazează pe o variantă simplă a metodei programării dinamice. Se observă foarte uşor că numărul de posibilităţi de a ajunge pe cea de-a i-a scândură depinde doar de numărul de posibilităţi de a ajunge pe cele trei scânduri aflate în faţa ei.

Vom nota cu t_i numărul de posibilităţi de a ajunge pe cea de-a i-a scândură. Vom considera malul opus ca fiind cea de-a $(N + 1)$-a scândură, unde N este numărul scândurilor care formează podul.

Soluţia problemei va fi dată de valoarea t_{N+1}.

În cazul în care cea de-a i-a scândură lipseşte, pe ea nu se poate ajunge, deci vom avea $t_i = 0$.

În cazul în care această scândură există, dar este deteriorată, pe ea se poate ajunge doar de pe scândura precedentă, deci vom avea $t_i = t_{i-1}$.

În cazul în care scândura există şi nu este deteriorată, pe ea se poate ajunge de pe scândura anterioară (chiar dacă este deteriorată) sau de pe oricare dintre precedentele două (dacă nu sunt deteriorate). Pentru a exprima relaţia matematică pentru t_i, vom folosi funcţia s_i definită prin $s_i = t_i$ dacă a i-a scândură există şi nu este deteriorată şi $s_i = 0$ în caz contrar.

Folosind această funcţie, relaţia este $t_i = s_{i-3} + s_{i-2} + t_{i-1}$.

Iniţial, vom avea $t_0 = 1$, deoarece se poate spune că există o singură posibilitate de a ajunge pe malul pe care ne aflăm.

Datorită faptului că valorile t_i pot avea până la 80 de cifre, este necesară simularea operaţiei de adunare pentru numere mari.

Determinarea unei soluţii corecte se poate realiza destul de uşor dacă, la fiecare pas, păstrăm indicele unei scânduri anterioare de pe care se poate trece pe scândura curentă. Acest indice va fi fie cel al scândurii anterioare (dacă aceasta există şi numărul posibilităţilor de a ajunge la ea este nenul), fie al uneia dintre precedentele două (dacă există, nu este deteriorată şi numărul posibilităţilor de a ajunge la ea este nenul).

Dacă valoarea t_{N+1} este nenulă, atunci există cu siguranţă cel puţin o soluţie.

În final, modalitatea de traversare va fi generată cu ajutorul unei tehnici recursive foarte simple.

Analiza complexităţii

Operaţia de adunare a numerelor mari are ordinul de complexitate $O(NCif)$, unde $NCif$ este numărul de cifre al celui mai mare dintre numerele care se adună. Deoarece $NCif$ este cel mult 80, ordinul de complexitate al operaţiei de adunare poate fi considerat a fi $O(80) = 80 \cdot O(1) = O(1)$. Trebuie efectuate cel mult $2 \cdot N$ astfel de adunări, deci operaţia de determinare a valorilor t_i are ordinul de complexitate $O(N) \cdot O(1) = O(N)$.

Pentru reconstituirea drumului, la fiecare pas trebuie păstrat indicele unei scânduri precedente de pe care se poate ajunge pe scândura curentă. Există doar

trei posibilităţi de a ajunge pe scândura curentă, deci ordinul de complexitate al acestei operaţii este $O(1)$. Pentru determinarea scândurii anterioare corespunzătoare fiecărei scânduri din componenţa podului sunt efectuate $O(N)$ astfel de operaţii, deci ordinul de complexitate al operaţiei de determinare a unei modalităţi de traversare este $O(N)$.

Citirea datelor de intrare se realizează într-un timp cu ordinul de compelexitate $O(N)$ deoarece pot exista cel mult $N-1$ scânduri care lipsesc şi cel mult N care sunt deteriorate.

Scrierea datelor de ieşire constă în generarea unei modalităţi de traversare care are lungimea cel mult N şi a numărului modalităţilor de traversare. Deoarece determinarea scândurii precedente se realizează pe baza unor indici păstraţi pentru fiecare scândură în parte, ordinul de complexitate al acestei operaţii este $O(1)$. Datorită faptului că vor fi cel mult N astfel de determinări, ordinul de complexitate al operaţiei de determinare a modalităţii de traversare este $O(N)$. Scrierea numărului posibilităţilor de traversare poate fi considerat a fi o operaţie elementară, deci are ordinul de complexitate $O(1)$. Aşadar, ordinul de complexitate al operaţiei de scriere a datelor de ieşire este O(N) + O(1) = O(N).

În concluzie, algoritmul de rezolvare al acestei probleme are ordinul de complexitate $O(N)+O(N)+O(N)+O(N)=O(N)$.

9.2.2 Rezolvare detaliată *

```
import java.io.*;
class Pod1
{
  static final int buna=0, deteriorata=1, lipsa=2;
  static int ns, nsl,nsd;
  static int nsol;
  static int[] x;
  static int[] y;
  static int[] p;

  public static void main(String[] args) throws IOException
  {
    int i,j;
    StreamTokenizer st=new StreamTokenizer(new BufferedReader(
                       new FileReader("pod.in")));
    PrintWriter out=new PrintWriter(new BufferedWriter(
                    new FileWriter("pod.out")));

    st.nextToken(); ns=(int)st.nval;
    x=new int[ns+2];
```

```
y=new int[ns+2]; // nsol[i] ajunge in i
p=new int[ns+2]; // predecesor

st.nextToken();nsl=(int)st.nval; // lipsa
for(i=1;i<=nsl;i++)
{
  st.nextToken(); j=(int)st.nval;
  x[j]=lipsa;
}

st.nextToken();nsd=(int)st.nval; // deteriorate
for(i=1;i<=nsd;i++)
{
  st.nextToken(); j=(int)st.nval;
  x[j]=deteriorata;
}
afisv(x); System.out.println();

y[1]=0;
if(x[1]!=lipsa) y[1]+=1; // 1 pas de pe mal
afisv(y);

y[2]=0;
if(x[2]==deteriorata) y[2]+=y[1]; // 1 pas din 1
else if(x[2]==buna) y[2]+=y[1]+1; // 1 pas din 1 si 2 pasi de pe mal
if(y[2]>0)  if(y[1]>0) p[2]=1;
            else p[2]=0;
afisv(y);

y[3]=0;
if(x[3]==deteriorata)
{
  if(x[2]!=lipsa) y[3]=y[2]+1; // 1 pas din 2
  if(x[1]==buna)  y[3]=y[3]+1; // 2 pasi din 1
}
else if(x[3]==buna)
      {
        y[3]+=1;                    // 3 pasi de pe mal;
        if(x[1]==buna)  y[3]+=y[1]; // 2 pasi din 1
        if(x[2]!=lipsa) y[3]+=y[2]; // 1 pas  din 2
      }
afisv(y); System.out.println();

for(i=4;i<=ns+1;i++)
```

```
    {
      switch(x[i])
      {
        case lipsa:
               y[i]=0; break;
        case deteriorata:
               if(x[i-1]!=lipsa) y[i]=y[i-1];  // 1 pas din i-1
               break;
        case buna:
               y[i]=0;                         // pentru suma
               if(x[i-3]==buna) y[i]+=y[i-3]; // 3 pasi din i-3
               if(x[i-2]==buna) y[i]+=y[i-2]; // 2 pasi din i-2
               if(x[i-1]!=lipsa) y[i]+=y[i-1]; // 1 pas  din i-1
    }
    afisv(y);
  }
}// main

  static void afisv(int[] a)
  {
    int i;
    for(i=1;i<=ns+1;i++) System.out.print(a[i]+" ");
    System.out.println();
  }
}// class

import java.io.*;
class Pod2
{
  static final int buna=0, deteriorata=1, lipsa=2;
  static int ns, nsl,nsd;
  static int nsol;
  static int[] x;
  static int[] y;
  static int[] p;

  public static void main(String[] args) throws IOException
  {
    int i,j;
    StreamTokenizer st=new StreamTokenizer(new BufferedReader(
                       new FileReader("pod.in")));
    PrintWriter out=new PrintWriter(new BufferedWriter(
                    new FileWriter("pod.out")));
```

```
st.nextToken();ns=(int)st.nval;
x=new int[ns+2];
y=new int[ns+2]; // nsol[i] ajunge in i
p=new int[ns+2]; // predecesor

st.nextToken();nsl=(int)st.nval; // lipsa
for(i=1;i<=nsl;i++)
{
  st.nextToken(); j=(int)st.nval;
 x[j]=lipsa;
}

st.nextToken();nsd=(int)st.nval; // deteriorate
for(i=1;i<=nsd;i++)
{
  st.nextToken(); j=(int)st.nval;
  x[j]=deteriorata;
}
afisv(x); System.out.println();

y[1]=0;
if(x[1]!=lipsa) y[1]+=1; // 1 pas de pe mal
afisv(y);

y[2]=0;
if(x[2]==deteriorata) y[2]+=y[1]; // 1 pas din 1
else if(x[2]==buna) y[2]+=y[1]+1; // 1 pas din 1 si 2 pasi de pe mal
if(y[1]>0)  p[2]=1;
afisv(y);

y[3]=0;
if(x[3]==deteriorata)
{
  if(x[2]!=lipsa) y[3]=y[2]+1; // 1 pas din 2
  if(x[1]==buna)  y[3]=y[3]+1; // 2 pasi din 1
}
else if(x[3]==buna)
{
  y[3]+=1; // 3 pasi de pe mal;
  if(x[1]==buna)  y[3]+=y[1]; // 2 pasi din 1
  if(x[2]!=lipsa) y[3]+=y[2]; // 1 pas  din 2
}
afisv(y); System.out.println();
```

```
    if(y[1]>0) p[3]=1;
    else  if(y[2]>0) p[3]=2;

    for(i=4;i<=ns+1;i++)
    {
      switch(x[i])
      {
        case lipsa:
               y[i]=0; break;
        case deteriorata:
               if(x[i-1]!=lipsa) y[i]=y[i-1];// 1 pas din i-1
               if(y[i-1]>0) p[i]=i-1;
               break;
        case buna:
               y[i]=0;                          // pentru suma
               if(x[i-3]==buna) y[i]+=y[i-3]; // 3 pasi din i-3
               if(x[i-2]==buna) y[i]+=y[i-2]; // 2 pasi din i-2
               if(x[i-1]!=lipsa) y[i]+=y[i-1]; // 1 pas  din i-1
               if(y[i-3]>0) p[i]=i-3;
               else if(y[i-2]>0) p[i]=i-2;
                    else if(y[i-1]>0) p[i]=i-1;
      }
      afisv(y);
    }

    System.out.println();
    drum(ns+1);
    System.out.println();
}// main

  static void drum(int j)
  {
    if(p[j]!=0) drum(p[j]);
    System.out.print(j+" ");
  }

  static void afisv(int[] a)
  {
    int i;
    for(i=1;i<=ns+1;i++) System.out.print(a[i]+" ");
    System.out.println();
  }
}// class
```

9.2.3 Codul sursă *

```
import java.io.*;
class Pod3
{
  static final int buna=0, deteriorata=1, lipsa=2;
  static int ns, nsl,nsd;
  static int nsol;
  static int[] x; // tip scandura
  static int[][] y; // y[i]=nr variante de a ajunge in i
  static int[] p; // p[i]=predecesorul lui i
  static PrintWriter out;

  public static void main(String[] args) throws IOException
  {
    int i,j;
    StreamTokenizer st=new StreamTokenizer(new BufferedReader(
                       new FileReader("pod.in")));
    out=new PrintWriter(new BufferedWriter(
        new FileWriter("pod.out")));

    st.nextToken();ns=(int)st.nval;
    x=new int[ns+2];
    y=new int[ns+2][1]; // nsol[i] ajunge in i
    p=new int[ns+2]; // predecesor

    st.nextToken();nsl=(int)st.nval; // lipsa
    for(i=1;i<=nsl;i++)
    {
      st.nextToken(); j=(int)st.nval;
      x[j]=lipsa;
    }

    st.nextToken();nsd=(int)st.nval; // deteriorate
    for(i=1;i<=nsd;i++)
    {
      st.nextToken(); j=(int)st.nval;
      x[j]=deteriorata;
    }

    if(x[1]!=lipsa) y[1]=nrv(1); // 1 pas de pe mal

    y[2]=nrv(0);
```

```
if(x[2]==deteriorata) y[2]=suma(y[2],y[1]); // 1 pas din 1
else if(x[2]==buna)
     y[2]=suma(y[2],suma(y[1],nrv(1)));
if(ok(y[1]))  p[2]=1;

y[3]=nrv(0);
if(x[3]==deteriorata)
{
  if(x[2]!=lipsa) y[3]=suma(y[2],nrv(1)); // 1 pas din 2
  if(x[1]==buna)  y[3]=suma(y[3],nrv(1)); // 2 pasi din 1
}
else if(x[3]==buna)
{
  y[3]=suma(y[3],nrv(1));               // 3 pasi de pe mal;
  if(x[1]==buna)  y[3]=suma(y[3],y[1]); // 2 pasi din 1
  if(x[2]!=lipsa) y[3]=suma(y[3],y[2]); // 1 pas  din 2
}

if(ok(y[1])) p[3]=1;
else  if(ok(y[2])) p[3]=2;

for(i=4;i<=ns+1;i++)
{
  switch(x[i])
  {
    case lipsa:
           y[i]=nrv(0); break;
    case deteriorata:
           if(x[i-1]!=lipsa) y[i]=suma(nrv(0),y[i-1]);// 1 pas din i-1
           if(ok(y[i-1])) p[i]=i-1;
           break;
    case buna:
           y[i]=nrv(0); // pentru suma
           if(x[i-3]==buna) y[i]=suma(y[i],y[i-3]); // 3 pasi din i-3
           if(x[i-2]==buna) y[i]=suma(y[i],y[i-2]); // 2 pasi din i-2
           if(x[i-1]!=lipsa) y[i]=suma(y[i],y[i-1]); // 1 pas  din i-1
           if(ok(y[i-3])) p[i]=i-3;
           else if(ok(y[i-2])) p[i]=i-2;
                else if(ok(y[i-1])) p[i]=i-1;
  }
}
afisv(y[ns+1]);
drum(ns+1);
out.println();
```

```
    out.close();
  }// main

  static boolean ok(int[] z)
  {
    int i;
    for(i=0;i<z.length;i++) if(z[i]!=0) return true;
    return false;
  }

  static void drum(int j)
  {
    if(p[j]!=0) drum(p[j]);
    out.print(j+" ");
  }

  static void afisv(int[] x)
  {
    int nx=x.length;
    int i;
    for(i=nx-1;i>=0;i--) out.print(x[i]);
    out.println();
  }

  static int[] nrv(int nr)
  {
    int nc;
    int nrrez=nr;
    nc=0;
    while(nr!=0) {nc++; nr=nr/10;}
    int[] x=new int[nc];
    nr=nrrez;
    nc=0;
    while(nr!=0) { x[nc]=nr%10; nc++; nr=nr/10; }
    return x;
  }

  static int[] suma(int[] x,int[] y)
  {
    int k,s,t;
    int nx=x.length;
    int ny=y.length;
    int nz;
    if(nx>ny)nz=nx+1; else nz=ny+1;
```

```
    int[] z=new int[nz];
    t=0;
    for(k=0;k<nz;k++)
    {
      s=t;
      if(k<nx) s=s+x[k];
      if(k<ny) s=s+y[k];
      z[k]=s%10;
      t=s/10;
    }
    if(z[nz-1]!=0)return z;
    else
    {
      int[] zz=new int[nz-1];
      for(k=0;k<nz-1;k++) zz[k]=z[k];
      return zz;
    }
  }
}// class
```

9.3 Suma

Florin Gheţu, Bucureşti

Fie şirul tuturor numerelor naturale de la 1 la un număr oarecare N.

Considerând asociate câte un semn (+ sau -) fiecărui număr şi adunând toate aceste numere cu semn se obţine o sumă S.

Problema constă în a determina pentru o sumă dată S, numărul minim N pentru care, printr-o asociere de semne tuturor numerelor de la 1 la N, se poate obţine S.

Cerinţă:

Pentru un S dat, găsiţi valoarea minimă N şi asocierea de semne numerelor de la 1 la N pentru a obţine S în condiţiile problemei.

Restricţii:

• $0 < S \leq 100.000$.

Date de intrare

În fişierul SUMA.IN se va afla pe prima linie un întreg pozitiv S reprezentând suma ce trebuie obţinută.

Date de ieşire

În fişierul SUMA.OUT se va scrie, pe prima linie numărul minim N pentru care se poate obţine suma S iar pe următoarele linii, până la sfârşitul fişierului, numerele care au semn negativ, câte unul pe linie.

Ordinea de afişare a numerelor nu are importanţă.
Celelalte numere care nu apar în fişier se consideră pozitive.
Dacă există mai multe soluţii se cere doar una.

Exemplu:

SUMA.IN	SUMA.OUT
12	7 1 7

Deci suma 12 se poate obţine din minimum 7 termeni astfel:

$$12 = -1 + 2 + 3 + 4 + 5 + 6 - 7.$$

Nu este singura posibilitate de asociere de semne termenilor de la 1 la 7.

Timpul de execuţie: 1 secundă/test

9.3.1 Indicaţii de rezolvare - descriere soluţie *

Soluţie prezentată în GInfo 12/6

Pentru ca numărul termenilor să fie minim, trebuie ca suma termenilor care apar cu semn negativ să fie cât mai mică posibil.

Iniţial vom presupune că nu va trebui să scădem nici o valoare şi ne propunem să găsim cel mai mic număr N pentru care

$$s_N = \sum_{i=1}^{N} i \geq S.$$

Valoarea N se obţine rezolvând ecuaţia de gradul II $S_N = S$ (vom avea $S_N \leq S + N$) şi rotunjind prin adaos rădăcina pozitivă obţinută. Avem:

$$\frac{N\cdot(N+1)}{2} = S \Rightarrow N^2 + N - 2 \cdot S = 0 \Rightarrow N = \frac{\sqrt{1+8\cdot S}-1}{2}$$

Aşadar, valoarea N este $\left\lceil \frac{\sqrt{1+8\cdot S}-1}{2} \right\rceil$. Datorită faptului că N este cea mai mică valoare pentru care suma atinge sau depăşeşte valoarea S, soluţia problemei nu poate fi dată de nici un număr $N' < N$.

În cele ce urmează vom demonstra că soluţia problemei este dată de N, $N+1$ sau $N+2$. Se observă că numerele S_N, S_{N+1} şi S_{N+2} nu pot avea toate aceeaşi paritate, deoarece $N+1$ şi $N+2$ au parităţi diferite. Vom nota prin D_N diferenţa dintre valoarea S_N şi valoarea care trebuie obţinută (S). Deoarece se scade aceeaşi valoare din S_N, S_{N+1} şi S_{N+2} este evident că D_N, D_{N+1} şi D_{N+2} nu pot avea toate aceeaşi paritate.

Soluţia problemei este dată de indicele celui mai mic număr par dintre aceste trei numere; fie acest indice N'.

Suma totală a elementelor cărora trebuie să le fie modificat semnul este $D_{N'}/2$. Prin modificarea semnului unui element, valoarea expresiei scade cu dublul acestui element, deci avem $S_{N'} - D_{N'}/2 \cdot 2 = S$.

Fie $x = D_{N'}/2$ (suma totală a elementelor care trebuie scăzute). Deoarece $N' < N + 2$, obţinem $S_{N'} \leq SN + N + 1 + N + 2$. Folosind relaţia $S_N \leq S + N$, se obţine relaţia $S_{N'} \leq S + 3 \cdot N + 3$. Cu alte cuvinte, avem $x \leq \frac{3\cdot N+3}{2} \leq \frac{3\cdot N'}{2}$.

Aşadar, valoarea x poate fi obţinută alegând doar două numere cuprinse între 1 şi N'.

În cazul în care $x > N'$ vom scădea N' şi $N' - x$, iar în caz contrar vom scădea doar x.

Analiza complexităţii

Valoarea N poate fi obţinută folosind o simplă formulă matematică, deci ordinul de complexitate al acestei operaţii este $O(1)$.

Valorile S_N, S_{N+1}, S_{N+2}, D_N, D_{N+1}, D_{N+2} sunt calculate tot cu ajutorul unor formule matematice simple, aşadar ordinul de complexitate al operaţiei de determinare a acestora este tot $O(1)$.

Identificarea valorii N' se face pe baza verificării parităţii a cel mult trei numere, deci şi această operaţie are ordinul de complexitate $O(1)$.

Determinarea valorii x şi a celor cel mult două numere cărora li se va modifica semnul este realizată tot pe baza unor calcule simple al căror ordin de complexitate este $O(1)$.

În concluzie, ordinul de complexitate al unui algoritm eficient de rezolvare a acestei probleme este $O(1) + O(1) + O(1) + O(1) = O(1)$.

9.3.2 Rezolvare detaliată

9.3.3 Codul sursă *

```
import java.io.*;
class Suma
{
public static void main(String[] args) throws IOException
{
int n=0,suma=0,s;
int np;
StreamTokenizer st=new StreamTokenizer(
new BufferedReader(new FileReader("suma.in")));
PrintWriter out=new PrintWriter(
new BufferedWriter(new FileWriter("suma.out")));
st.nextToken(); s=(int)st.nval;

while((suma<s)||((suma-s)%2==1))
{
n++;
suma=suma+n;
}
```

```
out.println(n);
np=(suma-s)/2;
if(np>0)
if(np<=n) out.println(np);
else out.println((np-n)+"\n"+n);
out.close();
  }
}
```

9.4 Becuri

prof. Cristina Bărbieru, Timişoara

Un panou publicitar, de formă dreptunghiulară conţine becuri, unul lângă altul, aliniate pe linii şi coloane.

Fiecare linie şi fiecare coloană are un comutator care schimbă starea tuturor becurilor de pe acea linie sau coloană, din starea în care se află în starea opusă (stins/aprins, aprins/stins). Asupra unui bec nu se poate acţiona individual. Iniţial panoul are toate becurile stinse.

Cerinţă:

Să se realizeze un program care acţ ionând asupra unui număr minim de linii şi coloane aduce panoul din starea iniţială, la o configuraţie dată, dacă acest lucru este posibil.

Datele de intrare

se vor citi din fişierul BECURI.IN, care are următoarea configuraţie:

• pe prima linie două numere naturale, m şi n, separate prin spaţiu, reprezentând numărul de linii şi de coloane ale panoului;

• pe următoarele m linii câte n coloane, formate din elementele 0 sau 1, separate prin spaţiu, reprezentând configuraţia finală a panoului. 1 este codificarea unui bec aprins iar 0 a unui bec stins.

Datele de ieşire

se va afişa o soluţie în fişierul BECURI.OUT astfel:

• pe prima linie a fişierului indicii coloanelor asupra cărora s-a acţionat, separaţi prin spaţiu.

• pe a doua linie a fişierului indicii linilor asupra cărora s-a acţionat, separaţi prin spaţiu.

Dacă nu se acţionează asupra liniilor sau coloanelor se va afişa 0 pe linia corespunzătoare.

Numerotarea liniilor, respectiv coloanelor începe de la 1.

Dacă problema nu are soluţie, pe prima linie a fişierului se va afişa -1.

Restrictii

• $m \leq 100, n \leq 100$

Exemplu:

BECURI.IN	BECURI.OUT
5 6	2 5
1 0 1 1 0 1	1 2 3
1 0 1 1 0 1	
1 0 1 1 0 1	
0 1 0 0 1 0	
0 1 0 0 1 0	

Timp maxim de execuţie: 1 secundă/test

9.4.1 Indicaţii de rezolvare - descriere soluţie *

Soluţie prezentată în GInfo 12/6

Este evident că pentru a obţine prima linie a matricei nu putem decât fie să comutăm toate coloanele pe care trebuie să se afle becuri aprinse şi să nu comutăm prima linie, fie să comutăm prima linie şi să comutăm toate coloanele pe care trebuie să se afle becuri stinse.

Analog, pentru prima coloană putem fie să comutăm toate liniile pe care trebuie să se afle becuri aprinse şi să nu comutăm prima coloană, fie să comutăm prima coloană şi să comutăm toate liniile pe care trebuie să se afle becuri stinse.

Astfel avem patru posibilităţi de a obţine configuraţia cerută:

• comutăm liniile cărora le corespund becuri aprinse pe prima coloană şi coloanele cărora le corespund becuri aprinse pe prima linie;

• comutăm liniile cărora le corespund becuri aprinse pe prima coloană şi coloanele cărora le corespund becuri stinse prima linie;

• comutăm liniile cărora le corespund becuri stinse pe prima coloană şi coloanele cărora le corespund becuri aprinse pe prima linie;

• comutăm liniile cărora le corespund becuri stinse pe prima coloană şi coloanele cărora le corespund becuri stinse pe prima linie.

Vom verifica fiecare dintre aceste patru variante şi apoi o vom alege pe cea care implică cele mai puţine comutări de linii şi coloane.

Dacă nici una dintre cele patru variante nu duce la obţinerea configuraţiei cerute, putem trage concluzia că aceasta nu poate fi obţinută prin comutarea unor linii şi a unor coloane.

Analiza complexităţii

Citirea datelor de intrare implică parcurgerea unei matrice pătratice, deci ordinul de complexitate al acestei operaţii este $O(N^2)$.

Verificarea fiecăreia dintre cele patru posibilităţi de a obţine soluţia implică parcurgerea primei linii şi a primei coloane; ordinul de complexitate al unei astfel de parcurgere este $O(N) + O(N) = O(N)$.

Pentru fiecare element este posibil ca linia/coloana corespunzătoare să fie comutată. Operaţia de comutare implică traversarea liniei sau coloanei, aşadar are

ordinul de complexitate $O(N)$. Ca urmare, ordinul de complexitate al fiecăreia dintre cele patru posibilităţi este $O(N) \cdot O(N) = O(N^2)$. Întreaga operaţie de determinare a soluţiei are ordinul de complexitate $4 \cdot O(N^2) = O(N^2)$.

Scrierea datelor de ieşire implică traversarea şirurilor care indică dacă o linie sau coloană a fost comutată; ordinul de complexitate al operaţiei este $O(N)$.

În concluzie, algoritmul de rezolvare al acestei probleme are ordinul de complexitate $O(N^2) + O(N^2) + O(N) = O(N^2)$.

9.4.2 Rezolvare detaliată *

Prima fază: pun zero pe prima linie şi verific.

```
import java.io.*;
class Becuri1
{
  static int m,n;
  static byte[][] a;
  static byte[][] b;
  static byte[] colc;
  static byte[] linc;
  static byte[] colcsol;
  static byte[] lincsol;
  static int ncolcsol=234, nlincsol=234;

  public static void main(String[]args) throws IOException
  {
    int i,j,k;
    BufferedReader br=new BufferedReader(new FileReader("becuri.in"));
    StreamTokenizer st=new StreamTokenizer(br);

    st.nextToken(); m=(int)st.nval;
    st.nextToken(); n=(int)st.nval;

    a=new byte[m+1][n+1];
    b=new byte[m+1][n+1];
    colc=new byte[n+1];
    colcsol=new byte[n+1];
    linc=new byte[m+1];
    lincsol=new byte[m+1];

    for(i=1;i<=m;i++)
      for(j=1;j<=n;j++)
        { st.nextToken(); a[i][j]=(byte)st.nval; }
```

```
    fac0Linia1();
    // .... fac1Linia1();

    System.out.println(ncolcsol+" "+nlincsol);

    PrintWriter out=new PrintWriter(
                    new BufferedWriter(new FileWriter("becuri.out")));
    out.close();
  }

  static void fac0Linia1()
  {
    int i,j;
    int nlc=0, ncc=0; // nr linii/coloane comutate
    boolean ok=true;

    for(j=1;j<=n;j++) colc[j]=0; // coloane necomutate inca
    for(i=1;i<=m;i++) linc[i]=0; // linii necomutate inca
    for(i=1;i<=m;i++) for(j=1;j<=n;j++) b[i][j]=a[i][j]; // copie

    for(j=1;j<=n;j++)
    if(b[1][j]==1)
    {
      comutaColoana(j);
      colc[j]=1; // coloana j este comutata;
      ncc++;
    }
    afisb();
    for(i=2;i<=m;i++)
      if(!okLinia(i)) {ok=false; break;}
      else if(b[i][1]==1) { linc[i]=1; nlc++; }
    if(ok&&(nlc+ncc<nlincsol+ncolcsol))
    {
      nlincsol=nlc;
      ncolcsol=ncc;
      for(i=1;i<=m;i++) lincsol[i]=linc[i];
      for(j=1;j<=n;j++) colcsol[j]=colc[j];
    }
  }

  static boolean okLinia(int i)
  {
    int j;
    for(j=2;j<=n;j++) if(b[i][j]!=b[i][1]) return false;
```

```
    return true;
  }

  static void comutaColoana(int j)
  {
    int i;
    for(i=1;i<=m;i++) b[i][j]=(byte)((1+b[i][j])%2); // 0<-->1
  }

  static void comutaLinia(int i)
  {
    int j;
    for(j=1;j<=n;j++) b[i][j]=(byte)((1+b[i][j])%2); // 0<-->1
  }

  static void afisb()
  {
    int i,j;
    for(i=1;i<=m;i++)
    {
      for(j=1;j<=n;j++) System.out.print(b[i][j]+" ");
      System.out.println();
    }
  }
}
```

A doua fază: pun 1 pe prima linie şi verific. Constatăm că se poate folosi aceeaşi metotă dar cu un parametru pentru a transmite valoarea 0 sau 1.

```
import java.io.*;
class Becuri2
{
  static int m,n;
  static byte[][] a;
  static byte[][] b;
  static byte[] colc;
  static byte[] linc;
  static byte[] colcsol;
  static byte[] lincsol;
  static int ncolcsol=234, nlincsol=234;

  public static void main(String[]args) throws IOException
  {
    int i,j,k;
    BufferedReader br=new BufferedReader(new FileReader("becuri.in"));
```

```
    StreamTokenizer st=new StreamTokenizer(br);

    st.nextToken();m=(int)st.nval;
    st.nextToken();n=(int)st.nval;

    a=new byte[m+1][n+1];
    b=new byte[m+1][n+1];
    colc=new byte[n+1];
    colcsol=new byte[n+1];
    linc=new byte[m+1];
    lincsol=new byte[m+1];

    for(i=1;i<=m;i++)
      for(j=1;j<=n;j++) { st.nextToken(); a[i][j]=(byte)st.nval; }

    facLinia1((byte)0);
    facLinia1((byte)1);

    System.out.println(ncolcsol+" "+nlincsol);

    PrintWriter out=new PrintWriter(
                    new BufferedWriter(new FileWriter("becuri.out")));
    out.close();
  }

  static void facLinia1(byte val)
  {
    int i,j;
    int nlc=0, ncc=0; // nr linii/coloane comutate
    boolean ok=true;

    for(j=1;j<=n;j++) colc[j]=0; // coloane necomutate inca
    for(i=1;i<=m;i++) linc[i]=0; // linii necomutate inca
    for(i=1;i<=m;i++) for(j=1;j<=n;j++) b[i][j]=a[i][j]; // copie

    for(j=1;j<=n;j++)
      if(b[1][j]!=val)
      {
        comutaColoana(j);
        colc[j]=1; // coloana j este comutata;
        ncc++;
      }
    afisb();
    for(i=1;i<=m;i++) if(!okLinia(i)) { ok=false; break; }
```

```
    else if(b[i][1]==1) { linc[i]=1; nlc++; }
    if(ok&&(nlc+ncc<nlincsol+ncolcsol))
    {
      nlincsol=nlc;
      ncolcsol=ncc;
      for(i=1;i<=m;i++) lincsol[i]=linc[i];
      for(j=1;j<=n;j++) colcsol[j]=colc[j];
    }
  }

  static boolean okLinia(int i)
  {
    int j;
    for(j=2;j<=n;j++) if(b[i][j]!=b[i][1]) return false;
    return true;
  }

  static void comutaColoana(int j)
  {
    int i;
    for(i=1;i<=m;i++) b[i][j]=(byte)((1+b[i][j])%2); // 0<-->1
  }

  static void comutaLinia(int i)
  {
    int j;
    for(j=1;j<=n;j++) b[i][j]=(byte)((1+b[i][j])%2); // 0<-->1
  }

  static void afisb()
  {
    int i,j;
    for(i=1;i<=m;i++)
    {
      for(j=1;j<=n;j++) System.out.print(b[i][j]+" ");
      System.out.println();
    }
    System.out.println();
  }
}
```

Analog pentru prima coloană. în plus, se finalizează scrierea rezultatelor în fişier.

9.4.3 Codul sursă *

```
import java.io.*;
class Becuri3
{
  static int m,n;
  static byte[][] a;
  static byte[][] b;
  static byte[] colc;
  static byte[] linc;
  static byte[] colcsol;
  static byte[] lincsol;
  static int ncolcsol=234, nlincsol=234;

  public static void main(String[]args) throws IOException
  {
    int i,j,k;
    BufferedReader br=new BufferedReader(new FileReader("becuri.in"));
    StreamTokenizer st=new StreamTokenizer(br);

    st.nextToken();m=(int)st.nval;
    st.nextToken();n=(int)st.nval;

    a=new byte[m+1][n+1];
    b=new byte[m+1][n+1];
    colc=new byte[n+1];
    colcsol=new byte[n+1];
    linc=new byte[m+1];
    lincsol=new byte[m+1];

    for(i=1;i<=m;i++)
      for(j=1;j<=n;j++) { st.nextToken(); a[i][j]=(byte)st.nval; }

    facLinia1((byte)0);
    facLinia1((byte)1);
    facColoana1((byte)0);
    facColoana1((byte)1);

    PrintWriter out=new PrintWriter(
                    new BufferedWriter(new FileWriter("becuri.out")));

    if(nlincsol+ncolcsol>n+m) out.println(-1);
    else
```

```
    {
      if(ncolcsol==0) out.print(0);
        else for(j=1;j<=n;j++) if(colcsol[j]==1) out.print(j+" ");
      out.println();
      if(nlincsol==0) out.print(0);
        else for(i=1;i<=m;i++) if(lincsol[i]==1) out.print(i+" ");
      out.println();
    }
    out.close();
  }

  static void facLinia1(byte val)
  {
    int i,j;
    int nlc=0, ncc=0; // nr linii/coloane comutate
    boolean ok=true;

    for(j=1;j<=n;j++) colc[j]=0; // coloane necomutate inca
    for(i=1;i<=m;i++) linc[i]=0; // linii necomutate inca
    for(i=1;i<=m;i++) for(j=1;j<=n;j++) b[i][j]=a[i][j]; // copie

    for(j=1;j<=n;j++)
    if(b[1][j]!=val)
    {
      comutaColoana(j);
      colc[j]=1; // coloana j este comutata;
      ncc++;
    }
    for(i=1;i<=m;i++)
      if(!okLinia(i)) {ok=false; break;}
      else if(b[i][1]==1) { linc[i]=1; nlc++; }
    if(ok&&(nlc+ncc<nlincsol+ncolcsol))
    {
      nlincsol=nlc;
      ncolcsol=ncc;
      for(i=1;i<=m;i++) lincsol[i]=linc[i];
      for(j=1;j<=n;j++) colcsol[j]=colc[j];
    }
  }

  static void facColoana1(byte val)
  {
    int i,j;
    int nlc=0, ncc=0; // nr linii/coloane comutate
```

```
    boolean ok=true;

    for(j=1;j<=n;j++) colc[j]=0; // coloane necomutate inca
    for(i=1;i<=m;i++) linc[i]=0; // linii necomutate inca
    for(i=1;i<=m;i++) for(j=1;j<=n;j++) b[i][j]=a[i][j]; // copie

    for(i=1;i<=m;i++)
      if(b[i][1]!=val)
      {
        comutaLinia(i);
        linc[i]=1; // linia i este comutata;
        nlc++;
      }
    for(j=1;j<=n;j++)
      if(!okColoana(j)) { ok=false; break;}
      else if(b[1][j]==1) { colc[j]=1; ncc++; }
    if(ok&&(nlc+ncc<nlincsol+ncolcsol))
    {
      nlincsol=nlc;
      ncolcsol=ncc;
      for(i=1;i<=m;i++) lincsol[i]=linc[i];
      for(j=1;j<=n;j++) colcsol[j]=colc[j];
    }
  }

  static boolean okLinia(int i)
  {
    int j;
    for(j=2;j<=n;j++) if(b[i][j]!=b[i][1]) return false;
    return true;
  }

  static boolean okColoana(int j)
  {
    int i;
    for(i=2;i<=m;i++) if(b[i][j]!=b[1][j]) return false;
    return true;
  }

  static void comutaColoana(int j)
  {
    int i;
    for(i=1;i<=m;i++) b[i][j]=(byte)((1+b[i][j])%2); // 0<-->1
  }
```

```
  static void comutaLinia(int i)
  {
    int j;
    for(j=1;j<=n;j++) b[i][j]=(byte)((1+b[i][j])%2); // 0<-->1
  }
}
```

9.5 Discuri

Florin Gheţu, Bucureşti

Se dau N numere reale considerate ca fiind razele a N discuri.

Considerăm că aşezăm un disc în sistemul xOy dacă îl plasăm la o coordonată x pozitivă suficient de mare, tangent cu axa Ox şi deasupra ei, apoi îl împingem spre Oy până când devine tangent cu Oy sau cu primul disc întâlnit, aşezat anterior.

După aşezarea tuturor discurilor în ordinea dată unele dintre ele pot fi considerate dispensabile, pentru că prin eliminarea lor nu se modifică lăţimea totală a figurii rezultate, adică nici un disc nu se mai poate deplasa spre stânga.

Cerinţă:

Identificaţi toate discurile dispensabile dintr-o configuraţie dată.

Date de intrare:

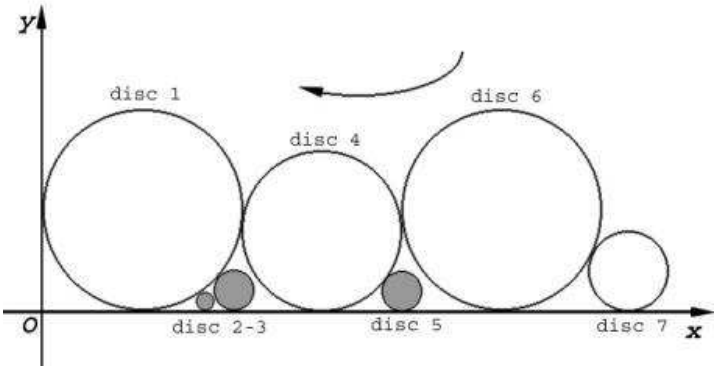
Din fişierul de intrare DISCURI.IN veţi citi de pe prima linie numărul N de discuri, iar de pe următoarele N linii, N numere reale reprezentând razele discurilor în ordinea de aşezare, câte unul pe linie.

Date de iesire:

În fişierul DISCURI.OUT veţi scrie pe prima linie numărul K de discuri dispensabile, iar pe următoarele K linii, K întregi reprezentând numerele de ordine ale discurilor considerate dispensabile, câte unul pe linie.

Restricţii:

- $N \leq 1000$.



Exemplu (figura de mai sus; discurile haşurate sunt dispensabile)

DISCURI.IN	DISCURI.OUT
7	3
4	2
0.1	3
0.5	5
3	
0.5	
4	
1	

Timpul maxim de execuţie: 1 secundă/test

9.5.1 Indicaţii de rezolvare - descriere soluţie *

Soluţie prezentată în GInfo 12/6

Pentru identificarea discurilor dispensabile va trebui să determinăm atingerile dintre discurile care influenţează lăţimea figurii. De exemplu, dacă avem şase discuri, cu raze de 1000, 1, 2, 3, 1000, respectiv 500, atunci atingerile care influenţează lăţimea figurii sunt între primul şi al cincilea disc, respectiv între al cincilea şi al şaselea.

Pentru fiecare disc i vom determina, pe rând, coordonatele orizonatale pe care le-ar avea discul dacă ar atinge unul dintre discurile anterioare.

În final, vom alege discul (sau axa Oy) pentru care coordonata orizontală a centrului noului disc este maximă şi putem afirma că discul i ajuge în această poziţie.

Dacă discul i va atinge un disc anterior j, atunci discurile cu numerele de ordine cuprinse între $j+1$ şi $i-1$ sunt dispensabile.

După ce vom lua în considerare toate cele N discuri, vom putea determina numerele de ordine ale tuturor discurilor dispensabile.

În final vom verifica dacă există discuri introduse la sfârşit care sunt dispensabile. Pentru aceasta vom determina lăţimea figurii şi ultimul disc care o influenţează. Toate discurile introduse după acest disc sunt dispensabile.

Pentru determinarea coordonatei orizontale x_i a centrului unui disc i care atinge un disc j avem nevoie de coordonata orizontală x_j a centrului discului j, precum şi de razele r_i şi r_j ale celor două discuri. Dacă aceste trei valori sunt cunoscute, se poate folosi următoarea formulă pentru a determina coordonata orizontală a centrului discului j:

$$x_j = x_i + \sqrt{(r_i + r_j)^2 - (r_i - r_j)^2}.$$

Analiza complexităţii

Pentru fiecare disc i care este introdus, se determină posibila coordonată x_i datorată atingerii cu toate cele $i-1$ discuri inserate anterior. Pentru cele N discuri se vor determina, în total, $0+1+2+...+(N-1) = N(N-1)/2$ coordonate, deci ordinul de complexitate al acestei operaţii este $O(N^2$.

La fiecare pas, pot fi marcate ca dispensabile cel mult toate discurile inserate anterior, aşadar ordinul de complexitate al acestei operaţii este tot $O(N^2)$.

Determinarea lăţimii figurii şi a cercurilor dispensabile de la sfârşitul secvenţei necesită a singură parcurgere a şirului care păstrează coordonatele centrelor discurilor, ceea ce implică ordinul de complexitate $O(N)$.

Afişarea cercurilor dispensabile precum şi citirea razelor cercurilor sunt operaţii care se efectuează în timp liniar ($O(N)$), necesitând o simplă parcurgere a unor şiruri.

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(N^2) + O(N^2) + O(N) + O(N) + O(N) = O(N^2)$.

9.5.2 Rezolvare detaliată

9.5.3 Codul sursă *

```
import java.io.*;
class Discuri
{
  static int ultimulDisc;
  static double latime;
  static boolean[] dispensabil;
  static double[] r;
  static double[] x;
  static int n,k;

  public static void main(String args[]) throws IOException
```

```
{
  citeste();
  calcul();
  scrie();
}

static void citeste() throws IOException
{
  BufferedReader br=new BufferedReader(new FileReader("discuri.in"));
  StreamTokenizer st=new StreamTokenizer(br);
  st.nextToken(); n=(int)st.nval;
  r=new double[n+1];
  x=new double[n+1];
  dispensabil=new boolean[n+1];               // implicit este false
  for(int i=1; i<=n; i++){st.nextToken();r[i]=st.nval;}
  br.close();
}

static void calcul()
{
  int i;
  for(i=1; i<=n; i++) plaseazaCerc(i);
  for(i=ultimulDisc+1; i<=n; i++) dispensabil[i]=true;
}

static void plaseazaCerc(int i)
{
  int j, discContact=0;
  double contactOx;
  x[i]=r[i];               // initializare contact cu axa Oy
  for(j=1; j<i; j++)
  {
    contactOx=coord(x[j],r[j],r[i]);
    if(contactOx>x[i]) {x[i]=contactOx; discContact=j;}
  }
  for(j=discContact+1; j<i; j++) dispensabil[j]=true;
  if(latime<x[i]+r[i]) {latime=x[i]+r[i]; ultimulDisc=i;}
}

static double coord(double xj,double rj, double ri)
{
  return(xj+Math.sqrt((ri+rj)*(ri+rj)-(ri-rj)*(ri-rj)));
}
```

```
  static void scrie() throws IOException
  {
    int i;
    k=0;
    for(i=1; i<=n; i++) if(dispensabil[i]) k++;
    PrintWriter out=new PrintWriter(
                    new BufferedWriter(new FileWriter("discuri.out")));
    out.println(k);
    for(i=1; i<=n; i++) if(dispensabil[i]) out.println(i);
    out.close();
  }
}
```

9.6 Cod

lect. univ. Ovidiu Domşa, Alba Iulia

Transmiterea şi memorarea informaţiilor necesită diverse sisteme de codificare în vederea utilizării optime a spaţiilor disponibile. Un sistem foarte des întâlnit este acela prin care unei secvenţe de caractere i se asociază un număr.

Se consideră cuvintele formate numai cu literele mici ale alfabetului englez a, b, c, ..., z (26 de caractere). Din toate aceste cuvinte le considerăm doar pe cele ale căror caractere sunt în ordine strict lexicografică (caracterul de pe orice poziţie este strict mai mic decât orice caracter următor).

Sistemul de codificare se obţine astfel:

• Se ordonează cuvintele în ordinea crescătoare a lungimilor lor.

• Cuvintele de aceeaşi lungime se ordonează lexicografic (în ordinea alfabetică a cuvintelor dintr-un dicţionar).

• Codificăm aceste cuvinte prin numerotarea lor începând cu a, după cum urmează:

a – 1
b – 2
...
z – 26
ab – 27
...
az – 51
bc – 52
...
$vwxzy$ – 83681
...

Cerinţă:

Dacă se dă un cuvânt să se precizeze dacă poate fi codificat conform sistemului de codificare. În caz afirmativ să se precizeze codul său.

Date de intrare
Fişierul de intrare COD.IN conţine pe o linie un cuvânt.

Date de ieşire
Fişierul COD.OUT va conţine codul cuvântului ce trebuie codificat, sau 0 în cazul în care cuvântul nu poate fi codificat.

Restricţii şi precizări
• Numărul maxim de litere ale unui cuvânt este 10
• Numărul de caractere din alfabetului englez este 26

Exemple

COD.IN	COD.OUT	COD.IN	COD.OUT	COD.IN	COD.OUT
bf	55	aab	0	vwxyz	83681

Timp maxim de execuţie: 2 secunde/test

9.6.1 Indicaţii de rezolvare - descriere soluţie *

Soluţie prezentată în GInfo 12/6

Din condiţiile precizate în enunţ rezultă că, pe baza unei mulţimi de litere distincte, se poate construi un singur cuvânt care respectă condiţiile date, şi anume cel care conţine literele ordonate lexicografic. Aşadar, oricărei mulţimi de cel mult zece litere distincte îi corespunde un cuvânt care respectă condiţiile din enunţ.

Se poate afirma că un cuvânt este o submulţime a mulţimii literelor; de aici rezultă că numărul cuvintelor formate din k litere este C_{26}^k. Mai mult, numărul cuvintelor formate din k dintre ultimele n litere ale alfabetului este C_n^k.

Numărul de ordine al cuvântului dat este mai mare decât cel al cuvintelor formate din mai multe cifre. Din aceste motive, pentru un cunvânt format din k litere, vom avea un număr mai mare decât $\sum_{i=1}^{k} C_{26}^i$.

În continuare, pentru prima literă a cuvântului, va trebui să găsim numărul cuvintelor care încep cu o literă mai mică (din punct de vedere lexicografic).

În cazul în care cuvântul are k litere, vor exista C_{25}^{k-1} cuvinte valide care k încep cu 'a', C_{25}^{k-1} cuvinte valide care încep cu 'b' etc.

În general, vor exista C_{26-i}^{k-1} cuvinte valide care încep cu a i-a literă a alfabetului.

Dacă prima literă a cuvântului este cea de-a n-a literă a alfabetului, vom avea $\sum_{i=1}^{n} C_{26-i}^{k-1}$ cuvinte valide de care încep cu o literă mai mică.

În acest moment ştim numărul de ordine minim al unui cuvânt care începe cu prima literă a cuvântului dat.

Pentru a doua literă vom proceda într-o manieră asemănătoare. Singura diferenţă este dată de faptul că a doua literă trebuie să fie strict mai mare decât prima. Aşadar, dacă prima literă este cea de-a n-a a alfabetului, iar a doua este

cea de-a m-a, atunci vom avea $\sum_{i=n+1}^{m-1} C_{26-i}^{k-2}$ cuvinte care au pe prima poziţia aceeaşi literă, iar pe cea de-a doua poziţie o literă mai mică.

Procedeul va continua pentru fiecare literă în parte.

În cazul în care litera curentă este cea de-a p-a a cuvântului, este a m-a a alfabetului, iar litera anterioară este a n-a a alfabetului, numărul de cuvite care au pe primele $p-1$ poziţii aceleaşi litere ca şi cuvântul dat, iar pe cea de-a p-a o literă mai mică este dat de formula $\sum_{i=n+1}^{m-1} C_{26-i}^{k-p}$.

Adunând toate valorile obţinute pe parcurs vom obţine numărul cuvintelor care se află înaintea cuvântului dat. Adunând 1 la această valoare, vom obţine numărul de ordine al cuvântului.

Analiza complexităţii

Pentru a analiza complexitatea acestui algoritm va trebui să precizăm faptul că numărul literelor din alfabet este constant, deci nu poate interveni în exprimarea ordinului de complexitate. Acesta va fi stabilit doar în funcţie de lungimea k a cuvântului.

Iniţial se calculează suma $\sum_{i=1}^{k} C_{26}^{i}$, operaţie realizabilă în timp liniar, având în vedere observaţia anterioară. Aşadar, primul pas al algoritmului are ordinul de complexitate $O(k)$.

Pentru fiecare literă a cuvântului se calculează suma $\sum_{i=n+1}^{m-1} C_{26-i}^{k-2}$, unde variabilele au semnificaţia prezentată anterior. Numărul de litere este implicat în determinarea valorii combinării. Aşadar, calculul combinării se realizează în timp liniar. Numărul de combinări calculate nu depinde de lungimea cuvântului, deci ordinul de complexitate al calculării acestei sume este $O(k)$.

În total vor fi calculate k astfel de sume, deci ordinul de complexitate al celui de-al doilea pas al algoritmului este $O(k) \cdot O(k) = O(k^2)$.

Citirea datelor de intrare şi scrierea celor de ieşire se realizează cu ajutorul unor operaţii elementare, deci putem considera că au ordinul de complexitate $O(1)$.

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(k^2) + O(k) + O(k) = O(k^2)$ având în vedere că numărul literelor din alfabetul englez este constant.

9.6.2 Rezolvare detaliată

9.6.3 Codul sursă *

```
import java.io.*;
class Cod
{
  static String cuvant;
  static long codcuvant;
```

```
public static void main(String args[]) throws IOException
{
  cuvant=citeste();
  codcuvant=calcul();
  scrie();
}

static String citeste() throws IOException
{
  BufferedReader br=new BufferedReader(new FileReader("cod.in"));
  String s=br.readLine();
  br.close();
  return(s);
}

static long calcul()
{
  int k=cuvant.length();
  int i,j,m,n=0;
  long nr=1;
  for(i=1; i<k; i++) nr+=comb(26,i);
  for(i=0; i<k; i++)
  {
    if(i>0)
    {
      n=cuvant.charAt(i-1)-'a'+1;
      if(cuvant.charAt(i)<=cuvant.charAt(i-1)) return 0;
    }
    m=cuvant.charAt(i)-'a'+1;
    for(j=n+1; j<m; j++) nr+=comb(26-j,k-i-1);
  }
  return nr;
}

static void scrie() throws IOException
{
  PrintWriter out=new PrintWriter(
                  new BufferedWriter(new FileWriter("cod.out")));
  out.print(codcuvant);
  out.close();
}

static long comb(int n,int k)
```

```
  {
    int i,j,d;
    if(k>n/2) k=n-k;
    int[] x=new int[k+1];
    int[] y=new int[k+1];
    for(i=1;i<=k;i++) x[i]=n-k+i;
    for(j=1;j<=k;j++) y[j]=j;
    for(j=2;j<=k;j++)
    for(i=1;i<=k;i++)
    {
      d=cmmdc(x[i],y[j]);
      x[i]=x[i]/d;
      y[j]=y[j]/d;
      if(y[j]==1) break;
    }
    long p=1;
    for(i=1;i<=k;i++) p=p*x[i];
    return p;
  }

  static int cmmdc(int a, int b)
  {
    int d,i,c,r;
    if (a>b) {d=a; i=b;} else {d=b; i=a;}
    while (i != 0){ c=d/i; r=d%i; d=i; i=r; }
    return d;
  }
}
```

Capitolul 10

ONI 2003 clasa a IX-a

10.1 Seti

Cercetătorii ce lucrează la programul SETI au recepţionat două transmisii de date foarte ciudate, date care ar putea veni din partea unor civilizaţii extraterestre. Primul set de date este format din 10 caractere distincte, date în ordinea lor lexicografică, ce formează alfabetul extraterestru. A doua transmisie conţine cuvinte din exact 4 caractere.

Cerinţă

Cercetătorii trebuie să ordoneze lexicografic cuvintele primite în a doua transmisie (conform alfabetului extraterestru).

Date de intrare

Fişierul de intrare **seti.in** conţine pe prima linie cele 10 caractere ale alfabetului, iar pe fiecare din următoarele linii câte un cuvânt.

Date de ieşire

Fişierul de ieşire **seti.out** va conţine cuvintele ordonate, câte unul pe linie.

Restricţii şi precizări

• În fişier nu sunt mai mult de 200.000 de cuvinte, iar caracterele sunt literele mici ale alfabetului englez.

• Datele de intrare se presupun ca fiind corecte.

Exemplu

seti.in	seti.out
abcdefghij	aaaa
aaaa	aabc
fgaa	fgaa
aabc	iihf
iihf	

Timp de execuţie: 1 sec/test

10.1.1 Indicaţii de rezolvare - descriere soluţie *

Soluţie prezentată de Mihai Stroe, GInfo nr. 13/6

Rezolvarea problemei constă în realizarea unei metode de trecere de la cuvintele de 4 litere la numerele de cel mult 4 cifre. Având o astfel de metodă, toate cuvintele se vor transforma în numere cuprinse între 0 şi 9999.

Numerele nu vor fi memorate, în schimb se va folosi un tablou cu 10.000 de elemente, fiecare element reprezentând numărul de apariţii al unui cuvânt din fişier.

În final, pentru fiecare număr de la 0 la 9999, se va afişa cuvântul reprezentat de atâtea ori de câte indică numărul său de apariţii.

Pentru a trece de la cuvânt la număr şi invers se poate folosi ca etapă intermediară un tablou de patru cifre, transformările realizându-se foarte uşor.

Analiza complexităţii

Fie N numărul de cuvinte din fişier care poate avea valoarea maximă 200.000.

Operaţia de citirea a datelor are ordinul de complexitate $O(N)$.

Metodele care transformă cuvintele în numere şi invers au ordinul de complexitate $O(1)$.

Actualizarea tabloului care contorizează numărul de apariţii se face concomitent cu citirea, deci această operaţie are ordinul de complexitate $O(N)$.

Operaţia de afişarea a datelor are ordinul de complexitate $O(N)$.

În final, ordinul de complexitate al algoritmului de rezolvare este $O(N) + O(N) + O(N) = O(N)$.

10.1.2 Rezolvare detaliată

10.1.3 Codul sursă *

Versiune cu mesaje de verificare!

```
import java.io.*;
class Seti1
{
  static int[] codd=new int[10];
  static int[] codi=new int[26];
  static char[] caractercd=new char[10];
  static int[] f=new int[10000];

  public static void main(String[]args) throws IOException
  {
    String alfabet,cuvant;
    int i,nr;
    long t1,t2;
    t1=System.currentTimeMillis();
    PrintWriter out = new PrintWriter(
                      new BufferedWriter(new FileWriter("seti.out")));
    StreamTokenizer st=new StreamTokenizer(
                       new BufferedReader(new FileReader("seti.in")));

    st.nextToken(); alfabet=st.sval.toString();
    System.out.println(alfabet);

    for(i=0;i<10;i++)
    {
      codd[i]=alfabet.charAt(i)-'a';
      caractercd[i]=alfabet.charAt(i);
    }
    for(i=0;i<10;i++)
      System.out.println(i+" : "+codd[i]+" "+caractercd[i]);
    System.out.println();

    for(i='a';i<='z';i++) codi[i-'a']=-1;
    for(i=0;i<10;i++) codi[codd[i]]=i;

    for(i=0;i<='z'-'a';i++)
      System.out.println(i+" : "+codi[i]+" "+(char)(i+'a'));
    System.out.println();

    while(st.nextToken()!=st.TT_EOF) // preluarea cuvantului
    {
```

```
      cuvant=st.sval.toString(); // conversia in String
      nr=0;
      for(i=0;i<4;i++) nr=nr*10+codi[cuvant.charAt(i)-'a'];
      System.out.println(cuvant+" "+nr);
      f[nr]++;
    }

    for(i=0;i<10000;i++)
    if(f[i]>0)
    {
      cuvant=cuvantul(i);
      for(nr=1;nr<=f[i];nr++) out.println(cuvant);
    }
    out.close();
    t2=System.currentTimeMillis();
    System.out.println("Timp = "+(t2-t1));
  }//main

  static String cuvantul(int nr)
  {
    int r,i;
    char[] c=new char[4];
    for(i=0;i<4;i++) c[i]=caractercd[0];

    i=3;
    while(nr!=0)
    {
      r=nr%10;
      c[i]=caractercd[r];
      nr=nr/10;
      i--;
    }
    String s=new String(c);
    return s;
  }
}//class
```

Versiune finala!

```
import java.io.*;
class Seti2
{
  static int[] codd=new int[10];
  static int[] codi=new int[26];
  static char[] caractercd=new char[10];
```

```
  static int[] f=new int[10000];

  public static void main(String[]args) throws IOException
  {
    String alfabet,cuvant;
    int i,nr;
   long t1,t2;
    t1=System.currentTimeMillis();
    PrintWriter out = new PrintWriter(
                      new BufferedWriter(new FileWriter("seti.out")));
    StreamTokenizer st=new StreamTokenizer(
                       new BufferedReader(new FileReader("seti.in")));

    st.nextToken(); alfabet=st.sval.toString();

    for(i=0;i<10;i++)
    {
      codd[i]=alfabet.charAt(i)-'a';
      caractercd[i]=alfabet.charAt(i);
    }

    for(i='a';i<='z';i++) codi[i-'a']=-1;
    for(i=0;i<10;i++) codi[codd[i]]=i;

    while(st.nextToken()!=st.TT_EOF) // preluarea cuvantului
    {
      cuvant=st.sval.toString(); // conversia in String
      nr=0;
      for(i=0;i<4;i++) nr=nr*10+codi[cuvant.charAt(i)-'a'];
      f[nr]++;
    }

    for(i=0;i<10000;i++)
    if(f[i]>0)
    {
      cuvant=cuvantul(i);
      for(nr=1;nr<=f[i];nr++) out.println(cuvant);
    }
    out.close();
    t2=System.currentTimeMillis();
    System.out.println("Timp = "+(t2-t1));
  }//main

  static String cuvantul(int nr)
```

```
  {
    int i;
    char[] c=new char[4];
    for(i=0;i<4;i++) c[i]=caractercd[0];

    i=3;
    while(nr!=0)
    {
      c[i]=caractercd[nr%10];
      nr=nr/10;
      i--;
    }
    String s=new String(c);
    return s;
  }
}//class
```

10.2 Scaune

Se consideră **ns** scaune numerotate de la 1 la **ns**, aranjate în cerc.

Exemplu pentru $ns = 20$ aşezarea scaunelor este dată în figură.

20	19	18	17	16	15	14	13	12	11
	←	←	←	←	←	←	←	←	←
↓									↑
	→	→	→	→	→	→	→	→	→
1	2	3	4	5	6	7	8	9	10

Pe fiecare din aceste scaune este aşezat un copil. Primul copil stă pe scaunul 1, iar ultimul pe scaunul **ns**. Pe lângă cele **ns** scaune deja ocupate, alţi **nc** copii ($1 \leq$ **nc** $\leq$ **ns**) aşteaptă să se elibereze un loc.

La un moment dat un singur copil se ridică de pe scaun şi pleacă. Atunci, cât timp în dreptul scaunului liber nu există un copil, toţi copiii aflaţi în aşteptare se mişcă în sens invers mişcării acelor ceasornicului, câte o poziţie, până când unul din ei ocupă locul liber.

Condiţii:

– la început toate scaunele sunt ocupate;

– fiecare copil aflat în aşteptare se află iniţial în dreptul unui scaun ocupat;

– când un copil avansează cu n poziţii spre un loc pe scaun, toţi cei care aşteaptă avansează tot cu n poziţii. Deoarece mişcarea este circulară, avansarea cu 4 poziţii de la poziţia 18, semnifică o deplasare în dreptul poziţiei 2;

Cerinţă

Dacă se dă o secvenţă a numerelor de ordine a copiilor care pleacă la fiecare moment, să se scrie un program care să afişeze numărul scaunului pe care s-a aşezat fiecare copil care aşteaptă, dacă acest lucru este posibil.

Date de intrare

• Pe prima linie a fişierului text de intrare **scaune.in** se află două numere, separate prin spaţiu, reprezentând numărul de scaune, **ns** şi respectiv numărul copiilor care stau în aşteptare **nc**.

• Pe următoarele **nc** linii vor fi date poziţiile copiilor aflaţi în aşteptare.

• În continuare până la sfârşitul fişierului sunt linii ce descriu numerele de ordine ale copiilor care se ridică unul câte unul de pe scaune şi părăsesc jocul.

Date de ieşire

Fişierul de ieşire **scaune.out** conţine **nc** linii, fiecare linie conţinând poziţia iniţială de aşteptare a copilului şi poziţa ocupată, separate printr-un spaţiu.

Liniile de ieşire trebuie să fie în aceeaşi ordine ca cele din fişierul de intrare.

În cazul în care un copil nu are nici o posibilitate să se aşeze, în dreptul său se va scrie 0 în fişierul de ieşire.

Restricţii

• $1 \leq$ **ns** ≤ 200

• **nc** $\leq$ **ns**

Exemplu

scaune.in	**scaune.out**
20 5 6 19 17 13 1 1 3 20 16	6 16 19 3 17 0 13 20 1 1

Timp maxim de execuţie: 1 secundă/test

10.2.1 Indicaţii de rezolvare - descriere soluţie *

Soluţie prezentată de Mihai Stroe, GInfo nr. 13/6

Rezolvarea acestei probleme constă în simularea deplasării copiilor, conform regulilor din enunţ.

Deoarece limita pentru numărul de scaune este destul de mică, avansarea copiilor se poate realiza pas cu pas; astfel, în momentul în care un scaun se

eliberează, fiecare copil avansează câte o poziţie până când unul dintre ei ajunge în dreptul scaunului şi îl ocupă.

O metodă mai rapidă constă în calcularea, pentru fiecare copil, a numărului de mutări până la ajungerea la scaunul liber.

Fie un copil situat în poziţia C, iar scaunul liber în poziţia S; atunci copilul va avansa $S - C$ poziţii, dacă $C = S$, respectiv $NS + S - C$ poziţii, dacă $C > S$.

Astfel, la fiecare ridicare a unui copil se poate afla cel mai apropiat copil, se calculează câte poziţii se va deplasa acesta (să spunem P), apoi toţi copiii vor avansa P poziţii. Un copil care avansează P poziţii din poziţia C va ajunge în poziţia $C + P$ (dacă $C + P = NS$), respectiv $C + P - NS$ (dacă $C + P > NS$).

Analiza complexităţii

Deoarece sunt NC copii în aşteptare, vor fi analizate cel mult NC ridicări de pe scaun, indiferent dacă mai urmează ceva în fişierul de intrare.

Dacă sunt mai puţin de NC copii care se ridică, atunci o parte dintre cei care aşteaptă vor rămâne în picioare.

La fiecare moment în care se ridică un copil, avansarea pas cu pas a copiilor aflaţi în aşteptare până la aşzarea pe scaun a unuia dintre ei nu poate necesita mai mult de NS paşi; fiecare pas necesită avansarea tuturor copiilor, deci a cel mult NC copii.

Complexitatea ocupării unui scaun prin avansare pas cu pas este deci $O(NS \cdot NC)$.

În cazul metodei mai rapide, calcularea distanţei pentru toţi copiii până la scaunul liber are complexitatea $O(NC)$. Aceeaşi complexitate au şi operaţiile de alegere a minimului P dintre aceste distanţe, respectiv de avansare a tuturor copiilor cu P poziţii.

Ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(NC \cdot NS \cdot NC)$ pentru prima variantă, respectiv $O(NC \cdot NC)$ pentru cea de-a doua.

10.2.2 Rezolvare detaliată

10.2.3 Codul sursă *

Diferenţa de timp între cele două variante este mică!

```
import java.io.*;
class Scaune1
{
  static int ns, nc;
  static int[] pozInitiala;
  static int[] pozFinala;
```

```
static boolean[] asezat;

public static void main(String[]args) throws IOException
{
  int i,j,k,nca=0; // nca=nr copii asezati
  long t1,t2;
  t1=System.currentTimeMillis();
  PrintWriter out = new PrintWriter(
                    new BufferedWriter(new FileWriter("scaune.out")));
  StreamTokenizer st=new StreamTokenizer(
                     new BufferedReader(new FileReader("scaune.in")));
  st.nextToken(); ns=(int)st.nval;
  st.nextToken(); nc=(int)st.nval;
  asezat=new boolean[nc+1];
  pozInitiala=new int[nc+1];
  pozFinala=new int[nc+1];

  for(k=1;k<=nc;k++)
  {
    st.nextToken();
    pozInitiala[k]=pozFinala[k]=(int)st.nval;
  }
  while(st.nextToken()!=st.TT_EOF)
  {
    k=(int)st.nval; // scaunul k este liber
    i=esteLaPozitia(k);
    while((nca<nc)&&(i==0)) { misca(); i=esteLaPozitia(k); }
    pozFinala[i]=k;
    asezat[i]=true;
    nca++;
  }

  for(j=1;j<=nc;j++)
    if(asezat[j]) out.println(pozInitiala[j]+" "+pozFinala[j]);
    else out.println(pozInitiala[j]+" "+0);
  out.close();
  t2=System.currentTimeMillis();
  System.out.println("Timp = "+(t2-t1));
}//main

static void misca()
{
  int i;
  for(i=1;i<=nc;i++)
```

```
    if(!asezat[i])
    {
      pozFinala[i]++;
      if(pozFinala[i]==ns+1) pozFinala[i]=1;
    }
  }

  static int esteLaPozitia(int k)
  {
    int i,copil=0;
    for(i=1;i<=nc;i++)
      if(!asezat[i]&&(pozFinala[i]==k)) { copil=i; break; }
    return copil;
  }
}//class
```

Pentru a doua varianta se calculează distanţele până la scaunul liber pentru fiecare copil rămas neaşezat.

```
import java.io.*;
class Scaune2
{
  static int ns, nc;
  static int[] pozInitiala;
  static int[] pozFinala;
  static boolean[] asezat;
  static int[] d;

  public static void main(String[]args) throws IOException
  {
    int i,j,k;
    long t1,t2;
    t1=System.currentTimeMillis();
    PrintWriter out = new PrintWriter(
                      new BufferedWriter(new FileWriter("scaune.out")));
    StreamTokenizer st=new StreamTokenizer(
                       new BufferedReader(new FileReader("scaune.in")));
    st.nextToken(); ns=(int)st.nval;
    st.nextToken(); nc=(int)st.nval;
    asezat=new boolean[nc+1];
    d=new int[nc+1];
    pozInitiala=new int[nc+1];
    pozFinala=new int[nc+1];
```

```
    for(k=1;k<=nc;k++)
    {
      st.nextToken();
      pozInitiala[k]=pozFinala[k]=(int)st.nval;
    }
    while(st.nextToken()!=st.TT_EOF)
    {
      k=(int)st.nval; // scaunul k este liber
      i=esteLaPozitia(k);
      pozFinala[i]=k;
      asezat[i]=true;
    }

    for(j=1;j<=nc;j++)
      if(asezat[j]) out.println(pozInitiala[j]+" "+pozFinala[j]);
      else out.println(pozInitiala[j]+" "+0);
    out.close();
    t2=System.currentTimeMillis();
    System.out.println("Timp = "+(t2-t1));
  }//main

  static int esteLaPozitia(int k)
  {
    int i,min,imin;
    for(i=1;i<=nc;i++)
      if(!asezat[i])
        if(pozFinala[i]<=k) d[i]=k-pozFinala[i];
        else d[i]=k+ns-pozFinala[i];

    imin=0; min=ns+1;
    for(i=1;i<=nc;i++)
      if(!asezat[i])
        if(d[i]<min){ min=d[i]; imin=i; }

    for(i=1;i<=nc;i++)
      if(!asezat[i])
      {
        pozFinala[i]=pozFinala[i]+min;
        if(pozFinala[i]>ns) pozFinala[i]=pozFinala[i]-ns;
      }
    return imin;
  }
}//class
```

10.3 Circular

Unele numere naturale sunt formate doar din cifre distincte nenule.

Dintre acestea, unele, numite *numere circulare*, au următoarea proprietate: pornind de la prima cifră şi numărând spre dreapta, după cifră, atâtea cifre cât indică aceasta, se determină o nouă cifră. Procedând la fel şi pentru aceasta şi pentru toate cele care urmează se va ajunge din nou la prima cifră.

Dacă toate cifrele au fost vizitate exact o dată, numărul se numeşte *circular*.

De exemplu numărul

1894256

este număr circular deoarece:

- are numai cifre distincte
- nu conţine cifra 0
- pornind de la 1 obţinem, pe rând: 8, 9, 2, 6, 5, 4, 1

Cerinţă

Scrieţi un program care, pentru un N dat, determină câte numere circulare sunt mai mici sau egale cu N, precum şi cel mai mare număr circular mai mic sau egal cu N.

Date de intrare

Pe prima linie a fişierului de intrare **circular.in** se află numărul natural N.

Date de ieşire

Fişierul de ieşire **circular.out** conţine o singură linie, pe care se află numărul de numere circulare mai mici ca N precum şi numărul circular maxim cerut, separate printr-un spaţiu.

Dacă nu există nici un număr circular mai mic ca N, în fişierul de ieşire se vor afişa două valori 0 separate printr-un spaţiu.

Restricţii

$10 \leq N < 10.000.000$

Exemplu

circular.in	circular.out	Semnificaţie
1894250	347 1849625	Există 347 numere circulare mai mici dacât 1894250 cel mai mare dintre acestea fiind numărul 1849625

Timp de execuţie: 1 sec/test

10.3.1 Indicaţii de rezolvare - descriere soluţie *

Soluţie prezentată de Mihai Stroe, GInfo nr. 13/6

O rezolvare foarte bună a acestei probleme se bazează pe faptul că există foarte puţine numere circulare între 10 şi 10.000.000. Acestea pot fi generate în timpul concursului, obţinându-se un fişier cu toate numerele circulare, câte 10 pe linie şi separate prin virgulă. Acest fişier poate fi transformat uşor într-un vector de constante care poate fi folosit de un program Pascal sau C/C++ care, astfel conţine toate numerele circulare. Acest program va citi valoarea lui N, apoi va rezolva cerinţele parcurgând vectorul de constante. Deci, concurentul va elabora două programe: cel de generare a numerelor circulare şi cel care, folosindu-le, rezolvă cerinţele problemei.

Pentru generarea numerelor circulare se poate folosi un ciclu for până la 10.000.000, în cadrul căruia se testează, pe rând, pentru fiecare număr dacă este circular. Un mod simplu de a realiza această testare este de a transforma numărul într-un vector de cifre, prin împărţiri succesive la 10 cu salvarea restului.

Se observă că se vor obţine cifrele numărului în ordine inversă; este foarte uşor să se restabilească ordinea corectă prin inversarea primei cifre cu ultima, a celei de-a doua cu penultima etc.

Pentru un număr reprezentat ca un vector de cifre se poate verifica foarte uşor dacă este format din cifre distincte şi nu conţine cifra 0; pentru condiţia rămasă se va număra spre dreapta, iar poziţiile pe care se opreşte numărarea vor fi marcate cu 0. Dacă la un moment dat se va ajunge pe o poziţie marcată cu 0, fără a se fi marcat în prealabil toate poziţiile, atunci numărul nu este circular.

O metodă mai bună şi mai rapidă ar consta în generarea tuturor vectorilor de cifre distincte nenule de cel mult 7 cifre, folosind metoda backtracking.

O altă metodă, şi mai rapidă, ar consta în generarea directă a numerelor circulare, folosind o altă variantă a metodei backtracking. Astfel, după încercarea de a folosi cifra C_1 pe prima poziţie, se va trece la poziţia $1 + C_1$ (de fapt se vor face C_1 avansări spre dreapta) şi se va continua de acolo; dacă acolo se va folosi cifra C_2, se avansează încă C_2 poziţii spre dreapta etc.

La fiecare pas din backtracking se folosesc cifrele cuprinse între 1 şi 9, care nu au fost folosite anterior; dacă avansarea spre dreapta duce într-o poziţie deja completată şi mai sunt poziţii libere, numărul în construcţie nu poate fi circular şi se revine la cifra precedentă.

Analiza complexităţii

Pentru această problemă, complexitatea rezolvării este dată de numărul M de numere circulare, deci soluţia rulează instantaneu, deci ordinul de complexitate al acesteia ar fi $O(log_2 M)$, în cazul în care se foloseşte o căutare binară.

Deoarece M este o constantă, ordinul de complexitate al soluţiei este $O(1)$.

Complexitatea metodei de generare a numerelor circulare depinde de metoda folosită.

10.3.2 Rezolvare detaliată

10.3.3 Codul sursă *

```
import java.io.*; // Timp = 18sec pentru  n=9.999.999 (448 --> 9.682.415)
class Circular1 // Timp = 1.8sec pentru n=1.000.000
{ // Timp = 0.9sec pentru n=  555.555 (3 teste peste !!!)
  static int n, nnc=0,ncmax=0;
  static int[] a=new int[7]; // vectorul cifrelor
  static int[] aa=new int[7]; // traseu!
  static int[] fc=new int[10];// frecventa cifrelor
  static int nc;

  public static void main(String[]args) throws IOException
  {
    int k;
    long t1,t2;
    t1=System.currentTimeMillis();
    PrintWriter out = new PrintWriter(
                      new BufferedWriter(new FileWriter("circular.out")));
    StreamTokenizer st=new StreamTokenizer(
                       new BufferedReader(new FileReader("circular.in")));
    st.nextToken(); n=(int)st.nval;

    for(k=12;k<=n;k++)
      if(esteCircular(k)) { nnc++; ncmax=k; }

    out.println(nnc+" "+ncmax);
    out.close();
    t2=System.currentTimeMillis();
    System.out.println("Timp = "+(t2-t1));
  }//main

  static boolean esteCircular(int k)
  {
    int i,j;
    int c; // cifra
    nc=0;
    for(i=0;i<=9;i++) fc[i]=0;
    while(k!=0) {c=k%10; a[nc]=c; fc[c]++; k=k/10; nc++;}
    if(fc[0]>0) return false;
    for(i=1;i<=9;i++) if(fc[i]>1) return false;
    for(i=0;i<nc/2;i++) {c=a[i]; a[i]=a[nc-1-i]; a[nc-1-i]=c;}
    for(i=0;i<nc;i++) aa[i]=a[i];
```

```
    j=0;
    for(i=1;i<=nc;i++)
    if(aa[j]==0) return false; else { aa[j]=0; j=(j+a[j])%nc; }
    if(j==0) return true; else return false;
  }
}//class

import java.io.*;
class Circular2
{
  static int n, nnc=0,ncmax=0;
  static int[] x=new int[7];
  static int[] a=new int[7];
  static int[] aa=new int[7];
  static int ncn; // nr cifrelor lui n
  static int nc; // nc=2, ..., ncn (lg numerelor generate)

  public static void main(String[] args) throws IOException
  {
    long t1,t2;
    t1=System.currentTimeMillis();
    PrintWriter out = new PrintWriter(
                      new BufferedWriter(new FileWriter("circular.out")));
    StreamTokenizer st=new StreamTokenizer(
                       new BufferedReader(new FileReader("circular.in")));
    st.nextToken(); n=(int)st.nval;
    int k=n;
    ncn=0;
    while(k!=0) {k=k/10; ncn++;}

    for(nc=2;nc<=ncn;nc++) f(0);

    out.println(nnc+" "+ncmax);
    out.close();
    t2=System.currentTimeMillis();
    System.out.println("Timp = "+(t2-t1));
  }

  static void f(int k)
  {
    boolean ok;
    int i,j;
    for(i=1;i<=9;i++) // pun valoarea i pe pozitia k in vectorul a
    {
```

```
      ok=true;
      if(k>0)
        for(j=0;j<k;j++) if(i==x[j]) {ok=false; break;}

      if(!ok) continue;
      x[k]=i;
      if(k<nc-1) f(k+1); else afisv();
    }
  }

  static void afisv()
  {
    if(!esteCircular(x)) return;
    if(numar(x)>n) return;
    nnc++;
    ncmax=numar(x);
    //System.out.print(nnc+" : ");
    //for(int i=0; i<=nc-1; i++) System.out.print(x[i]);
    //System.out.println();
  }

  static int numar(int[] x)
  {
    int nx=x[0];
    for(int i=1;i<=nc-1;i++) nx=nx*10+x[i];
    return nx;
  }

  static boolean esteCircular(int[] x)
  {
    int i,j;
    for(i=0;i<nc;i++) a[i]=aa[i]=x[i];

    j=0;
    for(i=1;i<=nc;i++)
    if(aa[j]==0) return false; else { aa[j]=0; j=(j+a[j])%nc; }
    if(j==0) return true; else return false;
  }
}
```

10.4 Criptare

Mircea şi Vasilică vor să-şi trimită mesaje pe care alţii să nu le înţeleagă. Au citit ei despre spioni şi modalităţi de a scrie mesaje şi, în final, au imaginat un mod de criptare a unui mesaj care foloseşte "cuvânt cheie" (le-a plăcut lor denumirea asta :-)).

Alegându-şi un cuvânt cheie format numai din litere distincte, ei numără literele acestuia şi împart mesajul în grupe de lungime egală cu numărul de litere ale cuvântului cheie, şi le aşează una sub alta. Desigur, se poate întâmpla ca ultima grupă să fie incompletă, aşa că o completează cu spaţii.

Apoi numerotează literele cuvântului cheie în ordinea apariţiei lor în alfabetul englez.

În final, rescriu mesajul astfel: coloana de sub litera numerotată cu 1, urmată de coloana de sub litera numerotată cu 2, etc. înlocuind totodată şi spaţiile cu caracterul '*' (asterisc).

Exemplu:

cuvântul cheie: **criptam**
mesaj de criptat: Incercam sa lucram cu coduri si criptari.

cuvântul cheie **criptam** are 7 litere
numerotare: **2635714**

deoarece, avem, în ordine

a	b	**c**	defgh	**i**	jkl	**m**	no	**p**	q	**r**	s	**t**	uvwxzy
1		2		3		4		5		6		7	

împărţire în grupe: Incerca|m sa lu|cram cu| coduri| si cri|ptari.

codificare:

2	6	3	5	7	1	4
I	n	c	e	r	c	a
m	*	s	a	*	l	u
c	r	a	m	*	c	u
*	c	o	d	u	r	i
*	s	i	*	c	r	i
p	t	a	r	i	.	*

mesaj criptat: clcrr.Imc**pcsaoiaauuii*eamd*rn*rcstr**uci

clcrr.	Imc**p	csaoia	auuii*	eamd*r	n*rcst	r**uci
col1	col2	col3	col4	col5	col6	col7

Cerinţă

Fiind date un cuvânt cheie şi un mesaj criptat, decodificaţi mesajul trimis de Mircea pentru Vasilică.

Date de intrare

Fişierul de intrare **criptare.in** conţine pe prima linie mesajul criptat iar pe linia a doua cuvântul cheie.

Date de ieşire

Fişierul de intrare **criptare.out** conţine pe prima linie mesajul decriptat.

Restricţii

- lungimea mesajului este de minim 20 şi maxim 1000 caractere
- cuvântul cheie are minim 5 şi maxim 20 de caractere
- cuvântul cheie conţine numai litere mici ale alfabetului

Exemplu

criptare.in
clcrr.Imc**pcsaoiaauuii*eamd*rn*rcstr**uci criptam

criptare.out
Incercam sa lucram cu coduri si criptari.

Timp maxim de execuţie: 1 secundă/test

10.4.1 Indicaţii de rezolvare - descriere soluţie *

Soluţie prezentată de Mihai Stroe, GInfo nr. 13/6

Pentru decriptarea mesajului se utilizează algoritmul de criptare descris, în sens invers.

Se determină ordinea literelor din cuvântul cheie. Considerând că acest cuvânt are lungimea L, iar textul are lungimea N, el va fi împărţit în L grupe de lungimea N/L.

Grupele sunt aşezate într-o matrice cu L coloane pe coloanele corespunzătoare ordinii literelor din cuvântul cheie.

În final, matricea este parcursă pe linii şi se afişează mesajul decriptat.

Analiza complexităţii

Operaţiile de citire şi afişare a datelor au ordinul de complexitate $O(N)$.

Deoarece lungimea L a cuvântului cheie este mică, operaţiile cu acesta (determinarea ordinii literelor) pot fi neglijate la calculul complexităţii.

Scrierea cuvântului în matrice are ordinul de complexitate $O(N)$.

În final, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(N) + O(N) = O(N)$.

10.4.2 Rezolvare detaliată

```
import java.io.*;
class Criptare1
{
  public static void main(String[]args)throws IOException
  {
    int i,j,k;
    BufferedReader br=new BufferedReader(
                      new FileReader("criptare.in"));
    String mesaj=br.readLine();
    String cheia=br.readLine();
    br.close();

    System.out.println("mesajul este:  "+mesaj+" "+mesaj.length());
    System.out.println("cheia este:  "+cheia+" "+cheia.length());

    int m=mesaj.length()/cheia.length(); // nr linii matrice
    int n=cheia.length(); // nr coloane matrice

    int[] p=new int[n];
    // sortare prin numarare
    for(j=1; j<=n-1; j++)
      for(i=0;i<=j-1;i++)
        if(cheia.charAt(i)<cheia.charAt(j)) ++p[j]; else ++p[i];
    afisv(p);
    int[] pp=new int[n]; // permutarea inversa
    for(i=0;i<n;i++) pp[p[i]]=i;
    afisv(pp);

    char a[][]=new char[m][n];

    k=0;
    for(j=0;j<n;j++)
      for(i=0;i<m;i++)
      {
        a[i][pp[j]]=mesaj.charAt(k);
        k++;
      }

    System.out.println("Matricea a[][] este:");
    for(i=0;i<m;i++)
    {
      for(j=0;j<n;j++) System.out.print(a[i][j]+" ");
      System.out.println();
    }
```

```
    for(i=0;i<m;i++)
      for(j=0;j<n;j++)
        if(a[i][j]!='*') System.out.print(a[i][j]);
        else System.out.print(" ");
    System.out.println();
  }//main

  static void afisv(int[] x)
  {
    int n=x.length, i;
    for(i=0;i<n;i++) System.out.print(x[i]+" ");
    System.out.println();
  }
}//class:
```

10.4.3 Codul sursă *

```
import java.io.*;
class Criptare2
{
  public static void main(String[]args)throws IOException
  {
    int i,j,k;
    BufferedReader br=new BufferedReader(
                      new FileReader("criptare.in"));
    PrintWriter out=new PrintWriter(
                    new BufferedWriter(
                    new FileWriter("criptare.out")));
    String mesaj=br.readLine();
    String cheia=br.readLine();
    br.close();

    int m=mesaj.length()/cheia.length(); // nr linii matrice
    int n=cheia.length(); // nr coloane matrice

    int[] p=new int[n];
    // sortare prin numarare
    for(j=1; j<=n-1; j++)
      for(i=0;i<=j-1;i++)
        if(cheia.charAt(i)<cheia.charAt(j)) ++p[j]; else ++p[i];
    int[] pp=new int[n]; // permutarea inversa
```

```
    for(i=0;i<n;i++) pp[p[i]]=i;

    char a[][]=new char[m][n];

    k=0;
    for(j=0;j<n;j++)
      for(i=0;i<m;i++)
      {
        a[i][pp[j]]=mesaj.charAt(k);
        k++;
      }

    for(i=0;i<m;i++)
      for(j=0;j<n;j++)
        if(a[i][j]!='*') out.print(a[i][j]); else out.print(" ");
    out.close();
  }
}
```

10.5 Maşina

O ţară are $3 \leq N \leq 30000$ oraşe, numerotate de la 1 la N, dispuse pe un cerc.

PAM tocmai şi-a luat carnet de conducere şi vrea să viziteze toate oraşele ţării. Lui PAM îi este frică să conducă prin locuri aglomerate aşa că ea şi-a propus să meargă numai pe şoselele unde traficul este mai redus.

Există şosele de legătură între oricare două oraşe alăturate: între oraşul 1 şi oraşul 2, ... , între oraşul i şi oraşul $i + 1$, iar oraşul N este legat de oraşulul 1.

Ca să nu se rătăcească, PAM şi-a propus să-şi aleagă un oraş de început şi să meargă pe şoselele respective în sens trigonometric până ajunge înapoi în oraşul de unde a plecat. Dacă PAM pleacă din oraşul K, atunci traseul ei va fi: $K, K + 1, ..., N, 1, 2, ..., K$.

Maşina lui PAM are un rezervor foarte mare (în care poate pune oricât de multă benzină). În fiecare oraş, PAM ia toată cantitatea de benzină existentă în oraş, iar parcurgerea fiecărei şosele necesită o anumită cantitate de benzină.

Cerinţă

Ştiind că PAM are, la începutul călătoriei, doar benzina existentă în oraşul de plecare, şi că, atunci când ajunge într-un oraş, ea va lua toată cantitatea de benzină disponibilă în acel oraş, să se găsească un oraş din care PAM îşi poate începe excursia astfel încât să nu rămână fără benzină.

Se consideră că PAM a rămas fără benzină dacă în momentul plecării dintr-un oraş, nu are suficientă benzină pentru a parcurge şoseaua care duce la oraşul

următor. Dacă benzina îi ajunge la fix (adică la plecare are tot atâta benzină câtă îi trebuie) se consideră că PAM poate ajunge până în oraşul următor.

Date de intrare

Fişierul de intrare **masina.in** conţine

• pe prima linie numărul N,

• pe cea de-a doua linie se găsesc N numere naturale $a[1]$, $a[2]$, ..., $a[N]$, separate prin câte un spaţiu, unde $a[i]$ reprezintă cantitatea de benzină disponibilă în oraşul i.

• linia a treia conţine un şir de N numere naturale $b[1]$, $b[2]$, ..., $b[N]$, separate prin câte un spaţiu, unde $b[i]$ reprezintă cantitatea de benzină necesară străbaterii şoselei dintre oraşele i şi $i+1$ (sau N şi 1, dacă $i = N$).

Date de ieşire

Fişierul de ieşire **masina.out** va conţine un singur număr s care reprezintă un oraş din care, dacă PAM îşi începe călătoria, poate completa turul ţării fără a face pana prostului.

Restricţii şi precizări

• Dacă există mai multe soluţii, se cere una singură.

• $0 \leq a[i] \leq 30000$

• $1 \leq b[i] \leq 30000$

Exemplu

masina.in	masina.out
6 0 3 2 5 10 5 7 8 3 2 1 4	4

Timp maxim de execuţie: 0.3 sec/test

10.5.1 Indicaţii de rezolvare - descriere soluţie *

Soluţie prezentată de Mihai Stroe, GInfo nr. 13/6

Având în vedere că oraşele sunt aşezate şi parcurse circular, putem dubla oraşele, iar apoi putem să le considerăm aşezate liniar:

$a_{n+1} = a_1, a_{n+2} = a_2, ..., a_{n+k} = a_k, ..., a_{2n} = a_n$

$b_{n+1} = b_1, b_{n+2} = b_2, ..., b_{n+k} = b_k, ..., b_{2n} = b_n$.

Cantitatea de benzină câştigată în urma opririi în oraşul i şi parcurgerii drumului către oraşul $i+1$ este $c_i = a_i - b_i$. Evident, această cantitate poate fi negativă.

Definim şirul sumelor parţiale ale vectorului c astfel:

$s_i = \sum_{k=1}^{i} s_i$.

Pentru ca PAM să poată efectua turul, este necesar ca suma cantităţii de benzină acumulate să fie mai mare sau egală cu 0, deci $s_n = 0$.

Rezolvarea problemei constă în a găsi un x astfel încât
$c_x \geq 0$, $c_x + c_{x-1} \geq 0$, ..., $c_x + c_{x+n-1} \geq 0$,
adică
$s_x - s_{x-1} \geq 0$, $s_{x+1} - s_{x-1} \geq 0$,..., $s_{x+n-1} - s_{x-1} \geq 0$.
Un candidat viabil este poziţia $m \leq n$ pentru care s_m este minim.
În continuare se demonstrează că această alegere conduce la o soluţie.
Fie $m < k \leq n$; avem $s_k \geq s_m$ (din definiţia lui m).
Fie $n < k$; avem
$s_k = c_1 + ... + c_n + c_{n+1} + ... + c_k = s_n + c_1 + ... + c_k = s_n + s_k \geq s_k \geq s_m$
(am folosit faptul că $s_n \geq 0$).
În concluzie $x = m + 1$ este o soluţie pentru problemă.
Implementarea constă în citirea datelor, calcularea şirului sumelor parţiale şi găsirea minimului dintre $s_1, s_2, ..., s_n$.

Analiza complexităţii

Operaţia de citire a datelor are ordinul de complexitate $O(n)$.
Calculul şirului sumelor parţiale are ordinul de complexitate $O(n)$ pentru că $s_i = s_{i-1} + a_i$.
Minimul din acest şir se calculează cu aceeaşi complexitate.
Operaţia de scriere a rezultatului are ordinul de complexitate $O(1)$.
În final, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(n) + O(n) + O(1) = O(n)$.

10.5.2 Rezolvare detaliată

10.5.3 Codul sursă *

```
import java.io.*;
class Masina
{
public static void main(String[] args) throws IOException
{
int i,rez;
int j,n;
StreamTokenizer st=new StreamTokenizer(
new BufferedReader(new FileReader("masina.in")));
PrintWriter out=new PrintWriter(
new BufferedWriter(new FileWriter("masina.out")));

st.nextToken();n=(int)st.nval;
int[] a=new int[n+1];
```

```
int[] b=new int[n+1];

for (j=1;j<=n;j++){ st.nextToken();a[j]=(int)st.nval;}
for (j=1;j<=n;j++){ st.nextToken();b[j]=(int)st.nval;}

for(i=1;i<=n;i++)
{
rez=a[i]-b[i];
j=i;
j++; if(j==n+1) j=1;
while((rez>=0)&&(j!=i))
{
rez=rez+a[j]-b[j];
j++;if(j==n+1) j=1;
}
if(rez>=0) {out.println(i); break;}
}
out.close();
}
}
```

10.6 Operaţii

Notăm cu c şi r câtul şi respectiv restul împărţirii unui numar nr la 2^k, unde k este un număr natural nenul.

Asupra numărului putem efectua succesiv următoarele operaţii:

• $O1(nr, k)$ reprezintă transformarea numărului nr în numărul $2^k(2c+1)+r$ pentru orice rest r, sau

• $O2(nr, k)$ reprezintă transformarea numărului nr în numărul $2^{k-1}c+r$ doar dacă $r < 2k-1$.

Cerinţă

Se dau m şi n două numere naturale nenule.

Efectuaţi asupra numerelor m şi n operaţii succesive, $O1$ sau $O2$, pentru valori alese ale lui k, astfel încât după un număr finit de operaţii cele două numere să devină egale, iar valoarea astfel obţinută să fie minimă.

Date de intrare

Fişierul de intrare **operatii.out** conţine pe o linie m n − două numere naturale nenule, separate printr-un spaţiu, reprezentând cele două numere date.

Date de ieşire

Fişierul de ieşire **operatii.out** conţine pe cele $i+j+3$ linii următoarele:

• $nmin$ − numărul natural nenul **nmin**, reprezentând valoarea minimă obţinută din m şi n prin aplicarea unor succesiuni de operaţii $O1$ sau $O2$,

• i − numărul operaţiilor efectuate asupra primului număr m, pe a doua linie
• op_1 k_1 − pe următoarele i linii: perechi de numere
• ... − reprezentând operaţia (1 sau 2) şi respectiv valoarea lui k
• op_i k_i − pentru operaţia respectivă, separate printr-un spaţiu.
• j − numărul operaţiilor efectuate asupra celui de al doilea număr n, pe linia i+2
• op_1 k_1 − pe următoarele j linii: perechi de numere
• ... − reprezentând operaţia (1 sau 2) şi respectiv valoarea lui k
• op_j k_j − pentru operaţia respectivă, separate printr-un spaţiu

Restricţii
$1 < m, n \leq 2.000.000.000$

Exemplu

OPERATII.IN	OPERATII.OUT
11 45	15 2 2 3 1 2 2 2 2 2 4

Timp maxim de execuţie: 1 sec/test

10.6.1 Indicaţii de rezolvare - descriere soluţie *

Soluţie prezentată de Mihai Stroe, GInfo nr. 13/6

Pentru rezolvarea problemei este necesară descompunerea în binar a celor două numere şi înţelegerea efectului pe care îl au cele două operaţii asupra acestei descompuneri.

Se observă că operaţia $O1$ inserează o cifră 1 în descompunere, pe o anumită poziţie, în timp ce operaţia $O2$ şterge un 0, la alegere.

Privind astfel cele două operaţii, algoritmul de rezolvare începe cu descompunerea numerelor în baza 2 şi ştergerea tuturor cifrelor 0 din ambele numere. Se vor obţine astfel două numere, nu neapărat egale, ale căror descompuneri în baza 2 sunt formate numai din cifre 1 (deci numerele sunt de forma $2 \cdot P - 1$).

Cum cifrele 1 nu pot fi şterse, singura variantă de a ajunge la acelaşi număr constă în inserarea de cifre 1 în cadrul numărului mai mic.

La efectuarea operaţiilor $O2$ pe fiecare număr, acestea se memorează pentru a fi afişate.

Pentru operaţiile $O1$ nu este necesară memorarea; ele se vor efectua numai asupra numărului care rămâne mai mic, valoarea pentru K putând fi 1 pentru toate aceste operaţii.

Analiza complexităţii

Operaţia de citire a datelor are ordinul de complexitate $O(1)$.

De asemenea, ordinul de complexitate al operaţiilor de descompunere a numerelor în baza 2, de detectare a zerourilor din descompunere, de memorare a poziţiilor operaţiilor de tip 2, de detectare a numărului mai mic şi afişarea mutărilor este $O(1)$.

În final, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este $O(1)$.

10.6.2 Rezolvare detaliată

10.6.3 Codul sursă *

```
import java.io.*;
class Operatii
{
  static int m, n;
  static int[] a=new int[64]; // vectorul cifrelor binare ale lui m
  static int[] b=new int[64]; // vectorul cifrelor binare ale lui n
  static int nca,ncb, nc1a, nc0a, nc1b,nc0b, nc1sol,nminsol;

  public static void main(String[]args) throws IOException
  {
    int i,j,k,nr;
    PrintWriter out = new PrintWriter(
                      new BufferedWriter(new FileWriter("operatii.out")));
    StreamTokenizer st=new StreamTokenizer(
                       new BufferedReader(new FileReader("operatii.in")));
    st.nextToken(); m=(int)st.nval;
    st.nextToken(); n=(int)st.nval;

    nr=m;
    k=0;
    while(nr!=0) { a[k]=nr%2; k++; nr=nr/2;}
    nca=k;
    nc1a=0;
    for(k=0;k<nca;k++) if(a[k]==1) nc1a++;
    nc0a=nca-nc1a;

    nr=n;
```

```
    k=0;
    nc1b=0;
    while(nr!=0) { b[k]=nr%2; k++; nr=nr/2;}
    ncb=k;
    for(k=0;k<ncb;k++) if(b[k]==1) nc1b++;
    nc0b=ncb-nc1b;

    if(nc1a<nc1b) nc1sol=nc1b; else nc1sol=nc1a;
    nminsol=1;
    for(k=1;k<=nc1sol;k++) nminsol=nminsol*2;
    nminsol=nminsol-1;
    out.println(nminsol);

    out.println(nc1sol-nc1a+nc0a);
    j=0;
    for(k=1;k<=nc0a;k++)
    {
      while(a[j]!=0) j++;
      out.println(2+" "+(j-k+2)); // sterg 0 de pe pozitia j
      j++;
    }
    for(k=nc1a;k<nc1sol;k++) out.println("1 1");

    out.println(nc1sol-nc1b+nc0b);
    j=0;
    for(k=1;k<=nc0b;k++)
    {
      while(b[j]!=0) j++;
      out.println(2+" "+(j-k+2)); // sterg 0 de pe pozitia j
      j++;
    }
    for(k=nc1b;k<nc1sol;k++) out.println("1 1");

    out.close();
  }//main
}//class
```

Capitolul 11

ONI 2004 clasa a IX-a

11.1 Coduri

Un detectiv particular are de rezolvat un caz special.

Este vorba de o deturnare de fonduri. Pentru a putea rezolva cazul trebuie să găsească un şir cu n coduri distincte. Fiecare cod este un număr natural scris în baza 10.

Din păcate lucrurile nu sunt simple, pentru că din cercetările efectuate a obţinut două informaţii. Prima informaţie este legată de faptul că suma pătratelor codurilor este un cub perfect, iar a doua spune că suma cuburilor codurilor este un pătrat perfect.

Cerinţă

Ajutaţi detectivul să găsească un şir de coduri $x_1, x_2, ..., x_n$, care verifică condiţiile din enunţ şi $x_i \leq n^{14}$, pentru orice i cu $1 \leq i \leq n$.

Date de intrare

Fişierul de intrare **coduri.in** conţine pe prima linie numărul natural n.

Date de ieşire

Fişierul de ieşire **coduri.out** va conţine n linii, câte una pentru fiecare cod din şir, în ordine crescătoare.

Restricţii

• $1 \leq n \leq 20$

Exemplu

coduri.in	coduri.out
2	625 1250

Timp maxim de execuţie: 1 sec/test

11.1.1 Indicaţii de rezolvare - descriere soluţie *

Fie $s = n(n+1)(2n+1)/6$. Se pot considera codurile $x_k = k \cdot s^4$, $1 \leq k \leq n$.

11.1.2 Rezolvare detaliată *

Codurile $x_1, x_2, ..., x_n$ trebuie să îndeplinească următoarele condiţii:

$$x_1^2 + x_2^2 + ... + x_n^2 = a^3 \text{ şi } x_1^3 + x_2^3 + ... + x_n^3 = b^2$$

unde a şi b sunt nişte numere naturale.

Aceste relaţii ne fac să ne gândim la formulele:

$$1^2 + 2^2 + ... + n^2 = \frac{n(n+1)(2n+1)}{6} \text{ şi } 1^3 + 2^3 + ... + n^3 = \left[\frac{n(n+1)}{2}\right]^2$$

şi ne sugerează să considerăm codurile sub forma $x_k = \alpha \cdot k$.

Pentru suma cuburilor obţinem

$$x_1^3 + x_2^3 + ... + x_n^3 = \alpha^3 \left[\frac{n(n+1)}{2}\right]^2$$

şi pentru a obţine un pătrat perfect vom considera $\alpha = \beta^2$, deci codurile vor fi considerate sub forma $x_k = \beta^2 \cdot k$.

Acum suma cuburilor

$$x_1^3 + x_2^3 + ... + x_n^3 = \beta^6 \left[\frac{n(n+1)}{2}\right]^2$$

este un pătrat perfect.

Suma pătratelor codurilor este

$$x_1^2 + x_2^2 + ... + x_n^2 = \beta^4 \frac{n(n+1)(2n+1)}{6} = \beta^3 \cdot \beta \frac{n(n+1)(2n+1)}{6}$$

şi pentru a obţine un cub perfect putem considera

$$\beta = \left[\frac{n(n+1)(2n+1)}{6}\right]^2.$$

Astfel, putem considera codurile sub forma

$$x_k = k \cdot \left[\frac{n(n+1)(2n+1)}{6}\right]^4.$$

11.1.3 Codul sursă *

```
import java.io.*;
class Coduri
{
  public static void main(String[] args) throws IOException
  {
    int n;
    int k;
    long s;
    long[] x;
    PrintWriter out = new PrintWriter(
                      new BufferedWriter(new FileWriter("coduri.out")));
    StreamTokenizer st=new StreamTokenizer(
                       new BufferedReader(new FileReader("coduri.in")));
    st.nextToken(); n=(int)st.nval;
    x=new long[n+1];
    s=n*(n+1)*(2*n+1)/6;
    s=s*s*s*s;
    for(k=1;k<=n;k++) x[k]=k*s;
    for(k=1;k<=n;k++) out.println(x[k]);
    out.close();
  }
}
```

11.2 Logic

Demonstrarea automată a teoremelor şi verificarea satisfiabilităţii unei formule constituie două capitole importante în cadrul logicii matematice.

Formulele propoziţionale sunt alcătuite din variabile propoziţionale (variabile care pot lua doar două valori: sau adevărat sau fals) şi din operatorii logici *şi, sau, negaţie, echivalent, implică*.

Iată câteva exemple de formule propoziţionale:

$\tilde{}p\&(q <=> p) => q$
$p|q <=> \tilde{}p\&\tilde{}q$
p
$p => q => a => t => \tilde{}p$

În acest exemplu, p şi q sunt *variabilele propoziţionale*, $\tilde{}$ este operatorul unar *negaţie*, & este operatorul binar *şi*, | este operatorul binar *sau*, => este *implicaţia* logică (şi apare numai în acest sens, nu şi <=), iar <=> este *echivalenţa* logică.

În plus, într-o *formulă propoziţională* pot să apară şi paranteze care stabilesc ordinea operaţiilor. În lipsa parantezelor, operatorii în ordinea priorităţii lor sunt & | => <=>.

În formulele de forma "$A_1\mathbf{op}A_2\mathbf{op}...bfopA_K$" asociaţiile se fac de la dreapta la stânga (adică "$A_1\mathbf{op}(A_2\mathbf{op}(...\mathbf{op}A_K)...)$"), unde **op** este unul dintre &, |, => sau <=> şi A_i sunt formule propoziţionale, cu i de la 1 la K.

În general, o formulă propoziţională se defineşte astfel:
– orice variabilă propoziţională este formulă propoziţională
– dacă A şi B sunt formule propoziţionale, atunci (A), $\tilde{}A$, $A\&B$, $A|B$, $A => B$, $A <=> B$ sunt formule propoziţionale.

Dacă înlocuim într-o *formulă propoziţională* toate *variabilele* cu *valori de adevăr* (*adevărat* sau *fals*), obţinem o *afirmaţie*.

Valoarea de adevăr a unei *afirmaţii* este dată de următoarea definiţie:
– dacă afirmaţia constă dintr-o singură valoare de adevăr, afirmaţia ia valoarea respectivă
– dacă A şi B sunt afirmaţii, atunci:

- A este adevărată dacă şi numai dacă valoarea sa de adevăr este *adevărat*
- (A) este adevărată dacă şi numai dacă este adevărată A
- ˜A este falsă dacă şi numai dacă A este adevărată
- $A\&B$ este adevărată dacă şi numai dacă atât A cât şi B sunt adevărate
- $A|B$ este falsă dacă şi numai dacă A este fals şi B este fals
- $A => B$ este adevărată dacă şi numai dacă $\tilde{}A|B$ este adevărată
- $A <=> B$ este adevărată dacă şi numai dacă $(A => B)\&(B => A)$ este adevărată.

Se numeşte *soluţie* a formulei propoziţionale P (formulă în care apar numai variabilele propoziţionale distincte $A_1, A_2, ..., A_N$) orice N-uplu

$(v_1, v_2, ..., v_N)$

(cu v_i *valori de adevăr*) pentru care înlocuind fiecare variabilă A_i cu valoarea v_i, afirmaţia rezultată este *adevărată*.

Cerinţă

Logica fiind un obiect nesuferit de studenţii de la informatică, ei apelează la informaticienii din clasa a IX-a pentru a-i ajuta să numere câte soluţii distincte are o formulă propoziţională dată.

Date de intrare
În fişierul de intrare **logic.in** se găseşte o formulă propoziţională unde variabilele propoziţionale sunt reprezentate de litere mici ale alfabetului englez.

Date de ieşire
În fişierul de ieşire **logic.out** se va afişa numărul de soluţii pentru formula propoziţională din fişierul de intrare, urmat de caracterul sfârşit de linie.

Restricţii şi precizări
• La intrare se va da întotdeauna o formulă propoziţională corectă sintactic
• Formula are mai puţin de 232 de caractere
• În formulă nu apar mai mult de 10 litere mici ale alfabetului latin

Exemplu:

logic.in	logic.out
$p\|q <=> p\& q$	4
A	1

Timp execuţie: 1 sec/test.

11.2.1 Indicaţii de rezolvare - descriere soluţie *

Informaţia *"în formulă nu apar mai mult de 10 litere mici ale alfabetului latin"* ne conduce la ideea de a genera toate configuraţiile (sunt cel mult $2^{10} = 1024$) şi a calcula valoarea de adevăr a formulei pentru fiecare configuraţie. Se foloseşte recursivitatea ţinând cont de priorităţile operatorilor.

11.2.2 Rezolvare detaliată *

O vizualizare a codurilor operatorilor:

```
import java.io.*;
class Logic0
{
  static char[] e; // expresia

  public static void main(String[]args) throws IOException
  {
    int i,n;
    PrintWriter out = new PrintWriter(
                      new BufferedWriter(new FileWriter("logic.out")));
```

```
    BufferedReader br=new BufferedReader(new FileReader("logic.in"));

    e=br.readLine().toCharArray();
    n=e.length;
    for(i=0;i<n;i++) System.out.print(e[i]);
    System.out.println();

    System.out.println("~ "+(int)'~');
    System.out.println("& "+(int)'&');
    System.out.println("| "+(int)'|');
    System.out.println("< "+(int)'<');
    System.out.println("= "+(int)'=');
    System.out.println("> "+(int)'>');
    System.out.println("a "+(int)'a');
    System.out.println("z "+(int)'z');
    out.close();
  }//main
}//class
```

Soluţia cu mesaje de urmărire a execuţiei:

```
import java.io.*; // prioritati: variabila ( ~ & | => <=> (descrescator!)
class Logic1      // la egalitate de la dreapta la stanga (recursivitate)
{
  static char[] e;                    // expresia
  static char[] c=new char[10];       // codurile variabilelor din expresie
  static boolean[] v=new boolean[127]; // valoarea logica a variabilelor
  static int nv=0;                    // nr variabile in expresie
  static int poz;                     // poz in expresie
  static int n;                       // lungime expresie

  public static void main(String[]args) throws IOException
  {
    int resultat=0, i, j;
    int[] f=new int[127]; // frecventa caracterelor

    PrintWriter out = new PrintWriter(
                      new BufferedWriter(new FileWriter("logic.out")));
    BufferedReader br=new BufferedReader(new FileReader("logic.in"));
    e=br.readLine().toCharArray();
    n=e.length;
    for(i=0;i<n;i++) f[e[i]]++;
    for(i='a';i<='z';i++) if(f[i]>0) c[nv++]=(char)i;
    for(i=0;i<(1<<nv);i++)               // 1<<nv este 2^{nv}
    {
```

```
      for(j=0;j<nv;j++) v[(int)c[j]]=((i&(1<<j))>0)?true:false;
      poz=0;
      resultat+=(formula()?1:0);
      System.out.println("----------------- ");
    }
    out.println(resultat);
    out.close();
  }//main

  static boolean formula()
  {
    System.out.println("  --> formula "+poz+" "+e[poz]);
    boolean val;
    val=echivalenta();
    if(poz<n) System.out.println("<-- formula "+poz+" "+e[poz]);
      else System.out.println("<-- formula "+poz);
    return val;
  }

  static boolean echivalenta()
  {
    System.out.println("  --> echivalenta "+poz+" "+e[poz]);
    boolean a,b,val;
    a=implicatie();
    val=a;
    if((poz<n)&&(e[poz]=='<')) {poz+=3; b=formula(); val=(a==b);}
    if(poz<n) System.out.println("<-- echivalenta "+poz+" "+e[poz]);
      else System.out.println("<-- echivalenta "+poz);
    return val;
  }

  static boolean implicatie()
  {
    System.out.println("  --> implicatie "+poz+" "+e[poz]);
    boolean a, b, val;
    a=sau();
    val=a;
    if((poz<n)&&(e[poz]=='=')) {poz+=2; b=implicatie(); val=(!a)||b;}
    if(poz<n) System.out.println("<-- implicatie "+poz+" "+e[poz]);
      else System.out.println("<-- implicatie "+poz);
    return val;
  }

  static boolean sau()
```

```
{
  System.out.println("  --> sau "+poz+" "+e[poz]);
  boolean a, b, val;
  a=si();
  val=a;
  if((poz<n)&&(e[poz]=='|')) {poz++; b=sau(); val=(a||b);}
  if(poz<n) System.out.println("<-- sau "+poz+" "+e[poz]);
    else System.out.println("<-- sau "+poz);
  return val;
}

static boolean si()
{
  System.out.println("  --> si "+poz+" "+e[poz]);
  boolean a, b, val;
  a=not();
  val=a;
  if((poz<n)&&(e[poz]=='&')) {poz++; b=si(); val=(a&&b);}
  if(poz<n) System.out.println("<-- si "+poz+" "+e[poz]);
    else System.out.println("<-- si "+poz);
  return val;
}

static boolean not()
{
  boolean val;
  System.out.println("  --> not "+poz+" "+e[poz]);
  if(e[poz]=='~') {poz++; val=!not();}
  else val=paranteza();
  if(poz<n) System.out.println("<-- not "+poz+" "+e[poz]);
    else System.out.println("<-- not "+poz);
  return val;
}

static boolean paranteza()
{
  System.out.println("  --> paranteza "+poz+" "+e[poz]);
  boolean val;
  if(e[poz]=='(') {poz++;val=formula(); poz++;}
  else if(e[poz] == ')') val=false;
       else val=variabila();
  if(poz<n) System.out.println("<-- paranteza "+poz+" "+e[poz]);
    else System.out.println("<-- paranteza "+poz);
  return val;
```

```
  }

  static boolean variabila()
  {
    System.out.println("  --> variabila "+poz+" "+e[poz]);
    boolean val;
    if((poz<n)&&(e[poz]>='a')&&(e[poz]<='z')) val=v[(int)e[poz++]];
    else val=formula();
    if(poz<n) System.out.println("<-- variabila "+poz+" "+e[poz]);
      else System.out.println("<-- variabila "+poz);
    return val;
  }
}//class
```

11.2.3 Codul sursă *

```
import java.io.*; // prioritati: variabila ( ~ & | => <=> (descrescator!)
class Logic1      // la egalitate de la dreapta la stanga (recursivitate)
{
  static char[] e;                      // expresia
  static char[] c=new char[10];         // codurile variabilelor din expresie
  static boolean[] v=new boolean[127]; // valoarea logica a variabilelor
  static int nv=0;                      // nr variabile in expresie
  static int poz;                       // poz in expresie
  static int n;                         // lungime expresie

  public static void main(String[]args) throws IOException
  {
    int resultat=0, i, j;
    int[] f=new int[127]; // frecventa caracterelor

    PrintWriter out = new PrintWriter(
                      new BufferedWriter(new FileWriter("logic.out")));
    BufferedReader br=new BufferedReader(new FileReader("logic.in"));
    e=br.readLine().toCharArray();
    n=e.length;
    for(i=0;i<n;i++) f[e[i]]++;
    for(i='a';i<='z';i++) if(f[i]>0) c[nv++]=(char)i;
    for(i=0;i<(1<<nv);i++)               // 1<<nv este 2^{nv}
    {
      for(j=0;j<nv;j++) v[(int)c[j]]=((i&(1<<j))>0)?true:false;
      poz=0;
```

```
    resultat+=(formula()?1:0);
  }
  out.println(resultat);
  out.close();
}//main

static boolean formula()
{
  boolean val;
  val=echivalenta();
  return val;
}

static boolean echivalenta()
{
  boolean a,b,val;
  a=implicatie();
  val=a;
  if((poz<n)&&(e[poz]=='<')) {poz+=3; b=formula(); val=(a==b);}
  return val;
}

static boolean implicatie()
{
  boolean a, b, val;
  a=sau();
  val=a;
  if((poz<n)&&(e[poz]=='=')) {poz+=2; b=implicatie(); val=(!a)||b;}
  return val;
}

static boolean sau()
{
  boolean a, b, val;
  a=si();
  val=a;
  if((poz<n)&&(e[poz]=='|')) {poz++; b=sau(); val=(a||b);}
  return val;
}

static boolean si()
{
  boolean a, b, val;
  a=not();
```

```
    val=a;
    if((poz<n)&&(e[poz]=='&')) {poz++; b=si(); val=(a&&b);}
    return val;
  }

  static boolean not()
  {
    boolean val;
    if(e[poz]=='~') {poz++; val=!not();}
    else val=paranteza();
    return val;
  }

  static boolean paranteza()
  {
    boolean val;
    if(e[poz]=='(') {poz++;val=formula(); poz++;}
    else if(e[poz] == ')') val=false;
         else val=variabila();
    return val;
  }

  static boolean variabila()
  {
    boolean val;
    if((poz<n)&&(e[poz]>='a')&&(e[poz]<='z')) val=v[(int)e[poz++]];
    else val=formula();
    return val;
  }
}//class
```

11.3 Poligon

Se dă un caroiaj de $M \times N$ în care sunt plasate K puncte. Fiecare punct poate fi legat de vecinul său direct pe maxim opt direcţii (N, NE, E, SE, S, SV, V, NV).

Cerinţă

Determinaţi patrulaterele având vârfurile în punctele date iar laturile formate din legături între două sau mai multe puncte coliniare.

Date de intrare

Fişierul de intrare **poligon.in** conţine

• pe prima linie trei numere naturale nenule, separate prin câte un spaţiu,

$M\ N\ K$

reprezentând dimensiunile M, N ale caroiajului şi K numărul de puncte, iar

• pe următoarele K linii câte trei numere naturale separate printr-un spaţiu,

$I_i\ J_i\ V_i$

reprezentând coordonatele punctului i, $1 \leq i \leq K$ respectiv direcţiile spre care este legat de vecini direcţi.

Codificarea direcţiilor se face printr-un număr cuprins între 0 şi 255.

Reprezentarea binară a acestuia pe 8 cifre reprezintă, începând de la stânga spre dreapta, legătură pe direcţiile (1 - legatură, 0 - nu):

$N\ NE\ E\ SE\ S\ SV\ V\ NV$.

De exemplu: 1 0 0 0 0 1 1 0 = 134 deci legături spre N, SV, V

Date de ieşire

Fişierul de ieşire **poligon.out** conţine numai numărul natural

npol

reprezentând numărul patrulaterelor.

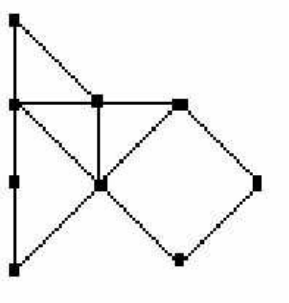
Restricţii

$1 < M, N \leq 100$

$4 \leq K \leq 50$

Exemplu

POLIGON.IN	POLIGON.OUT
4 4 9 1 1 24 2 1 184 2 2 43 2 3 22 3 1 136 3 2 213 3 4 5 4 1 192 4 3 65	6



Timp de execuţie: 2 sec/test

11.3.1 Indicaţii de rezolvare - descriere soluţie *

Se generează toate combinările de câte 4 puncte şi se verifică dacă acestea pot porma un patrulater (tinând cont de direcţii).

11.3.2 Rezolvare detaliată

11.3.3 Codul sursă *

```
import java.io.*;
class Poligon
{
  static int m,n,k,nsol=0;
  static int[] x, y, d;
  static int[] a=new int[5];
  static int[][] ePunctIn=new int[101][101];  // =i ==> e punctul i
                                              // =0 ==> nu e punct acolo
  public static void main(String[] args) throws IOException
  {
    int i,j,ymax;
    long t1,t2;
    t1=System.nanoTime();
    PrintWriter out=new PrintWriter(
                    new BufferedWriter(new FileWriter("poligon.out")));
    StreamTokenizer st=new StreamTokenizer(
                       new BufferedReader(new FileReader("poligon.in")));
    st.nextToken(); m=(int)st.nval;
```

```
    st.nextToken(); n=(int)st.nval;
    st.nextToken(); k=(int)st.nval;

    x=new int[k+1];
    y=new int[k+1];
    d=new int[k+1];

    for(i=1;i<=k;i++)
    {
      st.nextToken(); y[i]=(int)st.nval;      // linia
      st.nextToken(); x[i]=(int)st.nval;      // coloana
      st.nextToken(); d[i]=(int)st.nval;      // directia
    }

    ymax=y[1];
    for(i=2;i<=k;i++) if(y[i]>ymax) ymax=y[i];
    for(i=1;i<=k;i++) y[i]=ymax-y[i]+1;       // sa fie "normal"!

    for(i=1;i<=k;i++) ePunctIn[y[i]][x[i]]=i;

    generezCombinariPePozitia(1);

    out.println(nsol);
    out.close();
    t2=System.nanoTime();
    System.out.println("Timp = "+((double)(t2-t1))/1000000000);
  }

  static void generezCombinariPePozitia(int j)
  {
    int i;
    for(i=a[j-1]+1;i<=k-4+j;i++)
    {
      a[j]=i;
      if(j<4) generezCombinariPePozitia(j+1);
      else if(ePoligonOK()) nsol++;
    }
  }

  static boolean ePoligonOK()
  {
    if(coliniare3Puncte()) return false;
    if(ePoligon(a[1],a[2],a[3],a[4])) return true;
    if(ePoligon(a[1],a[2],a[4],a[3])) return true;
```

```
  if(ePoligon(a[1],a[3],a[2],a[4])) return true;
  if(ePoligon(a[1],a[3],a[4],a[2])) return true;
  if(ePoligon(a[1],a[4],a[2],a[3])) return true;
  if(ePoligon(a[1],a[4],a[3],a[2])) return true;
  return false;
}

static boolean coliniare3Puncte()
{
  if(coliniare(a[1],a[2],a[3])) return true;
  if(coliniare(a[1],a[2],a[4])) return true;
  if(coliniare(a[1],a[3],a[4])) return true;
  if(coliniare(a[2],a[3],a[4])) return true;
  return false;
}

static boolean coliniare(int p1, int p2, int p3)
{
  int s;
  s=x[p1]*y[p2]+x[p2]*y[p3]+x[p3]*y[p1];
  s=s-y[p1]*x[p2]-y[p2]*x[p3]-y[p3]*x[p1];
  if(s==0) return true; else return false;
}

static boolean ePoligon(int p1, int p2, int p3, int p4)
{
  if(!eLinie(p1,p2)) return false;
  if(!eLinie(p2,p3)) return false;
  if(!eLinie(p3,p4)) return false;
  if(!eLinie(p4,p1)) return false;
  if(eNedegenerat(p1,p2,p3,p4)) return true; else return false;
}

static boolean eLinie(int p1, int p2) // trece prin coordonate intregi!
{
  if(Math.abs(x[p1]-x[p2])==Math.abs(y[p1]-y[p2])) return eLinieOkOblica(p1,p2);
  else if(x[p1]==x[p2]) return eLinieOkVerticala(p1,p2);
  else if(y[p1]==y[p2]) return eLinieOkOrizontala(p1,p2);
  else return false;
}

static boolean eLinieOkOrizontala(int p1, int p2)
{
  int i;
```

```
  if(x[p1]>x[p2]) {i=p1;p1=p2;p2=i;} // p1 ... p2

  for(i=x[p1]+1; i<=x[p2]; i++)
  {
    if(ePunctIn[y[p1]][i]==0) return false;
    else if((d[ePunctIn[y[p1]][i]]&(1<<1))==0) // linie spre V
         return false;
  }
  return true;
}

static boolean eLinieOkVerticala(int p1, int p2)
{
  int i;
  if(y[p1]>y[p2]) {i=p1;p1=p2;p2=i;} // p1 ... p2

  for(i=y[p1]+1; i<=y[p2]; i++)
    if(ePunctIn[i][x[p1]]==0) return false;
    else if((d[ePunctIn[i][x[p1]]]&(1<<3))==0) // linie spre S
         return false;
  return true;
}

static boolean eLinieOkOblica(int p1, int p2)
{
  int i,j,pasy,dir;
  if(x[p1]>x[p2]) {i=p1;p1=p2;p2=i;} // p1 ... p2

  if(y[p1]>y[p2]) {pasy=-1; dir=0;} // NV
  else {pasy=+1; dir=2;} // SV

  i=y[p1];
  for(j=x[p1]+1; j<=x[p2]; j++)
  {
    i=i+pasy;
    if(ePunctIn[i][j]==0) return false;
    else if((d[ePunctIn[i][j]]&(1<<dir))==0) // linie spre SV sau NV
         return false;
  }
  return true;
}

static boolean eNedegenerat(int p1, int p2, int p3, int p4)
{
```

```
    // daca nu se intersecteaza p1p4 cu p2p3
    if(!seIntersecteaza(x[p1],y[p1],x[p4],y[p4],x[p2],y[p2],x[p3],y[p3]))
    return true; else return false;
  }

  static boolean seIntersecteaza( int x1,int y1,int x2,int y2,
  int x3,int y3,int x4,int y4)
  {
    // daca se intersecteaza segmentele [p1p2] cu [p3p4]
    // p3 si p4 sunt in semiplane diferite fata de dreapta (p1p2) si
    // p1 si p2 sunt in semiplane diferite fata de dreapta (p3p4)
    return (s(x3,y3,x1,y1,x2,y2)*s(x4,y4,x1,y1,x2,y2)<0) &&
           (s(x1,y1,x3,y3,x4,y4)*s(x2,y2,x3,y3,x4,y4)<0);
  }

  static int s(int xp,int yp,int xa,int ya,int xb,int yb)
  {
    //de ce parte a dreptei ((xa,ya);(xb,yb)]) se afla (xp,yp)
    double s=(double)yp*(xb-xa)-xp*(yb-ya)+xa*yb-xb*ya;
    if(s<-0.001) return -1;
    else if(s>0.001) return 1;
         else return 0;
  }
}
```

11.4 Şablon

Gigel şi Vasilică imaginează un mod de a transmite mesaje pe care nimeni să nu le poată descifra. Mesajul este ascuns într-un text care are N linii şi pe fiecare linie sunt exact N caractere - litere mari ale alfabetului englez, cifre, semne de punctuaţie şi caracterul spaţiu.

Decodificarea se face cu ajutorul unui şablon, de aceleaşi dimensiuni ca şi textul, care are câteva găuri.

Suprapunând şablonul peste text rămân vizibile câteva caractere.

Acestea se citesc în ordinea liniilor, de sus în jos, iar pe aceeaşi linie de la stânga la dreapta.

Apoi hârtia cu textul se roteşte spre stânga, în sens trigonometric, cu 90^o, şablonul rămânând fix. Alte caractere devin vizibile şi acestea se citesc în acelaşi mod.

Operaţia se repetă de încă două ori (rotire cu 180^o , respectiv cu 270^o), până când textul ajunge, printr-o nouă rotaţie cu 90^o, din nou în poziţia iniţială.

Din păcate, şablonul pentru codificare/decodificare s-a pierdut. În schimb a rămas la Gigel mesajul iniţial iar la Vasilică a ajuns textul care conţine mesajul.

Cerinţă

Să se reconstituie şablonul care a fost folosit la codificare.

Date de intrare

Fişierul de intrare **sablon.in** conţine

- pe prima linie, mesajul iniţial
- pe linia a doua a fişierului de intrare se găseşte valoarea N
- următoarele N linii conţin textul care ascunde mesajul.

Date de ieşire

Fişierul de ieşire **sablon.out** conţine N linii a câte N caractere. Caracterele sunt 'O' (pentru reprezentarea unei găuri) şi 'X'.

Restricţii şi

- prin rotirea textului nici una din găuri nu se va suprapune peste nici una din poziţiile ocupate de o gaură în poziţiile precedente ale textului
- $1 \leq N \leq 50$
- mesajul are maxim 1000 caractere şi se încheie cu un caracter diferit de spaţiu
- în cazul în care există mai multe soluţii, afişaţi una dintre ele

Exemplu

sablon.in	**sablon.out**
CODIFICARE CU SABLON	XXXX**O**XXXXX
10	XX**O**XXXXXXX
ABCDCEFAGH	**O**XXXXXXXXX
IJOKLEMNOP	XXX**O**XXXXXX
DQRSTUVWCX	X**O**XXXXXXXX
YZAIBCRDEF	XXXXXXXXXX
GFHIJKLMNI	XXXXXXXXXX
AJKLMNOPSQ	XXXXXXXXXX
RSTOUV WXY	XXXXXXXXXX
ZBABCDEFGU	XXXXXXXXXX
HIJKNLMCNO	
PQLRS TUVW	

Timp de execuţie: 1 sec/test

11.4.1 Indicaţii de rezolvare - descriere soluţie *

Soluţia oficială

Problema se rezolvă relativ simplu ţinând cont de următoarele observaţii:

1. Deoarece întregul mesaj a fost codificat prin 4 rotiri ale textului, este clar că la o poziţionare a textului sub şablon pot fi citite $Lung(Mesaj)/4$ caractere, deci întregul mesaj are $4 * NumarGauri$ caractere

2. Ca urmare a observaţiei de la punctul 1, mesajul poate fi împartit exact în 4 şiruri de lungimi egale $Mesaj1$, ..., $Mesaj4$

3. Dacă o gaură se află în poziţia $T[i, j]$ din şablon, ei îi corespund poziţiile

- $T[j, N - i + 1]$ la rotire cu 90 grade
- $T[N - i + 1, N - j + 1]$ la rotire cu 180 grade
- $T[N - j + 1, i]$ la rotire cu 270 grade

de unde deducem că nu e nevoie să rotim textul!!!

4. Dacă lungimea unui şir este $L4$ (vezi în sursă), este suficient să parcurgem numai primul din cele 4 şiruri cu un $Index$. Atunci, parcurgând textul care ascunde mesajul, în poziţia (i, j) există o gaură în şablon dacă şi numai dacă toate cele 4 caractere

$$Mesaj1[Index], Mesaj2[Index], Mesaj3[Index], Mesaj4[Index]$$

coincid cu cele 4 caractere obţinute prin rotire (vezi observaţia 3)

5. "Cel mai bun pseudocod este... PASCAL-ul", deci: ... (urmează sursa în Pascal).

11.4.2 Rezolvare detaliată *

Verificarea preluării corecte a datelor de intrare:

```
import java.io.*;
class Sablon0
{
  static int n;
  static String mesaj; // mesajul initial
  static char[][] text; // text (codificarea)
  static PrintWriter out;

  public static void main(String[] args) throws IOException
  {
    int i,j;
    out=new PrintWriter(new BufferedWriter(new FileWriter("sablon.out")));
    BufferedReader br=new BufferedReader(new FileReader("sablon.in"));
    StreamTokenizer st=new StreamTokenizer(br);

    mesaj=br.readLine(); System.out.println(mesaj);
    st.nextToken(); n=(int)st.nval; System.out.println(n);
```

```
    text=new char[n][n];
    br.readLine(); // citeste CR LF adica 0D 0A adica 13 10
    for(i=0;i<n;i++) text[i]=br.readLine().toCharArray();
    for(i=0;i<n;i++)
    {
      for(j=0;j<n;j++) System.out.print(text[i][j]);
      System.out.println();
    }
    out.close();
  }
}
```

11.4.3 Codul sursă *

```
import java.io.*;
class Sablon1
{
  static int n;
  static String mesajInitial; // mesajul initial
  static char[][] text; // text (codificarea)
  static boolean[][] gaura;

  static PrintWriter out;
  static BufferedReader br;
  static StreamTokenizer st;

  public static void main(String[] args) throws IOException
  {
    citire();
    rezolvare();
    afisare();
  }

  static void citire() throws IOException
  {
    int i,j;
    br=new BufferedReader(new FileReader("sablon.in"));
    st=new StreamTokenizer(br);
    mesajInitial=br.readLine();
    st.nextToken(); n=(int)st.nval;
    text=new char[n][n];
    gaura=new boolean[n][n];
```

```
    br.readLine(); // citeste CR LF adica 0D 0A adica 13 10
    for(i=0;i<n;i++) text[i]=br.readLine().toCharArray();
  }

  static void rezolvare()
  {
    int i,j,k;
    int nrGauri=mesajInitial.length()/4;
    char[][] mesajPartial=new char[4][nrGauri]; // 4 mesaje partiale
    for(i=0;i<nrGauri;i++) // impart mesajul in 4 parti egale
    {
      mesajPartial[0][i]=mesajInitial.charAt(i);
      mesajPartial[1][i]=mesajInitial.charAt(nrGauri+i);
      mesajPartial[2][i]=mesajInitial.charAt(2*nrGauri+i);
      mesajPartial[3][i]=mesajInitial.charAt(3*nrGauri+i);
    }

    k=0; // gaurile 0, 1, ..., nrGauri-1
    for(i=0;i<n;i++)
      for(j=0;j<n;j++)
        if( (mesajPartial[0][k]==text[i][j]) &&  // primul caracter
            (mesajPartial[1][k]==text[j][n-i-1]) &&  // rotit cu 90 grade
            (mesajPartial[2][k]==text[n-i-1][n-j-1]) && // rotit cu 180 grade
            (mesajPartial[3][k]==text[n-j-1][i]))  // rotit cu 270 grade
        { // daca toate 4 caractere coincid
          gaura[i][j]=true;
          k++;
          if(k>=nrGauri) return;
        }
  }

  static void afisare() throws IOException
  {
    int i,j;
    out=new PrintWriter(new BufferedWriter(new FileWriter("sablon.out")));
    for(i=0;i<n;i++)
    {
      for(j=0;j<n;j++)
        if(gaura[i][j]) out.print('O'); else out.print('X');
      out.println();
    }
    out.close();
  }
}
```

11.5 Şir

Gigel se distrează construind şiruri crescătoare de numere din mulţimea $\{1, 2, ..., n\}$. La un moment dat observă că unele şiruri, de cel puţin k termeni ($k \geq 3$), au o proprietate mai aparte: diferenţa dintre doi termeni consecutivi este constantă. Iată câteva exemple de astfel de şiruri pentru $n \geq 21$:

2,3,4
1,5,9,13
7,10,13,16,19,21

Cerinţă

Dându-se numărul natural n ajutaţi-l pe Gigel să numere câte astfel de şiruri poate să construiască.

Date de intrare

În fişierul de intrare **sir.in** se găseşte, pe prima linie, numărul n.

Date de ieşire

În fişierul de ieşire **sir.out** se va afişa, pe prima linie, numărul cerut urmat de caracterul sfârşit de linie.

Restricţii:

- $3 \leq n \leq 20000$
- $3 \leq k \leq n$

Exemple:

sir.in	sir.out
3	1
4	3
5	7

Timp execuţie: 1 sec/test

11.5.1 Indicaţii de rezolvare - descriere soluţie *

Soluţia oficială

Notând cu r diferenta dintre doi tereni consecutivi constatăm că pentru $r = 1$ se pot construi urmatoarele submulţimi, şiruri cu proprietea cerută, de lungime 3:

$$\{1, 2, 3\}, \{2, 3, 4\}, ..., \{n-2, n-1, n\}.$$

Cele de lungime superioară se construiesc adăugând elemente pe cele deja obţinute. Numărul lor va fi $\sum_{i=1}^{n-2} i$.

Similar pentru $r = 2$ obţinem următoarele submulţimi, şiruri de lungime 3:

$\{1, 3, 5\}, \{2, 4, 6\}, ..., \{n-4, n-2, n\}$ sau $\{n-5, n-3, n-1\}$ functie de paritatea lui n .

Cele de lungime superioară se construiesc adăugând elemente pe acestea. Numarul lor este o sumă te tipul precedent.

Se continuă astfel până la $r = n/2$, valoarea maximă a lui r.

11.5.2 Rezolvare detaliată

11.5.3 Codul sursă *

```
import  java.io.*;
class sir
{
  public static void main(String []args) throws IOException
  {
    int ns=0,n=19999,r,k,i;
    StreamTokenizer st=new StreamTokenizer(
          new BufferedReader(new FileReader("sir.in")));
    PrintWriter out=new PrintWriter(new BufferedWriter(
         new FileWriter("sir.out")));
    st.nextToken(); n=(int)st.nval;
    ns=0;
    for(r=1;r<=(n-1)/2;r++)
      for(k=3;k<=(n-1+r)/r;k++)
        ns=ns+n-(k-1)*r;
    System.out.println(ns);
    out.println(ns);
    out.close();
  }
}
```

11.6 Snipers

Se spune că în timpul războiului cu gnomii, trolii au trimis n trăgători de elită să lichideze cele n căpetenii inamice.

Din fericire căpeteniile inamice erau plasate în câmp deschis, iar trăgătorii au reuşit să se plaseze în zonă fără să fie observaţi.

Când să fie dată comanda de tragere s-a constatat că nu se transmisese fiecărui trăgător ce căpetenie să împuşte, iar dacă doi trăgători ar fi tras în aceeaşi căpetenie sau traiectoriile razelor ucigaşe s-ar fi intersectat, atunci ar fi scăpat cel

puţin o căpetenie care ar fi putut duce războiul până la capăt, iar trolii ar fi fost învinşi.

Deoarece căpeteniile aveau capacitatea de a deveni invizibile oricând doreau (pe o perioadă nelimitată), trebuiau lichidate simultan, altfel ...

Istoria ne spune că trolii au învins deoarece comandantul lor a reuşi ca în mai puţin de o secundă să transmită fiecărui trăgător în ce căpetenie să tragă.

Voi puteţi face asta?

Cerinţă

Scrieţi un program care, citind poziţiile trăgătorilor şi a căpeteniilor, determină căpetenia în care trebuie să tragă fiecare trăgător.

Date de intrare

Fişierul de intrare **snipers.in** conţine

• pe prima sa linie numărul n

• pe următoarele n linii se află perechi de numere întregi, separate prin spaţiu, ce reprezintă coordonatele trăgătorilor urmate de

• alte n perechi de numere întregi ce reprezintă coordonatele căpeteniilor (abscisă şi ordonată).

Date de ieşire

Fişierul de ieşire **snipers.out** conţine n linii.
Pe linia i a fişierului se află numărul căpeteniei ţintite de trăgătorul i ($i = 1...n$).

Restricţii şi precizări

• $0 < n < 200$

• Coordonatele sunt numere întregi din intervalul [0, 50000]

• Raza ucigaşă a oricărei arme se opreşte în ţinta sa.

• În datele de intrare nu vor exista trei persoane aflate în puncte coliniare.

Exemple

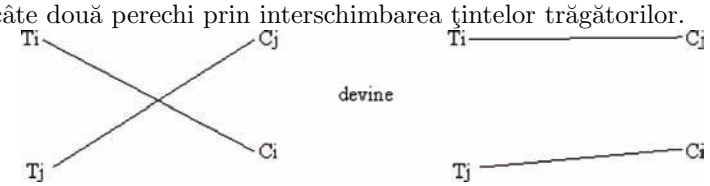
snipers.in	**snipers.out**	**snipers.in**	**snipers.out**
2	1	5	2
1 3	2	6 6	5
1 1		4 12	1
3 4		2 8	3
3 1		9 4	4
		5 2	
		6 11	
		9 7	
		3 9	
		1 4	
		7 3	

Timp de execuţie: 1 sec/test

11.6.1 Indicaţii de rezolvare - descriere soluţie *

Soluţia oficială

La început fiecărui tragator i îi asociem capetenia i, după care vom lua în considerare toate perechile trăgător-căpetenie şi vom elimina încrucişările razelor, la câte două perechi prin interschimbarea ţintelor trăgătorilor.



Se vor face interschimbări între câte două perechi trăgător-ţintă pâna când nu vor mai exista intersecţii.

Se observă că la eliminarea unei intersecţii suma distanţelor dintre trăgători şi ţinte scade, ceea ce asigură finitudinea algoritmului.

11.6.2 Rezolvare detaliată

Atenţie: valoarea expresiei `yp*(xb-xa)-xp*(yb-ya)+xa*yb-xb*ya` poate depăşi capacitatea de înregistrare a tipului `int`.

11.6.3 Codul sursă *

```
import java.io.*;
class Snipers
{
  static int n;
  static int[] xt, yt, xc, yc, t, c;

  static PrintWriter out;
  static StreamTokenizer st;

  public static void main(String[] args) throws IOException
  {
    citire();
    rezolvare();
    afisare();
  }

  static void citire() throws IOException
  {
    int k;
```

```
    st=new StreamTokenizer(new BufferedReader(new FileReader("snipers.in")));
    st.nextToken(); n=(int)st.nval;
    xt=new int[n+1];
    yt=new int[n+1];
    xc=new int[n+1];
    yc=new int[n+1];
    t=new int[n+1]; // tragator --> capetenie

    for(k=1;k<=n;k++)
    {
      st.nextToken(); xt[k]=(int)st.nval;
      st.nextToken(); yt[k]=(int)st.nval;
    }
    for(k=1;k<=n;k++)
    {
      st.nextToken(); xc[k]=(int)st.nval;
      st.nextToken(); yc[k]=(int)st.nval;
    }
  }

  static void rezolvare()  throws IOException
  {
    int i,j,aux;
    boolean seIntersecteaza_ij,ok;
    for(i=1;i<=n;i++) t[i]=i; // tragatorul i trage in capetenia i
    if(n==1) return;
    do
    {
      ok=true; // am gasit o combinatie valida tragator-capetenie
      i=1;
      do
      {
        j=i;
        do
        {
          j=j+1;
          seIntersecteaza_ij=seIntersecteaza(xt[i],yt[i],xc[t[i]],yc[t[i]],
                                             xt[j],yt[j],xc[t[j]],yc[t[j]]);
          if(seIntersecteaza_ij)
          {
            aux=t[i];t[i]=t[j];t[j]=aux;  // interschimbam tintele
            ok=false;                     // nu avem combinatie buna
          }
        } while ((j!=n) && !seIntersecteaza_ij);
```

```
        if(!seIntersecteaza_ij) i++; // trecem la urmatorul sniper
      } while(i!=n); // pana terminam lista de tragatori
    } while(!ok);
  }

  //de ce parte a dreptei [(xa,ya);(xb,yb)] se afla (xp,yp)
  static int s(int xp,int yp,int xa,int ya,int xb,int yb)
  {
    double s=(double)yp*(xb-xa)-xp*(yb-ya)+xa*yb-xb*ya;
    if(s<-0.001) return -1;
      else if(s>0.001) return 1;
             else return 0;
  }

  // testeaza daca segmentul[a1,b1] se intersecteaza cu [a2,b2]
  static boolean seIntersecteaza(int xa1,int ya1,int xb1,int yb1,
                                 int xa2,int ya2,int xb2,int yb2)
  {
    // a2 si b2 se afla de o parte si de alta a lui [a1,b1] si
    // a1 si b1 se afla de o parte si de alta a lui [a2,b2]
    return (s(xa2,ya2,xa1,ya1,xb1,yb1)*s(xb2,yb2,xa1,ya1,xb1,yb1)<0) &&
           (s(xa1,ya1,xa2,ya2,xb2,yb2)*s(xb1,yb1,xa2,ya2,xb2,yb2)<0);
  }

  static void afisare() throws IOException
  {
    int k;
    out=new PrintWriter(new BufferedWriter(new FileWriter("snipers.out")));
    for(k=1;k<=n;k++) out.println(t[k]);
    out.close();
  }
}
```

Capitolul 12

ONI 2005 clasa a IX-a

12.1 Bifo

Autor: Silviu Gănceanu

Pentru a-şi vindeca rana provocată de Spânul cel Negru, prinţul Algorel are nevoie de leacul miraculos aflat în posesia vrăjitoarei din pădurea întunecată.

Aceasta i-a promis leacul dacă îi rezolvă următoarea problemă, la care ea s-a gândit zadarnic o mie de ani: pornind de la două cuvinte iniţiale A_1 şi A_2 şi aplicând "formula bifo" $A_n = A_{n-2}A_{n-1}$ pentru $n \geq 3$, se obţin cuvintele A_3, A_4, A_5, ş.a.m.d.

Prin $A_{n-2}A_{n-1}$ înţelegem concatenarea cuvintelor A_{n-2} şi A_{n-1} în această ordine.

Toate aceste cuvinte (A_1, A_2, A_3, ş.a.m.d), sunt la rândul lor concatenate, în ordine, formând un şir de caractere infinit denumit *şir magic*.

Formula leacului miraculos are M caractere, pe care vrăjitoarea nu le ştie. Se ştiu însă cele M poziţii din şirul magic în care apar, în ordine, caracterele din formulă.

Cerinţă

Cu toată inteligenţa lui, Algorel nu poate rezolva această problemă. Ajutaţi-l pe prinţ să iasă din încurcătură aflând formula leacului magic.

Date de intrare

Primele două linii ale fişierului **bifo.in** conţin fiecare câte un şir de cel mult 100 de caractere reprezentând cuvintele A_1 (pe prima linie) şi respectiv A_2 (pe a doua linie).

A treia linie conţine un număr întreg M, reprezentând numărul de caractere din formula leacului miraculos.

Urmează M linii descriind, în ordine, poziţiile din şirul magic unde se găsesc caracterele din formulă.

Date de ieşire

Fişierul de ieşire **bifo.out** va conţine pe prima linie un şir de M caractere reprezentând formula leacului miraculos.

Restricţii şi precizări

- $1 \leq M \leq 100$;
- A_1 şi A_2 conţin doar litere mici ale alfabetului englez;
- Numerotarea poziţiilor din şirul infinit începe cu 1;
- Cele M poziţii vor fi numere întregi (nu neapărat distincte) de maxim 100 de cifre;
- Pentru 60% din teste poziţiile vor fi numere întregi între 1 şi 1.000.000.000;
- Fiecare linie din fişierul de intrare şi din fişierul de ieşire se termină cu marcaj de sfârşit de linie;

Exemplu

bifo.in	bifo.out
ab cdx 3 10 4 15	xdb

Explicaţie: Primele 5 şiruri obţinute folosind "formula bifo" sunt:

ab, cdx, abcdx, cdxabcdx, abcdxcdxabcdx

Concatenând aceste şiruri se obţine şirul magic:

abcdxabcdxcdxabcdxabcdxcdxabcdx...

Timp maxim de execuţie/test: 1 sec sub Windows şi 1 sec sub Linux

12.1.1 Indicaţii de rezolvare - descriere soluţie *

Soluţia oficială, Silviu Gănceanu

Utilizând "formula bifo" se obţine un şir de cuvinte $A1$, $A2$, $A3$, ş.a.m.d, care este practic un şir Fibonacci de cuvinte. Fie L_i lungimea cuvântului A_i din acest şir. Pentru a afla ce caracter se află pe poziţia X din şirul magic se procedează în modul următor :

$pas1$: se găseşte cel mai mic K astfel încât $L_1 + L_2 + ... + L_K \geq X$ (termenul din şirul de cuvinte unde se găseşte poziţia X)

$pas2$: se scade din valoarea X suma $L_1 + L_2 + ... + L_{K-1}$ pentru a afla poziţia din termenul K unde este caracterul din poziţia X

$pas3$:

1. dacă $K < 3$ se afişează caracterul de pe poziţia X din cuvântul corespunzător
2. altfel, ştiind lungimile cuvintelor A_{K-2} şi A_{K-1} şi ştiind că A_K se obţine prin concatenarea acestora, se decide în care din aceste cuvinte se va afla caracterul căutat în modul următor:
   - dacă $L_{K-2} < X$ atunci caracterul va fi în cuvântul A_{K-1} şi vom scădea K cu o unitate, iar X cu L_{K-2} şi revenim la $pas3$
   - altfel caracterul căutat se află în cuvântul A_{K-2} şi vom scădea din K două unităţi iar X va rămâne neschimbat şi se revine la $pas3$

Precizăm că termenii şirului (Fibonacci) de cuvinte nu se obţin explicit ci doar se lucrează cu lungimile acestora. Soluţia presupune utilizarea numerelor mari din moment ce poziţiile din şirul magic pot depăşi orice tip de date predefinit în compilatoarele utilizate.

Datele de test au fost create astfel încât:

- Implementarea unei soluţii imediate, care calculează termenii şirului (Fibonacci) necesari aflării caracterelor de pe poziţiile din fişierul de intrare, să fie punctată cu aproximativ 30 pct
- Implementarea soluţiei corecte fără folosirea numerelor mari să obţină aproximativ 60 pct
- Implementarea soluţiei corecte folosind numere mari (implementate de concurent) să obţină 100 pct

12.1.2 Rezolvare detaliată

Testarea preluării datelor de intrare.

```
import java.io.*;
class Bifo1
```

```
{
  static int m;
  static char[] a1;
  static char[] a2;
  static char[] f;// formula
  static int[] p; // pozitii

  static PrintWriter out;
  static BufferedReader br;
  static StreamTokenizer st;

  public static void main(String[] args) throws IOException
  {
    citire();
    rezolvare();
    afisare();
  }

  static void citire() throws IOException
  {
    int i,j;
    br=new BufferedReader(new FileReader("bifo.in"));
    st=new StreamTokenizer(br);
    st.nextToken(); a1=st.sval.toCharArray();
    st.nextToken(); a2=st.sval.toCharArray();
    st.nextToken(); m=(int)st.nval;
    f=new char[m+1];
    p=new int[m+1];
    for(i=1;i<=m;i++)
    {
      st.nextToken();
      p[i]=(int)st.nval;
    }

    afisv(a1);
    afisv(a2);
    System.out.println(m);
    afisv(p);
  }

  static void afisv(char[] a)
  {
    int i;
    for(i=0;i<a.length;i++) System.out.print(a[i]);
```

```
    System.out.println(" "+a.length);
  }

  static void afisv(int[] a)
  {
    int i;
    for(i=1;i<=m;i++) System.out.println(a[i]);
    System.out.println();
  }

  static void rezolvare()
  {

  }

  static void afisare() throws IOException
  {
    int i;
    out=new PrintWriter(new BufferedWriter(new FileWriter("bifo.out")));
    for(i=1;i<=m;i++) out.print(f[i]);
    out.println();
    out.close();
  }
}
```

12.1.3 Codul sursă *

Variantă fără numere mari.

```
import java.io.*;
class Bifo2
{
  static int m;
  static char[] a1;
  static char[] a2;
  static char[] f;// formula
  static int[] p; // pozitii
  static int[] lg;// lungimile cuvintelor a[i]=a[i-2]a[i-1]

  static PrintWriter out;
  static BufferedReader br;
  static StreamTokenizer st;
```

```
public static void main(String[] args) throws IOException
{
  int i;
  citire();
  for(i=1;i<=m;i++) rezolvare(i);
  afisare();
}

static void citire() throws IOException
{
  int i,j;
  br=new BufferedReader(new FileReader("bifo.in"));
  st=new StreamTokenizer(br);
  st.nextToken(); a1=st.sval.toCharArray();
  st.nextToken(); a2=st.sval.toCharArray();
  st.nextToken(); m=(int)st.nval;
  f=new char[m+1];
  p=new int[m+1];
  lg=new int[101];
  for(i=1;i<=m;i++)
  {
    st.nextToken();
    p[i]=(int)st.nval;
  }
}

static void rezolvare(int i)
{
  int x,k,sk1,sk,pk;
  lg[1]=a1.length;
  lg[2]=a2.length;

  x=p[i];
  if(x<=lg[1]){ f[i]=a1[x-1]; return; }
  if(x<=lg[1]+lg[2]){ f[i]=a2[x-lg[1]-1]; return; }

  sk1=lg[1];
  sk=sk1+lg[2];
  k=2; // k=cuvantul unde se gaseste caracterul de pe pozitia x
  while(sk<x)
  {
    k++;
    lg[k]=lg[k-2]+lg[k-1];
    sk1=sk;
```

```
      sk=sk1+lg[k];
    }
    x=x-sk1;

    while(k>2)        // a[k]=a[k-2]a[k-1]
    {
      if(lg[k-2]>=x)  // caracterul este in a[k-2] stanga
      {
        k=k-2;
      }
      else            // caracterul este in a[k-1] dreapta
      {
        x=x-lg[k-2];
        k--;
      }
    }
    if(k==1) f[i]=a1[x-1];
    if(k==2) f[i]=a2[x-1];
  }

  static void afisare() throws IOException
  {
    int i;
    out=new PrintWriter(new BufferedWriter(new FileWriter("bifo.out")));
    for(i=1;i<=m;i++) out.print(f[i]);
    out.println();
    out.close();
  }
}
```

Variantă cu numere mari.

```
import java.io.*; // Fibo[481]=101 cifre
class Bifo3
{
  static int m;
  static char[] a1;
  static char[] a2;
  static char[] f; // formula
  static char[][] p; // pozitii
  static char[][] lg=new char[482][1];// lungimile cuvintelor a[i]=a[i-2]a[i-1]

  static PrintWriter out;
  static BufferedReader br;
  static StreamTokenizer st;
```

```
public static void main(String[] args) throws IOException
{
  int i;
  citire();
  for(i=1;i<=m;i++) rezolvare(i);
  afisare();
}

static void citire() throws IOException
{
  int i,j;
  char aux;
  br=new BufferedReader(new FileReader("bifo.in"));
  st=new StreamTokenizer(br);
  st.nextToken(); a1=st.sval.toCharArray();
  st.nextToken(); a2=st.sval.toCharArray();
  st.nextToken(); m=(int)st.nval;
  f=new char[m+1];
  p=new char[m+1][1];

  br.readLine(); // citeste CR LF adica 0D 0A adica 13 10
  for(i=1;i<=m;i++)
  {
    p[i]=br.readLine().toCharArray();
    for(j=0;j<p[i].length/2;j++)
    {
      aux=p[i][j];
      p[i][j]=p[i][p[i].length-j-1];
      p[i][p[i].length-j-1]=aux;
    }
    for(j=0;j<p[i].length;j++)
    p[i][j]=(char)(p[i][j]-0x30); // coduri cifre --> cifre reale
  }
}

static void rezolvare(int i)
{
  char[] x,sk1,sk,pk;
  int k;
  lg[1]=nr2v(a1.length);
  lg[2]=nr2v(a2.length);

  x=suma(nr2v(0),p[i]); // improvizatie pentru x=p[i];   !!!
```

```
    if(compar(x,lg[1])<=0){f[i]=a1[v2nr(x)-1];return;}
    if(compar(x,suma(lg[1],lg[2]))<=0){f[i]=a2[v2nr(x)-v2nr(lg[1])-1];return;}

    sk1=suma(nr2v(0),lg[1]);
    sk=suma(sk1,lg[2]);
    k=2; // k=cuvantul unde se gaseste caracterul de pe pozitia x
    while(compar(sk,x)<0)
    {
      k++;
      lg[k]=suma(lg[k-2],lg[k-1]);
      sk1=suma(nr2v(0),sk);
      sk=suma(sk1,lg[k]);
    }
    x=scade(x,sk1);
    while(k>2) // a[k]=a[k-2]a[k-1]
    {
      if(compar(lg[k-2],x)>=0) // caracterul este in a[k-2] stanga
      {
        k=k-2;
      }
      else // caracterul este in a[k-1] dreapta
      {
        x=scade(x,lg[k-2]);
        k--;
      }
    }
    if(k==1) f[i]=a1[v2nr(x)-1];
    if(k==2) f[i]=a2[v2nr(x)-1];
  }

  static void afisare() throws IOException
  {
    int i;
    out=new PrintWriter(new BufferedWriter(new FileWriter("bifo.out")));
    for(i=1;i<=m;i++) out.print(f[i]);
    out.println();
    out.close();
  }

  static int compar(char[] a, char[] b) //-1, 0, 1 ... a < = > b
  {
    int na=a.length;
    int nb=b.length;
    if(na>nb) return 1; else if(na<nb) return -1;
```

```
    int i=na-1;
    while((i>=0)&&(a[i]==b[i])) i--;
    if(i==-1) return 0;
    else if(a[i]>b[i]) return 1; else return -1;
  }

  static char[] scade(char[] x,char[] y) // z=x-y unde x>=y
  {
    int nx=x.length;
    int ny=y.length;
    int nz=nx;
    int i,s;
    char t;
    char[] z=new char[nz];
    char[] yy=new char[nz];
    for(i=0;i<ny;i++) yy[i]=y[i];
    t=0;
    for(i=0;i<nz;i++)
    {
      s=x[i]-yy[i]-t; // poate fi negativ ==> nu merge tipul char !!!
      if(s<0) {z[i]=(char)(s+10); t=1;} else {z[i]=(char)s; t=0;}
    }
    if(z[nz-1]!=0) return z;
    else
    {
      int poz=nz-1; // de exemplu 123-121=002 ==>
      while((int)z[poz]==0) poz--;// pot fi mai multe zero-uri la inceput
      char[] zz=new char[poz+1];
      for(i=0;i<=poz;i++) zz[i]=z[i];
      return zz;
    }
  }

  static char[] suma(char[] x,char[] y)
  {
    int nx=x.length;
    int ny=y.length;
    int nz;
    if(nx>ny) nz=nx+1; else nz=ny+1;
    int t,i;
    char[] z=new char[nz];
    char[] xx=new char[nz];
    char[] yy=new char[nz];
```

```
    for(i=0;i<nx;i++) xx[i]=x[i];
    for(i=0;i<ny;i++) yy[i]=y[i];
    t=0;
    for(i=0;i<nz;i++)
    {
      z[i]=(char)(xx[i]+yy[i]+t);
      t=z[i]/10;
      z[i]=(char)(z[i]%10);
    }
    if(z[nz-1]!=0) return z;
    else
    {
      char[] zz=new char[nz-1];
      for(i=0;i<nz-1;i++) zz[i]=z[i];
      return zz;
    }
  }

  static char[] nr2v(int nr)
  {
    int nrr=nr,nc=0,i;
    while(nr!=0) { nc++; nr=nr/10; }
    char[] nrv=new char[nc];
    nr=nrr;
    for(i=0;i<nc;i++) { nrv[i]=(char)(nr%10); nr=nr/10; }
    return nrv;
  }

  static int v2nr(char[] x)
  {
    int nr=0,i;
    for(i=x.length-1;i>=0;i--) nr=nr*10+x[i];
    return nr;
  }
}
```

12.2 Romeo

Autor: prof. Dan Grigoriu

Oraşul Julietei este de formă pătrată şi are străzi doar pe direcţiile Nord-Sud şi Est-Vest, toate la distanţe egale şi numite ca în desen: strada verticală 0, 1, 2, 3, ..., respectiv strada orizontală 0, 1, 2, 3... . Julieta locuieşte la intersecţia străzilor: verticală x şi orizontalu a y (poziţia (x, y)).

Romeo se află în colţul de Sud-Vest al oraşului (poziţia (0,0)) şi doreşte să ajungă la Julieta, nu ştim exact de ce, dar este treaba lui. Peste toate necazurile cunoscute ale bietului băiat, mai apar şi altele:

– oraşul urcă în pantă spre Nord, ceea ce îngreunează mersul în acea direcţie;

– nu poate merge decât spre Nord sau spre Est, pentru că dacă "ea" l-ar vedea mergând spre Vest sau spre Sud, atunci ar crede că el se îndepărtează definitiv de ea.

Numim **segment elementar** distanţa dintre două străzi paralele alăturate.

Dacă Romeo merge spre Est, atunci el consumă $1J$ (J = joule = o unitate de energie) pentru fiecare segment elementar parcurs.

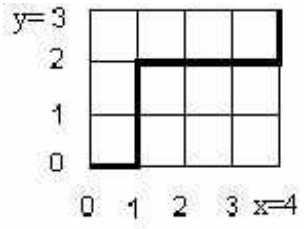
Din cauza pantei, dacă merge spre Nord k segmente elementare consecutive, consumă $(1 + 2 + 3 + ... + k)J$.

Romeo vrea să ajungă la Julieta (mergând **în condiţiile de mai sus**) cu un **consum minim de energie**.

De exemplu, dacă datele sunt:

$x = 4$ şi $y = 3$,

atunci desenul alăturat prezintă un drum posibil (dar **nu cu consum minim de energie**).



În desen, avem

• un prim segment elementar orizontal (consum = $1J$), apoi

• spre Nord două segmente elementare (consum: $1 + 2 = 3J$)

• urmează 3 segmente spre Est (consum: $1 + 1 + 1 = 3J$) şi

• ultima porţiune de un segment vertical (consum: $1J$).

Total consum energie: $1 + 3 + 3 + 1 = 8J$.

Cerinţă

Scrieţi un program care citeşte x şi y şi care afişează numărul minim de J consumaţi pentru tot drumul de la poziţia $(0, 0)$ la poziţia (x, y), mergând doar în direcţiile precizate.

Date de intrare

Fişierul de intrare **romeo.in** conţine numerele x şi y pe prima linie, separate de un spaţiu.

Date de ieşire

Fişierul de ieşire **romeo.out** conţine o singură linie cu numărul de J consumaţi pentru distanţa totală parcursă din poziţia de plecare până în cea finală.

Restricţii şi precizări

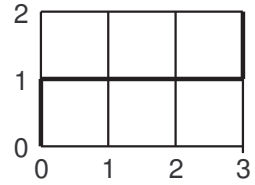
• x şi y sunt numere naturale;

• $0 \leq x, y \leq 40000$

• Fiecare linie din fişierul de intrare şi din fişierul de ieşire se încheie cu marcaj de sfârşit de linie.

Exemplu

romeo.in	romeo.out
3 2	5



Explicaţie

Datele de intrare indică un oraş ca în desen.

Un drum posibil (el nu este unic) este dat de linia îngroşată.

Primul segment vertical consumă 1J, porţiunea orizontală consumă 3J şi ultimul segment vertical (cel din dreapta), încă 1J, deci vom afişa numărul 5, care reprezintă 1J+3J+1J=5J.

Timp maxim de execuţie/test: 1 sec sub Windows şi 1 sec sub Linux

12.2.1 Indicaţii de rezolvare - descriere soluţie *

Soluţia oficială, profesor Dan Grigoriu

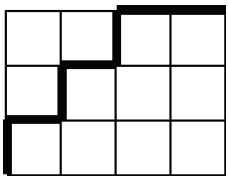
Soluţia descrisă este realizată didactic şi se bazează pe un calcul algebric simplu; soluţia nu conţine structuri repetive.

Energia pentru segmentele orizontale este în toate cazurile aceeaşi: X.

Problema rămâne pentru energia consumată cu segmentele verticale.

Analizăm două cazuri:

1) Dacă $X \geq Y - 1$, atunci se poate adopta un drum cu consum minim de energie mergând pe un zig-zag cu segmente elementare, începând cu direcţa spre Nord, până la strada orizontală a destinaţiei, după care (dacă este cazul), se merge pe acea stradă până la destinaţie, pe orizontală, ca în desenul alaturat.



Drumul prezentat nu este unic.

Bilanţul pentru un astfel de drum (în exemplu: $X = 4$ şi $Y = 3$) este: $(X + Y)J$.

2) Dacă $X < Y - 1$, atunci vom avea şi porţiuni verticale mai mari decât un segment elementar pe verticală. Cu cât o asemenea porţiune este mai lungă, cu atât energia consumată este mai mare. De aceea, vom încerca să avem porţiuni verticale de lungimi cât mai mici, chiar dacă sunt mai multe, lungimea lor totală fiind aceeaşi: Y.

Numărul de porţiuni verticale va fi $X+1$, adică vom merge vertical pe fiecare stradă verticală o porţiune, pentru a avea cât mai multe porţiuni verticale (deci şi mai mici).

Fie $Z = [Y/(X+1)]$ = lungimea unei porţiuni verticale (numărul de segmente elementare). Porţiunile verticale vor avea toate lungimea Z, dacă $(X + 1)|Y$, sau Z şi $Z + 1$, în caz contrar. Pentru acest caz, fie M = numărul de segmente de mărime Z şi N = numărul de segmente de mărime $Z + 1$.

Se va rezolva sistemul:

$$M + N = X + 1 \qquad (1)$$

$$M * Z + N * (Z + 1) = Y \qquad (2)$$

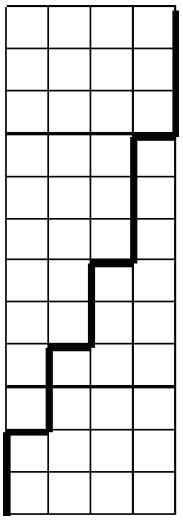
Semnificaţia ecuaţiilor:

(1): (numărul de porţiuni verticale de lungime Z) + (numărul de porţiuni verticale de lungime $Z + 1$) = (numărul total de porţiuni verticale, adica $X + 1$).

(2): (lungimea totală a porţiunilor de lungime Z) + (lungimea totală a poţiunilor de lungime $Z + 1$) = (distanţa totală de parcurs pe verticală, adică Y).

Odată obţinute M şi N, energia pe verticală se va calcula ca fiind suma dintre $E1$ = energia pentru cele M porţiuni de lungime Z şi $E2$ = energia pentru cele N porţiuni de lungime $Z + 1$.

$E1 + E2 = M * (1 + 2 + 3 + \ + Z) + N * (1 + 2 + 3 + \ + (Z + 1))$.



Mai jos, avem conform desenului:

$X = 4$ şi $Y = 12$.

Atunci: $Z = [Y/(X + 1)] = [12/(4 + 1)] = [12/5] = 2$

Sistemul devine:

(1) $M + N = 4 + 1$

(2) $M * 2 + N * (2 + 1) = 12$,

adică

$M + N = 5$

$M * 2 + N * (2 + 1) = 12$

cu soluţia:

$M = 3$ şi $N = 2$

Cu acestea, $E1$ şi $E2$ devin:

$E1 = M * (1 + + Z) = 3 * (1 + 2) = 9J$

$E2 = N * (1 + + Z + 1) = 2 * (1 + 2 + 3) = 12J$

Bilanţ:

Total energie pe verticală: $E1 + E2 = 9 + 12 = 21J$.

Total energie pe orizontală: $X = 4J$

Total energie pentru tot drumul: $21 + 4 = 25J$

Observaţie

Oraşul este de formă pătrată, dar în desen apare doar o parte din el (cea interesantă pentru noi).

12.2.2 Rezolvare detaliată *

Rezolvarea sistemului conduce la $n = y-(x+1)z$ şi $m = x+1-y+(x+1)z$ unde $z = \left[\frac{y}{x+1}\right]$ iar energia totală este

$$m \cdot \frac{z(z+1)}{2} + n \cdot \frac{(z+1)(z+2)}{2} + x.$$

12.2.3 Codul sursă *

```
import java.io.*;
class Romeo
{
  public static void main(String[] args) throws IOException
  {
    int x,y;
    int z,m,n,e;
    PrintWriter out = new PrintWriter(
                      new BufferedWriter(new FileWriter("romeo.out")));
    StreamTokenizer st=new StreamTokenizer(
                       new BufferedReader(new FileReader("romeo.in")));
    st.nextToken(); x=(int)st.nval;
    st.nextToken(); y=(int)st.nval;
    if(x>=y-1) e=x+y;
    else
    {
      z=y/(x+1);
      n=y-(x+1)*z;
      m=x+1-y+(x+1)*z;
      e=m*z*(z+1)/2+n*(z+1)*(z+2)/2+x;
    }
    out.println(e);
    out.close();
  }
}
```

12.3 Seceta

lect. Ovidiu Domşa

Grădinile roditoare ale Bărăganului suferă anual pierderi imense din cauza secetei. Căutătorii de apă au găsit n fântâni din care doresc să alimenteze n grădini. Fie G_i, F_i, $i = 1, ..., n$ puncte în plan reprezentând puncte de alimentare

ale grădinilor şi respectiv punctele în care se află fântânile. Pentru fiecare punct se dau coordonatele întregi (x, y) în plan.

Pentru a economisi materiale, legătura dintre o grădină şi o fântână se realizează printr-o conductă în linie dreaptă. Fiecare fântână alimentează o singură grădină. Consiliul Judeţean Galaţi plăteşte investiţia cu condiţia ca lungimea totală a conductelor să fie minimă.

Fiecare unitate de conductă costă 100 lei noi (RON).

Cerinţă

Să se determine m, costul minim total al conductelor ce leagă fiecare grădină cu exact o fântână.

Date de intrare

Fişierul de intrare **seceta.in** va conţine:

• Pe prima linie se află numărul natural n, reprezentând numărul grădinilor şi al fântânilor.

• Pe următoarele n linii se află perechi de numere întregi G_x G_y, separate printr-un spaţiu, reprezentând coordonatele punctelor de alimentare ale grădinilor.

• Pe următoarele n linii se află perechi de numere întregi F_x F_y, separate printr-un spaţiu, reprezentând coordonatele punctelor fântânilor.

Date de ieşire

Fişierul de ieşire **seceta.out** va conţine:

m − un număr natural reprezentând partea întreagă a costului minim total al conductelor.

Restricţii şi precizări

• $1 < n < 13$

• $0 \leq Gx, Gy, Fx, Fy \leq 200$

• Nu există trei puncte coliniare, indiferent dacă sunt grădini sau fântâni

• Orice linie din fişierele de intrare şi ieşire se termină prin marcajul de sfârşit de linie.

Exemplu

seceta.in	**seceta.out**	Explicaţie
3	624	Costul minim este [6.24264 * 100]=624
1 4		prin legarea perechilor:
3 3		Gradini Fantani
4 7		1 4 2 3
2 3		3 3 3 1
2 5		4 7 2 5
3 1		

Timp maxim de execuţie/test: 1 sec sub Windows şi 0.5 sec sub Linux.

12.3.1 Indicaţii de rezolvare - descriere soluţie *

Soluţia oficială, lect. Ovidiu Domşa

Numărul mic al punctelor permite generarea tuturor posibilităţilor de a conecta o grădină cu o fântână neconectată la un moment dat.

Pentru fiecare astfel de combinaţie găsită se calculează suma distanţelor (Gi, Fj), în linie dreapta, folosind formula distanţei dintre două puncte în plan, studiată la geometrie. $(d(A(x,y), B(z,t) = \sqrt{(x-z)^2 + (y-t)^2})$.

Acestă soluţie implementată corect asigură $60 - 70$ de puncte.

Pentru a obţine punctajul maxim se tine cont de următoarele aspecte:

1. Se construieşte în prealabil matricea distanţelor $d(i,j)$ cu semnificaţia distanţei dintre grădina i şi fântâna j. Aceasta va reduce timpul de calcul la variantele cu peste 9 perechi.

2. Pentru a elimina cazuri care nu pot constitui soluţii optime se foloseşte proprietatea patrulaterului că suma a doua laturi opuse (condiţie care asigură unicitatea conectării unei singure fântâni la o singură grădină) este mai mică decât suma diagonalelor. De aceea nu se vor lua în considerare acele segmente care se intersectează. Condiţia de intersecţie a două segmente care au capetele în punctele de coordonate $A(a1, a2)$, $B(b1, b2)$, $C(c1, c2)$, $D(d1, d2)$ este ca luând segmentul AB, punctele C şi D să se afle de aceeaşi parte a segmentului AB şi respectiv pentru segmentul CD, punctele A şi B să se afle de aceeaşi parte (se înlocuieşte în ecuaţia dreptei ce trece prin două puncte, studiată în clasa a 9-a).

Observaţie: Pentru cei interesaţi, problema are soluţie şi la un nivel superior, folosind algoritmul de determinare a unui flux maxim de cost minim.

12.3.2 Rezolvare detaliată

12.3.3 Codul sursă *

Variantă cu determinarea intesecţiei segmentelor.

```
import java.io.*; // cu determinarea intesectiei segmentelor
class Seceta1     // Java este "mai incet" decat Pascal si C/C++
{                 // test 9 ==> 2.23 sec
  static int nv=0;
  static int n;
  static int[] xg, yg, xf, yf, t, c;
  static int[] a; // permutare: a[i]=fantana asociata gradinii i
  static double costMin=200*1.42*12*100;
  static double[][] d;
  static PrintWriter out;
```

```
static StreamTokenizer st;

public static void main(String[] args) throws IOException
{
  long t1,t2;
  t1=System.currentTimeMillis();
  citire();
  rezolvare();
  afisare();
  t2=System.currentTimeMillis();
  System.out.println("Timp = "+(t2-t1)+" ms");
}

static void citire() throws IOException
{
  int k;
  st=new StreamTokenizer(new BufferedReader(new FileReader("seceta.in")));
  st.nextToken(); n=(int)st.nval;

  xg=new int[n+1];
  yg=new int[n+1];
  xf=new int[n+1];
  yf=new int[n+1];
  a=new int[n+1];
  d=new double[n+1][n+1];

  for(k=1;k<=n;k++)
  {
    st.nextToken(); xg[k]=(int)st.nval;
    st.nextToken(); yg[k]=(int)st.nval;
  }
  for(k=1;k<=n;k++)
  {
    st.nextToken(); xf[k]=(int)st.nval;
    st.nextToken(); yf[k]=(int)st.nval;
  }
}

static void rezolvare()  throws IOException
{
  int i,j;
  int s;
  for(i=1;i<=n;i++) // gradina i
    for(j=1;j<=n;j++) // fantana j
```

```
    {
      s=(xg[i]-xf[j])*(xg[i]-xf[j])+(yg[i]-yf[j])*(yg[i]-yf[j]);
      d[i][j]=Math.sqrt(s);
    }
  f(1); // generez permutari
}

static void f(int k)
{
  boolean ok;
  int i,j;
  for(i=1;i<=n;i++)
  {
    ok=true; // k=1 ==> nu am in stanga ... for nu se executa !
    for(j=1;j<k;j++) if(i==a[j]) {ok=false; break;}
    if(!ok) continue;
    for(j=1;j<k;j++)
      if(seIntersecteaza(xg[k],yg[k],xf[i],   yf[i],
                         xg[j],yg[j],xf[a[j]],yf[a[j]]))
      {
        ok=false;
        break;
      }
    if(!ok) continue;
    a[k]=i;
    if(k<n) f(k+1); else verificCostul();
  }
}

static void verificCostul()
{
  int i;
  double s=0;
  for(i=1;i<=n;i++) s=s+d[i][a[i]];
  if(s<costMin) costMin=s;
}

// de ce parte a dreptei [(xa,ya);(xb,yb)] se afla (xp,yp)
static int s(int xp,int yp,int xa,int ya,int xb,int yb)
{
  double s=(double)yp*(xb-xa)-xp*(yb-ya)+xa*yb-xb*ya;
  if(s<-0.001) return -1;      // in zona "negativa"
    else if(s>0.001) return 1; // in zona "pozitiva"
         else return 0;        // pe dreapta suport
```

```
}

// testeaza daca segmentul[P1,P1] se intersecteaza cu [P3,P4]
static boolean seIntersecteaza(int x1, int y1, int x2, int y2,
                               int x3, int y3, int x4, int y4)
{
  double x,y;
  if((x1==x2)&&(x3==x4))     // ambele segmente verticale
    if(x1!=x3) return false;
    else if(intre(y1,y3,y4)||intre(y2,y3,y4)) return true;
         else return false;

  if((y1==y2)&&(y3==y4))     // ambele segmente orizontale
    if(y1!=y3) return false;
    else if(intre(x1,x3,x4)||intre(x2,x3,x4)) return true;
         else return false;

  if((y2-y1)*(x4-x3)==(y4-y3)*(x2-x1))    // au aceeasi panta (oblica)
    if((x2-x1)*(y3-y1)==(y2-y1)*(x3-x1))  // au aceeasi dreapta suport
      if(intre(x1,x3,x4)||intre(x2,x3,x4)) return true;
      else return false;
    else return false;// nu au aceeasi dreapta suport
  else // nu au aceeasi panta (macar unul este oblic)
  {
    x=(double)((x4-x3)*(x2-x1)*(y3-y1)-
                x3*(y4-y3)*(x2-x1)+
                x1*(y2-y1)*(x4-x3))/
              ((y2-y1)*(x4-x3)-(y4-y3)*(x2-x1));
    if(x2!=x1) y=y1+(y2-y1)*(x-x1)/(x2-x1); else y=y3+(y4-y3)*(x-x3)/(x4-x3);
    if(intre(x,x1,x2)&&intre(y,y1,y2)&&intre(x,x3,x4)&&intre(y,y3,y4))
      return true; else return false;
  }
}

static boolean intre(int c, int a, int b) // c este in [a,b] ?
{
  int aux;
  if(a>b) {aux=a; a=b; b=aux;}
  if((a<=c)&&(c<=b)) return true; else return false;
}

static boolean intre(double c, int a, int b) // c este in [a,b] ?
{
  int aux;
```

```
    if(a>b) {aux=a; a=b; b=aux;}
    if((a<=c)&&(c<=b)) return true; else return false;
  }

  static void afisare() throws IOException
  {
    int k;
    out=new PrintWriter(new BufferedWriter(new FileWriter("seceta.out")));
    out.println((int)(costMin*100));
    out.close();
  }
}
```

Variantă cu cu determinarea pozitiei punctelor in semiplane şi mesaje pentru depanare.

```
import java.io.*;  // cu determinarea pozitiei punctelor in semiplane
class Seceta2      // cu mesaje pentru depanare !
{
  static int nv=0;
  static int n;
  static int[] xg, yg, xf, yf, t, c;
  static int[] a; // permutare: a[i]=fantana asociata gradinii i
  static double costMin=200*1.42*12*100;
  static double[][] d;
  static PrintWriter out;
  static StreamTokenizer st;

  public static void main(String[] args) throws IOException
  {
    long t1,t2;
    t1=System.currentTimeMillis();
    citire();
    rezolvare();
    afisare();
    t2=System.currentTimeMillis();
    System.out.println("Timp = "+(t2-t1)+" ms");
  }

  static void citire() throws IOException
  {
    int k;
    st=new StreamTokenizer(new BufferedReader(new FileReader("seceta.in")));
    st.nextToken(); n=(int)st.nval;
    xg=new int[n+1];
```

```
    yg=new int[n+1];
    xf=new int[n+1];
    yf=new int[n+1];
    a=new int[n+1];
    d=new double[n+1][n+1];

    for(k=1;k<=n;k++)
    {
      st.nextToken(); xg[k]=(int)st.nval;
      st.nextToken(); yg[k]=(int)st.nval;
    }
    for(k=1;k<=n;k++)
    {
      st.nextToken(); xf[k]=(int)st.nval;
      st.nextToken(); yf[k]=(int)st.nval;
    }
  }

  static void rezolvare()  throws IOException
  {
    int i,j;
    int s;
    for(i=1;i<=n;i++) // gradina i
    for(j=1;j<=n;j++) // fantana j
    {
      s=(xg[i]-xf[j])*(xg[i]-xf[j])+(yg[i]-yf[j])*(yg[i]-yf[j]);
      d[i][j]=Math.sqrt(s);
    }
    f(1); // generez permutari
  }

  static void f(int k)
  {
    boolean ok;
    int i,j;
    for(i=1;i<=n;i++)
    {
      ok=true; // k=1 ==> nu am in stanga ... for nu se executa !
      for(j=1;j<k;j++) if(i==a[j]) {ok=false; break;}
      if(!ok) continue;

      for(j=1;j<k;j++)
        if((s(xg[k],yg[k],xg[j],yg[j],xf[a[j]],yf[a[j]])*
            s(xf[i],yf[i],xg[j],yg[j],xf[a[j]],yf[a[j]])<0)&&
```

```
          (s(xg[j],   yg[j],   xg[k],yg[k],xf[i],yf[i])*
           s(xf[a[j]],yf[a[j]],xg[k],yg[k],xf[i],yf[i])<0))
       {
         afisv(k-1);// pe pozitia k(gradina) vreau sa pun i(fantana)
         System.out.print(i+"  ");// pe pozitia j(gradina) e pus a[j](fantana)
         System.out.print(k+""+i+" "+j+""+a[j]);
         System.out.print("  ("+xg[k]+","+yg[k]+") "+"  ("+xf[i]+","+yf[i]+") ");
         System.out.println("  ("+xg[j]+","+yg[j]+") "+"  ("+xf[a[j]]+","+yf[a[j]]+") ");

         ok=false;
         break;
       }
     if(!ok) continue;

     a[k]=i;
     if(k<n) f(k+1); else verificCostul();
   }
 }

 static void verificCostul()
 {
   int i;
   double s=0;
   for(i=1;i<=n;i++) s=s+d[i][a[i]];
   if(s<costMin) costMin=s;
   afisv(n); System.out.println(" "+s+" "+costMin+" "+(++nv));
 }

 static void afisv(int nn)
 {
   int i;
   for(i=1;i<=nn;i++) System.out.print(a[i]);
 }

 // de ce parte a dreptei [(xa,ya);(xb,yb)] se afla (xp,yp)
 static int s(int xp,int yp,int xa,int ya,int xb,int yb)
 {
   double s=(double)yp*(xb-xa)-xp*(yb-ya)+xa*yb-xb*ya;
   if(s<-0.001) return -1;     // in zona "negativa"
   else if(s>0.001) return 1;  // in zona "pozitiva"
        else return 0;         // pe dreapta suport
 }

 static void afisare() throws IOException
```

```
  {
    int k;
    out=new PrintWriter(new BufferedWriter(new FileWriter("seceta.out")));
    out.println((int)(costMin*100));
    out.close();
  }
}
```

Variantă cu cu determinarea pozitiei punctelor in semiplane, fărămesaje pentru depanare.

```
import java.io.*;   // cu determinarea pozitiei punctelor in semiplane
class Seceta3       // Java este "mai incet" decat Pascal si C/C++
{                   // test 9 ==> 2.18 sec
  static int n;
  static int[] xg, yg, xf, yf, t, c;
  static int[] a; // permutare: a[i]=fantana asociata gradinii i
  static double costMin=200*1.42*12*100;
  static double[][] d;
  static PrintWriter out;
  static StreamTokenizer st;

  public static void main(String[] args) throws IOException
  {
    long t1,t2;
    t1=System.currentTimeMillis();
    citire();
    rezolvare();
    afisare();
    t2=System.currentTimeMillis();
    System.out.println("Timp = "+(t2-t1)+" ms");
  }

  static void citire() throws IOException
  {
    int k;
    st=new StreamTokenizer(new BufferedReader(
                           new FileReader("seceta.in")));
    st.nextToken(); n=(int)st.nval;
    xg=new int[n+1];
    yg=new int[n+1];
    xf=new int[n+1];
    yf=new int[n+1];
    a=new int[n+1];
    d=new double[n+1][n+1];
```

```
    for(k=1;k<=n;k++)
    {
      st.nextToken(); xg[k]=(int)st.nval;
      st.nextToken(); yg[k]=(int)st.nval;
    }
    for(k=1;k<=n;k++)
    {
      st.nextToken(); xf[k]=(int)st.nval;
      st.nextToken(); yf[k]=(int)st.nval;
    }
  }

  static void rezolvare()  throws IOException
  {
    int i,j;
    int s;
    for(i=1;i<=n;i++) // gradina i
    for(j=1;j<=n;j++) // fantana j
    {
      s=(xg[i]-xf[j])*(xg[i]-xf[j])+(yg[i]-yf[j])*(yg[i]-yf[j]);
      d[i][j]=Math.sqrt(s);
    }
    f(1); // generez permutari
  }

  static void f(int k)
  {
    boolean ok;
    int i,j;
    for(i=1;i<=n;i++)
    {
      ok=true; // k=1 ==> nu am in stanga ... for nu se executa !
      for(j=1;j<k;j++) if(i==a[j]) {ok=false; break;}
      if(!ok) continue;
      for(j=1;j<k;j++)
      if((s(xg[k],   yg[k],   xg[j],yg[j],xf[a[j]],yf[a[j]])*
          s(xf[i],   yf[i],   xg[j],yg[j],xf[a[j]],yf[a[j]])<0)&&
         (s(xg[j],   yg[j],   xg[k],yg[k],xf[i],   yf[i])*
          s(xf[a[j]],yf[a[j]],xg[k],yg[k],xf[i],   yf[i])<0))
      {
        ok=false;
        break;
      }
```

```
      if(!ok) continue;
      a[k]=i;
      if(k<n) f(k+1); else verificCostul();
    }
  }

  static void verificCostul()
  {
    int i;
    double s=0;
    for(i=1;i<=n;i++) s=s+d[i][a[i]];
    if(s<costMin) costMin=s;
  }

  //de ce parte a dreptei [(xa,ya);(xb,yb)] se afla (xp,yp)
  static int s(int xp,int yp,int xa,int ya,int xb,int yb)
  {
    double s=(double)yp*(xb-xa)-xp*(yb-ya)+xa*yb-xb*ya;
    if(s<-0.001) return -1;       // in zona "negativa"
    else  if(s>0.001) return 1;   // in zona "pozitiva"
          else return 0;          // pe dreapta suport
  }

  static void afisare() throws IOException
  {
    int k;
    out=new PrintWriter(new BufferedWriter(new FileWriter("seceta.out")));
    out.println((int)(costMin*100));
    out.close();
  }
}
```

Varianta 4:

```
import java.io.*;   // gresit (!) dar ... obtine 100p ... !!!
class Seceta4       // test 9 : 2.18 sec --> 0.04 sec
{
  static int n;
  static int[] xg, yg, xf, yf, t, c;
  static int[] a;      // permutare: a[i]=fantana asociata gradinii i
  static double costMin=200*1.42*12*100;
  static double[][] d;
  static boolean[] epus=new boolean[13];

  static PrintWriter out;
```

```
  static StreamTokenizer st;

  public static void main(String[] args) throws IOException
  {
    long t1,t2;
    t1=System.currentTimeMillis();

    citire();
    rezolvare();
    afisare();

    t2=System.currentTimeMillis();
    System.out.println("Timp = "+(t2-t1)+" ms");
  }// main(...)

  static void citire() throws IOException
  {
    int k;
    st=new StreamTokenizer(new BufferedReader(new FileReader("seceta.in")));
    st.nextToken(); n=(int)st.nval;
    xg=new int[n+1];
    yg=new int[n+1];
    xf=new int[n+1];
    yf=new int[n+1];
    a=new int[n+1];

    d=new double[n+1][n+1];

    for(k=1;k<=n;k++)
    {
      st.nextToken(); xg[k]=(int)st.nval;
      st.nextToken(); yg[k]=(int)st.nval;
    }
    for(k=1;k<=n;k++)
    {
      st.nextToken(); xf[k]=(int)st.nval;
      st.nextToken(); yf[k]=(int)st.nval;
    }
  }// citire(...)

  static void rezolvare()  throws IOException
  {
    int i,j;
    int s;
```

```
    for(i=1;i<=n;i++)      // gradina i
      for(j=1;j<=n;j++)    // fantana j
      {
        s=(xg[i]-xf[j])*(xg[i]-xf[j])+(yg[i]-yf[j])*(yg[i]-yf[j]);
        d[i][j]=Math.sqrt(s);
      }
    f(1);                 // generez permutari
  }// rezolvare(...)

  static void f(int k)
  {
    int i,j;
    boolean seIntersecteaza;
    for(i=1;i<=n;i++)
    {
      if(epus[i]) continue;
      seIntersecteaza=false;
      for(j=1;j<=k-1;j++)
        if(d[k][i]+d[j][a[j]]>d[j][i]+d[k][a[j]])
        {
          seIntersecteaza=true;
          break;
        }

      if(seIntersecteaza) continue;

      a[k]=i;

      epus[i]=true;
      if(k<n) f(k+1); else verificCostul();
      epus[i]=false;
    }// for i
  }// f(...)

  static void verificCostul()
  {
    int i;
    double s=0;
    for(i=1;i<=n;i++) s=s+d[i][a[i]];
    if(s<costMin) costMin=s;
  }// verificCostul(...)

  static void afisare() throws IOException
  {
```

```
    int k;
    out=new PrintWriter(new BufferedWriter(new FileWriter("seceta.out")));
    out.println((int)(costMin*100));
    out.close();
  }// afisare(...)
}// class
```

12.4 Biblos

Maria şi Adrian Niţă

Din dorinţa de a realiza un fond de carte cât mai voluminos, oficialităţile oraşului Galaţi, au modernizat pentru început, o sală pentru depozitarea cărţilor şi l-au numit pe Biblos coordonatorul acestei biblioteci.

Achiziţionarea de carte s-a realizat în mai multe etape.

De fiecare dată cărţile achiziţionate au fost depozitate pe câte un stativ construit special de Biblos.

Pentru a avea spaţiu de depozitare Biblos a construit mai multe stative decât i-ar fi fost necesare, unele putând rămâne fără cărţi.

După mai multe etape de achiziţionare, Biblos a constatat că spaţiul alocat bibliotecii este prea mic.

Primind un alt spaţiu mai încăpător, mută primul stativ cu toate cărţile conţinute de acesta şi se opreşte deoarece îşi doreşte să mute acele stative care nu sunt aşezate unul lângă celălalt şi care fac ca fondul de carte din noua sală să fie cât mai mare posibil.

Cerinţă

Scrieţi un program care, cunoscând numărul stativelor, precum şi numărul de volume de carte de pe fiecare stativ, determină care este numărul maxim de volume care pot fi mutate în noua sală, ştiind că primul stativ a fost deja mutat iar celelalte se aleg astfel încât să nu fie aşezate unul lângă celălalt. Dacă există stative care nu au cărţi acestea nu vor fi mutate în a doua sală.

Date de intrare

Fişierul de intrare **biblos.in** conţine

• pe prima linie o valoare n, număr natural cu semnificaţia numărul de stative,

• pe a doua linie n numere naturale, x_1, x_2, ..., x_n separate prin câte un spaţiu cu semnificaţia

x_i = numărul de volume de carte existente pe fiecare stativ.

Date de ieşire

Fişierul de ieşire **biblos.out** va conţine o singură linie unde se află un număr natural cu semnificaţia: numărul maxim de volume ce au fost transferate.

Restricţii şi precizări

• $1 \leq n \leq 30000$

• $0 \leq x_i \leq 32767$, unde $i = 1, ..., n$ iar x_i reprezintă numărul de cărţi de pe stativul i.

• Pentru 70% dintre teste $n \leq 1000$

• Fiecare linie din fişierul de intrare şi din fişierul de ieşire se termină cu marcaj de sfârşit de linie.

Exemple

biblos.in	**biblos.out**
7 1 3 6 2 5 8 4	16

Explicaţie: Suma maximă se obţine din mutarea stativelor 1 (obligatoriu), 3, 5, 7 (nu pot fi stative alăturate)

biblos.in	**biblos.out**
15 3 1 84 9 89 55 135 49 176 238 69 112 28 175 142	836

Explicaţie: Suma maximă obţinută din mutarea stativelor 1, 3, 5, 7, 10, 12, 14

biblos.in	**biblos.out**
8 7 1 4 12 9 9 12 4	32

Explicaţie: Suma maximă obţinută din mutarea stativelor 1, 3, 5, 7, sau din mutarea stativelor 1, 4, 6, 8.

Timp maxim de execuţie/test: 0.5 sec sub Windows şi 0.1 sec sub Linux

12.4.1 Indicaţii de rezolvare - descriere soluţie *

Soluţia oficială, Maria şi Adrian Niţă

Problema cere determinarea unei sume maxime formată din numerele citite, astfel încât să se aleagă doar valori care nu sunt pe poziţii consecutive.

Structurile folosite sunt:

• $carti(1)$, $carti(2)$, ... $carti(n)$

unde $carti(i)$ = numărul de volume existente pe stativul i ($i = 1, ..., n$).

• $cmax(1)$, $cmax(2)$, ..., $cmax(n)$

unde $cmax(i)$ = numărul maxim de volume ce pot fi transferate utilizând cărţile existente pe stativele de la 1 la $i - 2$, astfel încât să fie selectat şi raftul i.

Primul stativ fiind întotdeauna transferat atunci realizarea lui $cmax$ se poate face astfel:

$$cmax(1) = carti(1);$$

Pentru toate celelalte valori $cmax(i)$, se observă că se poate folosi formula:

$$cmax(i) = carti(i) + maxim\{cmax(k) \text{ unde } k = 1, ..., i-2\}$$

Prin alegerea maximului dintre valorile lui $cmax$, se obţine rezultatul cerut.

Pentru optimizare, se poate face observaţia că dintre valorile lui $cmax$, interesează doar cele de pe poziţiile $i-3$ şi $i-2$, deci:

dacă $cmax(i-2) > cmax(i-3)$

atunci $cmax(i) = cmax(i-2) + carti(i)$

altfel $cmax(i) = cmax(i-3) + carti(i)/$

12.4.2 Rezolvare detaliată

12.4.3 Codul sursă *

Variantă pentru depanare.

```
import java.io.*;
class Biblos1
{
  static int n;
  static int[] x;
  static int[] smax;
  static int[] p; // predecesor, pentru depanare

  public static void main(String []args) throws IOException
  {
    long t1,t2;
    t1=System.currentTimeMillis();

    int i,j,max,jmax,ji;
    StreamTokenizer st=new StreamTokenizer(
                       new BufferedReader(new FileReader("biblos.in")));
    PrintWriter out = new PrintWriter (
                      new BufferedWriter( new FileWriter("biblos.out")));

    st.nextToken();n=(int)st.nval;
    x=new int[n+1];
    smax=new int[n+1];
    p=new int[n+1];

    for(i=1;i<=n;i++) { st.nextToken(); x[i]=(int)st.nval;}
```

```
    smax[1]=x[1];
    for(i=2;i<=n;i++)
    {
      max=0; jmax=0;
      if(i==2) ji=0; else ji=i-3;
      for(j=ji;j<=i-2;j++)
        if(smax[j]>=max) { max=smax[j]; jmax=j; }
      if(max!=0) { smax[i]=max+x[i]; p[i]=jmax;} else {smax[i]=0; p[i]=0;}
    }

    if(smax[n]>smax[n-1]) {max=smax[n]; jmax=n;}
      else {max=smax[n-1]; jmax=n-1;}
    out.println(max);
    out.close();

    t2=System.currentTimeMillis();
    drum(jmax);
    System.out.println("Timp = "+(t2-t1)+" ms");
  }//main

  static void drum(int j)
  {
    if(p[j]!=0) drum(p[j]);
    System.out.println(j+" "+x[j]+" "+smax[j]);
  }
}//class
```

Variantă cu vectori.

```
import java.io.*;
class Biblos2
{
  static int n;
  static int[] x;
  static int[] smax;

  public static void main(String []args) throws IOException
  {
    int i,j,max,jmax,ji;
    StreamTokenizer st=new StreamTokenizer(
                       new BufferedReader(new FileReader("biblos.in")));
    PrintWriter out = new PrintWriter (
                      new BufferedWriter( new FileWriter("biblos.out")));
```

```
    st.nextToken();n=(int)st.nval;
    x=new int[n+1];
    smax=new int[n+1];

    for(i=1;i<=n;i++) { st.nextToken(); x[i]=(int)st.nval;}

    smax[1]=x[1];
    for(i=2;i<=n;i++)
    {
      max=0; jmax=0;
      if(i==2) ji=0; else ji=i-3;
      for(j=ji;j<=i-2;j++)
        if(smax[j]>=max) { max=smax[j]; jmax=j; }
      if(max!=0) smax[i]=max+x[i]; else smax[i]=0;
    }

    if(smax[n]>smax[n-1]) max=smax[n];  else max=smax[n-1];
    out.println(max);
    out.close();
  }//main
}//class
```

Variantă fără vectori (optimizat spaţiul de memorie folosit).

```
import java.io.*; // s-a renuntat la vectorii x si smax
class Biblos3
{
  static int n;
  static int xi;
  static int si,si1,si2,si3;

  public static void main(String []args) throws IOException
  {
    int i,j,max,jmax,ji;
    StreamTokenizer st=new StreamTokenizer(
                       new BufferedReader(new FileReader("biblos.in")));
    PrintWriter out = new PrintWriter (
                      new BufferedWriter( new FileWriter("biblos.out")));

    st.nextToken();n=(int)st.nval;
    st.nextToken(); xi=(int)st.nval;
    si3=0;
    si2=xi;
    si1=0;
    st.nextToken(); xi=(int)st.nval; // citesc x[2] si il neglijez !
```

```
    for(i=3;i<=n;i++)
    {
      st.nextToken(); xi=(int)st.nval;
      if(si2>si3) si=si2+xi; else si=si3+xi;
      si3=si2;
      si2=si1;
      si1=si;
    }

    if(si1>si2) out.println(si1); else out.println(si2);
    out.close();
  }//main
}//class
```

12.5 Joc

Cristina Luca

Pe o tablă pătrată de dimensiune $n \times n$ se desenează o secvenţă de triunghiuri dreptunghic isoscele.

Fiecare triunghi are vârfurile numerotate cu 1, 2 şi 3 (2 corespunde unghiului drept iar ordinea 1, 2, 3 a vârfurilor este în sens invers acelor de ceasornic - vezi figura). Triunghiurile au catetele paralele cu marginile tablei.

Primul triunghi, având lungimea catetei Lg, are vârful 1 pe linia L şi coloana C şi este orientat ca în figură.

				3	
				a	
			a	**a**	
		a	**a**	**a**	
1	**a**	**a**	**a**	**a**	2

Jocul constă în alipirea câte unui nou triunghi la unul din vârfurile 2 sau 3 ale triunghiului curent. Dacă se alătură colţului 2, noul triunghi se aşează cu vârful 1 în prelungirea laturii $[1, 2]$ a triunghiului curent, iar dacă se alătură colţului 3 se aşează cu vârful 1 în prelungirea laturii $[2, 3]$.

Iniţial noul triunghi este orientat ca şi cel anterior. El se poate plasa pe tablă dacă nu sunt depăşite marginile acesteia sau nu se suprapune peste un alt triunghi. În caz contrar, se face o singurărotaţie cu 90^o spre stânga, obţinându-se o nouă orientare a triunghiului. Dacă nici în acest caz noul triunghi nu poate fi plasat, jocul se opreşte.

Zona ocupată de primul triunghi se completează cu litera 'a'; zona celui de-al doilea se completezăcu litera 'b', ş.a.m.d. Când literele mici ale alfabetului englez sunt epuizate, se reîncepe de la 'a'.

Cerinţă

Cunoscându-se dimensiunea tablei, poziţia primului triunghi (linie, coloană) şi lungimea catetei precum şi o secvenţă de triunghiuri care se doresc a fi alipite se cere să se genereze matricea rezultată în finalul jocului.

Jocul se termină dacă un triunghi nu mai poate fi alipit sau au fost plasate toate triunghiurile descrise în secvenţă.

Date de intrare

În fişierul de intrare **joc.in**, pe prima linie se află n (dimensiunea tablei). Pe a doua linie separate prin câte un spaţiu se află: L (linia), C (coloana) şi Lg (lungimea catetei) corespunzătoare primului triunghi. Următoarele linii, până la sfârşitul fişierului, conţin câte două numere naturale separate prin câte un singur spaţiu reprezentând colţul triunghiului curent la care va fi alipit triunghiul următor şi dimensiunea catetei triunghiului urmăor.

Date de ieşire

În fişierul de ieşire **joc.out** va fi afişată matricea rezultată. Celulele tablei care nu sunt completate cu litere ale alfabetului vor fi completate cu '.'.

Restricţii şi precizări

• $1 \leq n \leq 100$; $1 \leq C$, $L \leq n$; $2 \leq Lg \leq n$

• Fiecare linie din fişierul de intrare şi din fic serul de iec sre se termină cu marcaj de sfârşit de linie.

Exemplu

joc.in	joc.out
20	. . . .
16 8 4	. . . .
3 5	. . . . f f f f f e e e . .
2 3	. . . . f f f f . . e e . .
3 4	. . . . f f f . . . . e . .
2 3	. . . . f f . . d d d d . .
3 5	. . . . f . . . . d d d . .
3 3	. . . . . h h g g g . . . b . d d . .
2 2	. . . . . h . g g . . . b b . . d . .
3 4	j j j i i i i . g . . . b b b . . c . .
2 3	j j . i i i . . . . . b b b b . c c . .
3 3	j . . i i . . . . . b b b b b c c c . .
3 2	k . . i . . . . . . a . . . .
3 3	k k . . . . a a . . . .
3 3	k k k l . . . . a a a . . . .
2 4	. . . l l m . a a a a . . . .
	. . . . . m m . . . .
	. . . . . m m m n . . . .
	. . . . . n n . . . .
	. . . . . n n n . . . .

Explicaţii

– Triunghiul 'a' este plasat în linia 16 coloana 8 şi are latura 4.

– Triunghiul 'b' se alipeşte în colţul 3 şi are lungimea 5.

– Triunghiul 'c' se alipeşte în colţul 2 şi are lungimea 3.

– Tringhiurile 'a', 'b' şi 'c' păstrează aceeaşi aranjare.

– Triunghiul 'd' nu se poate alipi în aceeaşi aranjare colţului 3 deoarece are cateta de lungimea 4 şi depăşeşte tabla. Rotim triunghiul cu 90^o spre stânga şi obţinem o nouă aranjare.

– Triunghiul 'e' se alipeşte în colţul 2 şi are cateta de lungime 3 în aranjarea curentă.

– Triunghiul 'f' nu se poate alipi în aceeaşi aranjare cu 'e' în colţul 3 deoarece are cateta de lungimea 5 şi depăşeşte tabla. Rotim triunghiul cu 90^o spre stânga şi obţinem o nouă aranjare. Triunghiul 'f' se alipeşte în colţul 3, are lungimea 5 şi o nouă aranjare.

– Algoritmul continuă până la al 14-lea triunghi, 'n'.

– Al 15-lea triunghi nu se mai poate plasa.

Timp maxim de execuţie/test: 0.2 sec în Windows şi 0.2 sec sub Linux

12.5.1 Indicaţii de rezolvare - descriere soluţie *

Soluţia oficială, Cristina Luca

Soluţia problemei constă în gestionarea spaţiului unei matrice folosind proprietăţi ale indicilor, parcurgerea pe linii, pe coloane, pe diagonale, fiecare în două sensuri.

Plasarea triunghiurilor pe tabla de joc presupune verificarea următoarelor două situaţii:

1. cel puţin un element al zonei triunghiului care urmează să fie plasat se află în afara tablei;

a. în exemplul de mai jos situaţiile triunghiurilor 'd', 'f', 'k', triunghiuri care nu pot fi plasate în aceeaşi orientare cu cele care le preced deoarece lungimile catetelor depăşesc marginile dreaptă, sus, respectiv stâangă ale matricei;

b. dacă triunghiul 'k' ar fi plasat în poziţia 2 faţă de triunghiul 'j' ar avea vârful în afara tablei;

2. cel puţin un element al zonei triunghiului care urmează să fie plasat se suprapune peste un element al unui triunghi deja plasat.

Verificarea celor două situaţi se face şi dacă se încearcă rotirea cu 90^o a triunghiului. De exemplu, dacă triunghiul 'd' nu este rotit apare situaţia 1.a. De aceea el este rotit obţinându-se noua orientare în care nu apar situaţiile 1 sau 2.

Există două situaţii în care jocul se opreşte:

• se epuizează secvenţa de triunghiuri descrise în fişier;

• triunghiul nu poate fi plasat nici direct nici după o singură rotire; restul triunghiurilor nu vor mai fi luate în considerare: în exemplul de mai jos ultimul triunghi, (24), nu va mai fi plasat.

Exemplu

joc.in	**joc.out**
20	. . . .
16 8 4	. . . .
3 5	. . . . f f f f f e e e . .
2 3	. . . . f f f f . . e e . .
3 4	. . . . f f f . . . . e . .
2 3	. . . . f f . . d d d d . .
3 5	. . . . f . . . . d d d . .
3 3	. . . . . h h g g g . . . b . d d . .
2 2	. . . . . h . g g . . . b b . . d . .
3 4	j j j i i i i . g . . . b b b . . c . .
2 3	j j . i i i . . . . . b b b b . c c . .
3 3	j . . i i . . . . . b b b b b c c c . .
3 2	k . . i . . . . . . a . . . .
3 3	k k . . . . a a . . . .
3 3	k k k l . . . . a a a . . . .
2 4	. . . l l m . a a a a . . . .
	. . . . . m m . . . .
	. . . . . m m m n . . . .
	. . . . n n . . . .
	. . . . n n n . . . .

12.5.2 Rezolvare detaliată

12.5.3 Codul sursă *

```
import java.io.*;
class Joc
{
  static int n;
  static int[][] a;
  static int val; // numar/caracter de umplere: 0,1,...,25 ==> (val+1)%26
  static int dir; // directia: 0,1,2,3 circular ==> (dir+1)%4

  public static void main(String[] args) throws IOException
  {
```

```
long t1,t2;
t1=System.currentTimeMillis();
int v,d,i,j;
int l1, c1, d1;
boolean ok=true;
StreamTokenizer st=new StreamTokenizer(
                   new BufferedReader(new FileReader("joc.in")));
PrintWriter out=new PrintWriter(
                new BufferedWriter(new FileWriter("joc.out")));
st.nextToken(); n=(int)st.nval;

a=new int[n+1][n+1];
for(i=1;i<=n;i++) for(j=1;j<=n;j++) a[i][j]=-1;

st.nextToken(); l1=(int)st.nval; // linia varful 1
st.nextToken(); c1=(int)st.nval; // coloana varful 1
st.nextToken(); d1=(int)st.nval; // latura

val=0;
dir=0;
if(esteGol0(l1,c1,d1)) umple0(l1,c1,d1);
else if(esteGol1(l1,c1,d1)) {dir=(dir+1)%4; umple1(l1,c1,d1);}

while(ok&&(st.nextToken()!=StreamTokenizer.TT_EOF))
{
  v=(int)st.nval;
  st.nextToken(); d=(int)st.nval;
  val=(val+1)%26;

  if(v==2)
  switch(dir)
  {
    case 0: if(esteGol0(l1,c1+d1,d)) // direct
            {
              c1=c1+d1; d1=d; umple0(l1,c1,d1);
            }
            else if(esteGol1(l1,c1+d1,d)) // rotit
                 {
                   c1=c1+d1; d1=d;
                   dir=(dir+1)%4;
                   umple1(l1,c1,d1);
                 }
                 else ok=false;
            break;
```

```
  case 1: if(esteGol1(l1-d1,c1,d)) // direct
          {
            l1=l1-d1; d1=d; umple1(l1,c1,d1);
          }
          else if(esteGol2(l1-d1,c1,d)) // rotit
               {
                 l1=l1-d1; d1=d;
                 dir=(dir+1)%4;
                 umple2(l1,c1,d1);
                }
                else ok=false;
          break;
  case 2: if(esteGol2(l1,c1-d1,d)) // direct
          {
            c1=c1-d1; d1=d; umple2(l1,c1,d1);
          }
          else if(esteGol3(l1,c1-d1,d)) // rotit
               {
                 c1=c1-d1;d1=d;
                 dir=(dir+1)%4;
                 umple3(l1,c1,d1);
               }
               else ok=false;
          break;
  case 3: if(esteGol3(l1+d1,c1,d)) // direct
          {
            l1=l1+d1;d1=d; umple3(l1,c1,d1);
          }
          else if(esteGol0(l1+d1,c1,d)) // rotit
               {
                 l1=l1+d1; d1=d;
                 dir=(dir+1)%4;
                 umple0(l1,c1,d1);
               }
               else ok=false;
          break;
 default: System.out.println("Ciudat!!!");
}

if(v==3)
switch(dir)
{
  case 0: if(esteGol0(l1-d1,c1+d1-1,d)) // direct
          {
```

```
          c1=c1+d1-1; l1=l1-d1; d1=d; umple0(l1,c1,d1);
        }
        else if(esteGol1(l1-d1,c1+d1-1,d))  // rotit
             {
               l1=l1-d1; c1=c1+d1-1; d1=d;
               dir=(dir+1)%4;
               umple1(l1,c1,d1);
             }
             else ok=false;
        break;
case 1: if(esteGol1(l1-d1+1,c1-d1,d)) // direct
        {
          l1=l1-d1+1; c1=c1-d1; d1=d; umple1(l1,c1,d1);
        }
        else if(esteGol2(l1-d1+1,c1-d1,d)) // rotit
             {
               l1=l1-d1+1; c1=c1-d1; d1=d;
               dir=(dir+1)%4;
               umple2(l1,c1,d1);
             }
             else ok=false;
        break;
case 2: if(esteGol2(l1+d1,c1-d1+1,d)) // direct
        {
          c1=c1-d1+1; l1=l1+d1; d1=d; umple2(l1,c1,d1);
        }
        else if(esteGol3(l1+d1,c1-d1+1,d)) // rotit
             {
               c1=c1-d1+1; l1=l1+d1; d1=d;
               dir=(dir+1)%4;
               umple3(l1,c1,d1);
             }
             else ok=false;
        break;
case 3: if(esteGol3(l1+d1-1,c1+d1,d)) // direct
        {
          l1=l1+d1-1;c1=c1+d1; d1=d; umple3(l1,c1,d1);
        }
        else if(esteGol0(l1+d1-1,c1+d1,d)) // rotit
             {
               l1=l1+d1-1;c1=c1+d1; d1=d;
               dir=(dir+1)%4;
               umple0(l1,c1,d1);
             }
```

```
                    else ok=false;
                break;
        default: System.out.println("Ciudat!!!");
      }
    }

    for(i=1;i<=n;i++)
    {
      for(j=1;j<=n;j++)
      if(a[i][j]==-1) out.print("."); else out.print((char)(a[i][j]+'a'));
      out.println();
    }
    out.println();
    out.close();

    t2=System.currentTimeMillis();
    System.out.println("Timp = "+(t2-t1)+" ms");
  }

  static void umple0(int lin, int col, int d) // 12=sd==>dir=0
  {
    int i,j,k;
    for(k=0;k<d;k++)
    {
      i=lin-k;
      for(j=col+k;j<col+d;j++) a[i][j]=val;
    }
  }

  static boolean esteGol0(int lin, int col, int d)
  {
    int i,j,k;
    if((lin-0>n)||(lin-d+1<1)||(col<1)||(col+d-1>n)) return false;
    boolean gol=true;
    for(k=0;k<d;k++)
    {
      i=lin-k;
      for(j=col+k;j<col+d;j++) if(a[i][j]!=-1){ gol=false;break;}
    }
    return gol;
  }

  static void umple1(int lin, int col, int d) // 12=js==>dir=1
  {
```

```
    int i,j,k;
    for(k=0;k<d;k++)
    {
      i=lin-k;
      for(j=col-k;j<=col;j++) a[i][j]=val;
    }
  }

  static boolean esteGol1(int lin, int col, int d)
  {
    int i,j,k;
    if((lin-0>n)||(lin-d+1<1)||(col-d+1<1)||(col>n)) return false;
    boolean gol=true;
    for(k=0;k<d;k++)
    {
      i=lin-k;
      for(j=col-k;j<=col;j++) if(a[i][j]!=-1) {gol=false; break;}
    }
    return gol;
  }

  static void umple2(int lin, int col, int d) // 12=ds==>dir=2
  {
    int i,j,k;
    for(k=0;k<d;k++)
    {
      i=lin+k;
      for(j=col-d+1;j<=col-k;j++) a[i][j]=val;
    }
  }

  static boolean esteGol2(int lin, int col, int d)
  {
    int i,j,k;
    if((lin<1)||(lin+d-1>n)||(col-d+1<1)||(col>n)) return false;
    boolean gol=true;
    for(k=0;k<d;k++)
    {
      i=lin+k;
      for(j=col-d+1;j<=col-k;j++) if(a[i][j]!=-1){ gol=false;break;}
    }
    return gol;
  }
```

```
  static void umple3(int lin, int col, int d) // 12=sj==>dir=3
  {
    int i,j,k;
    for(k=0;k<d;k++)
    {
      i=lin+k;
      for(j=col;j<=col+k;j++) a[i][j]=val;
    }
  }

  static boolean esteGol3(int lin, int col, int d)
  {
    int i,j,k;
    if((lin<1)||(lin+d-1>n)||(col<1)||(col+d-1>n)) return false;
    boolean gol=true;
    for(k=0;k<d;k++)
    {
      i=lin+k;
      for(j=col;j<=col+k;j++) if(a[i][j]!=-1) {gol=false; break;}
    }
    return gol;
  }
}
```

12.6 Pal

Autor: Silviu Gănceanu

Prinţul Algorel este în încurcătură din nou: a fost prins de Spânul cel Negru în încercarea sa de a o salva pe prinţesă şi acum este închis în Turnul cel Mare.

Algorel poate evada dacă găseşte combinaţia magică cu care poate deschide poarta turnului.

Prinţul ştie cum se formează această combinaţie magică: trebuie să utilizeze toate cifrele scrise pe uşa turnului pentru a obţine două numere palindroame, astfel încât suma lor să fie minimă, iar această sumă este combinaţia magică ce va deschide uşa.

Primul număr palindrom trebuie să aibă cel puţin L cifre, iar cel de-al doilea poate avea orice lungime diferită de 0. Numerele palindroame formate nu pot începe cu cifra 0. Acum interveniţi dumneavoastră în poveste, fiind prietenul său cel mai priceput în algoritmi.

Prin noul super-telefon al său, prinţul transmite numărul de apariţii a fiecărei cifre de pe uşa turnului precum şi lungimea minimă L a primului număr, iar dumneavoastră trebuie să-i trimiteţi cât mai repede numerele cu care poate obţine

combinaţia magică.

Cerinţă

Având datele necesare, aflaţi două numere palindroame cu care se poate obţine combinaţia magică.

Date de intrare

Prima linie a fişierului **pal.in** conţine un număr întreg L reprezentând lungimea minimă a primului număr. Urmează 10 linii: pe linia $i + 2$ se va afla un număr întreg reprezentând numărul de apariţii ale cifrei i, pentru i cu valori de la 0 la 9.

Date de ieşire

Prima linie a fişierului de ieşire **pal.out** conţine primul număr palidrom, iar cea de-a doua linie conţine cel de-al doilea număr palindrom. Dacă există mai multe soluţii se va scrie doar una dintre ele.

Restricţii şi precizări

• În total vor fi cel mult 100 de cifre

• $1 \leq L < 100$ şi L va fi mai mic decât numărul total de cifre

• Pentru datele de test va exista întotdeauna soluţie: se vor putea forma din cifrele scrise pe uşa turnului două numere care încep cu o cifră diferită de 0, iar primul număr să aibă cel puţin L cifre

• Un număr este palindrom dacă el coincide cu răsturnatul său. De exemplu 12321 şi 7007 sunt numere palindroame, în timp ce 109 şi 35672 nu sunt.

• Pentru 30% dintre teste, numărul total de cifre va fi cel mult 7; pentru alte 40% din teste numărul total de cifre va fi cel mult 18, iar pentru restul de 30% din teste numărul total de cifre va fi mai mare sau egal cu 30.

• Fiecare linie din fişierul de intrare şi din fişierul de ieşire se termină cu marcaj de sfârşit de linie.

Exemplu

pal.in	**pal.out**	Explicaţie
5	10001	Pentru acest exemplu avem $L = 5$,
3	222	3 cifre de 0, 2 cifre de 1şi 3 cifre de 2.
2		Cifrele de la 3 la 9 lipsesc
3		de pe uşa turnului.
0		
0		Cele două palindroame cu care
0		se generează combinaţia magică
0		sunt 10001 şi 222.
0		Combinaţia magică va fi suma acestora
0		şi anume 10223 (care este suma minimă
0		pe care o putem obţine).

Timp maxim de execuţie/test: 1 sec sub Windows şi 1 sec sub Linux

12.6.1 Indicaţii de rezolvare - descriere soluţie *

Soluţia oficială, Silviu Gănceanu

Problema se rezolvă utilizând tehnica greedy. Notăm numărul total de cifre cu NC. Se observă că, pentru ca suma celor două numere palindroame să fie minimă, trebuie ca lungimea maximă a celor două numere să fie cât mai mică. Aşadar, pentru început, se stabileşte lungimea exactă a primului număr (care va fi cel mai lung), în funcţie de L în modul următor:

1. dacă $L < NC/2$, atunci lungimea primului palindrom va fi aleasă astfel încât cele două numere să fie cât mai apropiate ca lungime

2. dacă $L >= NC/2$ apar cazuri particulare şi se stabileşte dacă lungimea primului palindrom creşte sau nu cu o unitate.

Având lungimile numerelor stabilite putem merge mai departe utilzând strategia greedy, observând că cifrele mai mici trebuie să ocupe poziţii cât mai semnificative. Pentru a realiza acest lucru se vor completa în paralel cele două numere cu cifrele parcurse în ordine crescătoare stabilind în care din cele două numere se vor poziţiona. De asemenea, trebuie avut în vedere ca niciunul din cele două numere să nu înceapă cu cifra 0.

Datele de test au fost construite astfel încât şi soluţii neoptime să obţină puncte, gradat, în funcţie de optimizările efectuate asupra implementării.

12.6.2 Rezolvare detaliată

12.6.3 Codul sursă *

```
import java.io.*;  // la inceput cifre mici in p1
class Pal          // apoi in ambele in paralel!
{
  static int L;
  static int NC=0;
  static int nc1,nc2;
  static int ncfi=0;           // nr cifre cu frecventa impara
  static int[] fc=new int[10]; // frecventa cifrelor
  static int[] p1;             // palindromul 1
  static int[] p2;             // palindromul 2

  public static void main(String []args) throws IOException
  {
    int i;
    StreamTokenizer st=new StreamTokenizer(
```

```
                    new BufferedReader(new FileReader("pal.in")));
  PrintWriter out =  new PrintWriter (
                    new BufferedWriter( new FileWriter("pal.out")));

  st.nextToken();L=(int)st.nval;
  for(i=0;i<=9;i++) { st.nextToken(); fc[i]=(int)st.nval;}

  for(i=0;i<=9;i++) NC+=fc[i];           // nr total cifre
  for(i=0;i<=9;i++) ncfi=ncfi+(fc[i]%2); // nr cifre cu frecventa impara

  nc1=L;
  nc2=NC-nc1;
  while(nc1<nc2) {nc1++; nc2--;}
  if((ncfi==2)&&(nc1%2==0)) {nc1++; nc2--;}

  p1=new int[nc1];
  p2=new int[nc2];
  switch(ncfi)
  {
    case 0: impare0(); break;
    case 1: impare1(); break;
    case 2: impare2(); break;
    default: System.out.println("Date de intrare eronate!");
  }
  corectez(p1);
  corectez(p2);

  for(i=0;i<p1.length;i++) out.print(p1[i]);
  out.println();
  for(i=0;i<p2.length;i++) out.print(p2[i]);
  out.println();
  int[] p12=suma(p1,p2);// pentru ca in teste rezultat = suma!
  for(i=p12.length-1;i>=0;i--) out.print(p12[i]);
  out.println();
  out.close();
}//main

static void impare0() // cea mai mare cifra la mijloc
{
  int i,k1,k2;
  int imp1=nc1/2, imp2=nc2/2;

  if(nc1%2==1) // numai daca nc1 si nc2 sunt impare !
    for(i=9;i>=0;i--)
```

```
        if(fc[i]>0)
        {
          p1[imp1]=i; fc[i]--;
          p2[imp2]=i; fc[i]--;
          break;
        }
    k1=0;
    k2=0;
    while(k1<nc1-nc2) {incarcPozitia(k1,p1); k1++;}// incarc numai p1
    while((k1<imp1)||(k2<imp2))
    {
      if(k1<imp1) {incarcPozitia(k1,p1); k1++;}
      if(k2<imp2) {incarcPozitia(k2,p2); k2++;}
    }
  }

  static void impare1()
  {
    int i,k1,k2;
    int imp1=nc1/2, imp2=nc2/2;

    for(i=0;i<=9;i++)
      if(fc[i]%2==1) { p1[imp1]=i; fc[i]--; break;}
    for(i=0;i<=9;i++)
      if(fc[i]%2==1) { p2[imp2]=i; fc[i]--; break;}

    k1=0;
    k2=0;
    while(k1<nc1-nc2) {incarcPozitia(k1,p1); k1++;} // incarc numai p1
    while((k1<imp1)||(k2<imp2))
    {
      if(k1<imp1) {incarcPozitia(k1,p1); k1++;}
      if(k2<imp2) {incarcPozitia(k2,p2); k2++;}
    }
  }

  static void impare2()
  {
    int i,k1,k2;
    int imp1=nc1/2, imp2=nc2/2;

    for(i=0;i<=9;i++)
      if(fc[i]%2==1) { p1[imp1]=i; fc[i]--; break;}
    for(i=0;i<=9;i++)
```

```
      if(fc[i]%2==1) { p2[imp2]=i; fc[i]--; break;}

    k1=0;
    k2=0;
    while(k1<nc1-nc2) {incarcPozitia(k1,p1); k1++;}// incarc numai p1
    while((k1<imp1)||(k2<imp2))
    {
      if(k1<imp1) { incarcPozitia(k1,p1); k1++; }
      if(k2<imp2) { incarcPozitia(k2,p2); k2++; }
    }
  }

  static void corectez(int[] x)
  {
    int pozdif0,val, n=x.length;
    pozdif0=0;
    while(x[pozdif0]==0) pozdif0++;
    if(pozdif0>0)
    {
      val=x[pozdif0];
      x[0]=x[n-1]=val;
      x[pozdif0]=x[n-pozdif0-1]=0;
    }
  }

  static void incarcPozitia(int k, int[] x)
  {
    int i;
    int n=x.length;
    for(i=0;i<=9;i++)
      if(fc[i]>0)
      {
        x[k]=i; fc[i]--;
        x[n-k-1]=i; fc[i]--;
        break;
      }
  }

  static int[] suma(int[] x, int[] y)
  {
    int[] z=new int[(x.length>y.length) ? (x.length+1) : (y.length+1)];
    int k,t=0;
    for(k=0;k<=z.length-2;k++)
    {
```

```
      z[k]=t;
      if(k<x.length) z[k]+=x[k];
      if(k<y.length) z[k]+=y[k];
      t=z[k]/10;
      z[k]=z[k]%10;
    }
    z[z.length-1]=t;
    if(z[z.length-1]!=0) return z;
    else
    {
      int[] zz=new int[z.length-1];
      for(k=0;k<zz.length;k++) zz[k]=z[k];
      return zz;
    }
  }
}//class
```

Capitolul 13

ONI 2006 clasa a IX-a

13.1 Factorial

autor

Pentru un număr natural nenul, definim factorialul său ca fiind produsul tuturor numerelor naturale nenule mai mici sau egale decât el şi îl notăm $N!$ (adică $N! = 1 * 2 * ... * N$). Pentru o bază de numeraţie B şi un număr natural nenul N, se cere determinarea ultimei cifre nenule a scrierii în baza B a lui $N!$.

Cerinţă

Se citesc 5 perechi de forma (N_i, B_i), unde $1 \leq i \leq 5$. Pentru fiecare din cele 5 perechi citite, aflaţi ultima cifră nenulă a scrierii în baza B_i a factorialului numărului N_i.

Date de intrare

Fişierul de intrare **fact.in** conţine 5 linii, pe fiecare dintre ele fiind scrise

câte două numere naturale nenule N_i şi B_i, scrise în baza 10, despărţite printr-un spaţiu.

Date de ieşire

Fişierul de ieşire **fact.out** va conţine 5 linii. Pe linia i se va afla cifra corespunzătoare unei perechi (N_i, B_i), citită de pe linia i din fişierul de intrare.

Restricţii şi precizări

- $1 \leq N_i \leq 1000000$, pentru $1 \leq i \leq 5$;
- $2 \leq B_i \leq 36$, pentru $1 \leq i \leq 5$;
- în cazul în care $B_i > 10$, cifrele mai mari decât 9 vor fi reprezentate prin litere mari ale alfabetului englez ($10 =' A'$, $11 =' B'$,...,$35 =' Z'$);
- un test va fi punctat doar dacă toate cele 5 rezultate cerute sunt corecte.

Exemplu

.in	.out	Explicaţie
5 10	2	$5! = 120$, în baza 10, deci ultima cifră nenulă este 2
7 10	4	$7! = 5040$, în baza 10, deci ultima cifră nenulă este 4
7 20	C	$7! = CC0$, în baza 20, deci ultima cifră nenulă este C
8 16	8	$8! = 9D80$, în baza 16, deci ultima cifră nenulă este 8
9 8	6	$9! = 1304600$, în baza 8, deci ultima cifră nenulă este 6

Timp maxim de execuţie/test: 1 secundă (Windows), 0.3 secunde (Linux)

13.1.1 Indicaţii de rezolvare - descriere soluţie *

Soluţia comisiei

Se descompune mai întâi B în produs de factori primi. Retinem în $d[i][0]$ factorul prim cu numarul de ordine i şi în $d[i][1]$ puterea la care apare acesta în descompunerea lui B.

Apoi se parcurg numerele de la 1 la N şi din fiecare dintre acestea se elimină toţi factorii primi ai lui B, reţinuţi în $d[i][0]$. Se păstrează totodată în $d[i][2]$ puterea la care ar fi aparut aceştia în descompunerea lui $N!$. Odată cu eliminarea factorilor primi ai lui B din $N!$, se calculează şi restul la împărţirea cu B al produsului (notat p).

Se calculează apoi numărul de zerouri consecutive care se vor afla la sfârşitul lui $N!$, reprezentat în baza B: minimul rapoartelor dintre $d[i][2]$ şi $d[i][1]$, notat min. Se face diferenta între $d[i][2]$ şi $min * d[i][1]$, adică numărul de elemente $d[i][0]$ care mai trebuie înmulţite cu p şi se fac aceste înmulţiri $modulo B$. Acesta este rezultatul căutat.

13.1.2 Rezolvare detaliată

13.1.3 Codul sursă *

```
import java.io.*;
class Factorial
{
  static int n,b,m,p;
  static int[] fb=new int[10];
  static int[] eb=new int[10];
  static int[] en=new int[10];

  static void descf(int nr)
  {
    int d;
    m=0;

    d=2;
    if(nr%d==0) {m++; fb[m]=d; eb[m]=0;}
    while(nr%d==0) {eb[m]++; nr/=d; }

    d=3;
    while(d*d<=nr)
    {
      if(nr%d==0) {m++; fb[m]=d; eb[m]=0;}
      while(nr%d==0) {eb[m]++;nr/=d; }
      d=d+2;
    }
    if(nr!=1) {m++; fb[m]=nr; eb[m]=1;}
  }//descf(...)

  static void calcul()
  {
    int i,ii,j,min;
    descf(b);
    p=1;
    for(j=1;j<=m;j++) en[j]=0;
    for(i=2;i<=n;i++) // n!
    {
      ii=i;
      for(j=1;j<=m;j++) while(ii%fb[j]==0) { en[j]++; ii/=fb[j]; }
```

```
      p=(p*(ii%b))%b;
    }

    min=en[1]/eb[1];
    for(j=2;j<=m;j++) if(en[j]/eb[j]<min) min=en[j]/eb[j];
    for(j=1;j<=m;j++) en[j]=en[j]-min*eb[j];
    for(j=1;j<=m;j++) for(i=1;i<=en[j];i++) p=(p*(fb[j]%b))%b;
  }// calcul()

  public static void main(String[] args) throws IOException
  {
    int k;
    StreamTokenizer st=new StreamTokenizer(
      new BufferedReader(new FileReader("fact.in")));
    PrintWriter out=new PrintWriter(
      new BufferedWriter(new FileWriter("fact.out")));

    for(k=1;k<=5;k++)
    {
      st.nextToken(); n=(int)st.nval;
      st.nextToken(); b=(int)st.nval;
      calcul();
      if(p<=9) out.println(p); else out.println((char)(p-10+'A'));
    }
    out.close();
  }// main(...)
}// class
```

13.2 Limbaj

autor

Definim un limbaj de programare, cu instrucţiuni de atribuire şi de decizie. Sintaxa instrucţiunilor este:

Instrucţiunea de *atribuire*

variabila=variabila

Instrucţiunea de *decizie*

if
semn variabila variabila
{**da**
instrucţiuni }
{ **nu**

instrucţiuni }

fi

Variabilele limbajului sunt identificate printr-un singur caracter, literă mică, din alfabetul englez. Valorile variabilelor sunt de tip întreg.

semn este unul din caracterele $'<'$, $'>'$ sau $'='$.

Liniile instrucţiunilor nu conţin spaţii.

Instrucţiunile din { } pot lipsi.

Cerinţă

Dacă se dau valorile iniţiale ale tuturor variabilelor (de la a la z), în ordine alfabetică începând cu a, se cer valorile tuturor variabilelor de la a la z, după execuţia secvenţei de program.

Date de intrare

Fişier de intrare: **limbaj.in**

Linia 1: V_a V_b ... V_z - Valorile iniţiale ale variabilelor, de la a la z, separate prin câte un spaţiu.

Linia 2: linie de program - Următoarele linii, până la sfârşitul fişierului conţin linii de program,

Linia 3: linie de program cu instrucţiuni corecte din punct de vedere sintactic.

....

Date de ieşire

Fişier de ieşire: **limbaj.out**

Linia 1: V_a V_b ... V_z - şirul valorilor variabilelor, de la a la z, în ordine alfabetică, pe un rând, separate prin câte un spaţiu.

Restricţii şi precizări

- $-30000 < a, b, .., z < 30000$
- Numărul liniilor de program este mai mic decât 10000
- Limbajul este case sensitive (se folosesc doar litere mici de la a la z).

Exemplu

limbaj.in
1 3 5 7 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
if
=ab
nu
if
<ab
da
b=d
fi
f=d
fi

limbaj.out
1 7 5 7 4 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Timp maxim de execuţie/test: 0.5 secunde (Windows), 0.1 secunde (Linux)

13.2.1 Indicaţii de rezolvare - descriere soluţie *

Soluţia comisiei

Soluţia problemei constă în parcurgerea liniară a instrucţiunilor linie cu linie şi păstrarea valorilor logice ale condiţiilor şi respectiv a ramurilor **da** şi **nu** într-un şir.

Semnificaţia poziţiei din şir este nivelul de imbricare la care ne aflăm. Practic parcurgem, fără a evalua, acele instrucţiuni care nu aparţin nivelului şi ramurii pe care ne aflăm.

În acest sens vom evalua instrucţiunile de atribuire doar cele care îndeplinesc condiţiile de a se afla pe nivelul corespunzător şi pentru care valorile condiţiilor şi respectiv a ramurilor sunt în acealşi timp adevarate.

13.2.2 Rezolvare detaliată

13.2.3 Codul sursă *

Variantă cu mesaje:

```
import java.io.*;    // test 25 g,h= gresit la ei!!!
class LimbajMesaje   // test 26 ==> liniile 321 si 400 si 479= "fiif" ... ???
{                    // test 28 ==> multe linii cu "fiif" ... ???
  static int[] x=new int[26];
```

```
static char[] a=new char[4];
static String s;

static BufferedReader brin;
static StreamTokenizer stin;
static BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

static void afisx()
{
  int i;
  for(i=0;i<26;i++) System.out.print(x[i]+" ");
  System.out.println();
}// afisx()

static void afisa()
{
  int i;
  for(i=0;i<a.length;i++) System.out.print(a[i]+" ");
  System.out.println();
}// afisa()

static boolean esteif()
{
  if((a[0]=='i')&&(a[1]=='f')) return true; else return false;
} // esteif()

static boolean estefi()
{
  if((a[0]=='f')&&(a[1]=='i')) return true; else return false;
}// estefi()

static boolean esteda()
{
  if((a[0]=='d')&&(a[1]=='a')) return true; else return false;
}// esteda()

static boolean estenu()
{
  if((a[0]=='n')&&(a[1]=='u')) return true; else return false;
}// estenu()

static void blocda(boolean ok) throws IOException
{
  System.out.println("         --> blocDA:  "+s+" ok="+ok); //br.readLine();
```

```
    bloc(ok);
    System.out.println("         <-- blocDA:  "+s+" ok="+ok); //br.readLine();
  }// blocda(...)

  static void blocnu(boolean ok) throws IOException
  {
    System.out.println("         --> blocNU:  "+s+" ok="+ok); //br.readLine();
    bloc(ok);
    System.out.println("         <-- blocNU:  "+s+" ok="+ok); //br.readLine();
  }// blocnu(...)

  static void blocif(boolean ok) throws IOException
  {
    System.out.println("      --> blocif:  "+s+" ok="+ok); //br.readLine();
    boolean oke=false;

    // urmeaza expresia logica (conditia din IF)
    s=brin.readLine(); a=s.toCharArray();
    System.out.println("evaluare expresie: "+s+" --> oke = "+oke+" ok="+ok);
    //br.readLine();
    afisx();

    if(a[0]=='=' && x[a[1]-'a'] == x[a[2]-'a']) oke=true; else
    if(a[0]=='<' && x[a[1]-'a'] <  x[a[2]-'a']) oke=true; else
    if(a[0]=='>' && x[a[1]-'a'] >  x[a[2]-'a']) oke=true;

    // urmeaza "DA" sau "NU"
    s=brin.readLine(); a=s.toCharArray();

    System.out.println(" ? bloc DA ?   s= "+s);
    if(esteda())
    {
      s=brin.readLine(); a=s.toCharArray();
      blocda(ok && oke);
    }

    System.out.println(" ? bloc NU ?   s= "+s);
    if(estenu())
    {
      s=brin.readLine(); a=s.toCharArray();
      blocnu(ok && !oke);
    }

    s=brin.readLine(); if(s!=null) a=s.toCharArray();
```

```
  System.out.println("      <-- blocif:  "+s+" ok="+ok); //br.readLine();
}// blocif(...)

static void blocatr(boolean ok) throws IOException
{
  System.out.println("         --> blocatr:  "+s+" ok="+ok); //br.readLine();
  if(ok) x[a[0]-'a']=x[a[2]-'a'];

  // mai sunt si alte atribuiri consecutive ...
  s=brin.readLine(); if(s!=null) a=s.toCharArray();
  while((s!=null)&&(a[1]=='='))
  {
    System.out.println("                        "+s+" ok="+ok); //br.readLine();
    if(ok) x[a[0]-'a']=x[a[2]-'a'];
    s=brin.readLine(); if(s!=null) a=s.toCharArray();
  }
  System.out.println("         <-- blocatr:  "+s); //br.readLine();
}// blocatr(...)

static void blocelem(boolean ok) throws IOException
{
  if(estefi()||esteda()||estenu()) return;
  System.out.println("   --> blocelem:  "+s); //br.readLine();

  if(a[1]=='=') blocatr(ok);
  if(esteif()) blocif(ok);

  System.out.println("   <-- blocelem:  "+s); //br.readLine();
}// blocelem(...)

static void bloc(boolean ok) throws IOException // blocElementar + blocElementar +...
{
  System.out.println("--> bloc:  "+s); //br.readLine();
  while(s!=null&&!estefi()&&!esteda()&&!estenu())
  {
    blocelem(ok);
  }
  System.out.println("<-- bloc:  "+s); //br.readLine();
}// bloc(...)

public static void main(String[] args) throws IOException
{
  int k;
  brin=new BufferedReader(new FileReader("limbaj.in"));
```

```
    stin=new StreamTokenizer(brin);
    PrintWriter out=new PrintWriter(
      new BufferedWriter(new FileWriter("limbaj.out")));

    int i;
    for(i=0;i<26;i++) { stin.nextToken(); x[i]=(int)stin.nval; }
    brin.readLine();
    afisx();

    s=brin.readLine();
    if(s!=null) a=s.toCharArray();

    bloc(true);

    for(i=0;i<26;i++) out.print(x[i]+" ");
    out.println();
    out.close();
  }// main(...)
}// class
```

Variantă fără mesaje:

```
import java.io.*; // test 25 g,h= gresit la ei!!!
class Limbaj      // test 26 ==> liniile 321 si 400 si 479= "fiif" ... ???
{                 // test 28 ==> multe linii cu "fiif" ...
  static int[] x=new int[26];
  static char[] a=new char[4];
  static String s;

  static BufferedReader brin;
  static StreamTokenizer stin;

  static boolean esteif()
  {
    if((a[0]=='i')&&(a[1]=='f')) return true; else return false;
  }// esteif()

  static boolean estefi()
  {
    if((a[0]=='f')&&(a[1]=='i')) return true; else return false;
  }// esteif()

  static boolean esteda()
  {
```

```
    if((a[0]=='d')&&(a[1]=='a')) return true; else return false;
  }// esteda()

  static boolean estenu()
  {
    if((a[0]=='n')&&(a[1]=='u')) return true; else return false;
  }// estenu()

  static void blocif(boolean ok) throws IOException
  {
    boolean oke=false;

    // urmeaza expresia logica (conditia din IF)
    s=brin.readLine(); a=s.toCharArray();

    if(a[0]=='=' && x[a[1]-'a'] == x[a[2]-'a']) oke=true; else
    if(a[0]=='<' && x[a[1]-'a'] <  x[a[2]-'a']) oke=true; else
    if(a[0]=='>' && x[a[1]-'a'] >  x[a[2]-'a']) oke=true;

    // urmeaza "DA" sau "NU"
    s=brin.readLine(); a=s.toCharArray();

    if(esteda())
    {
      s=brin.readLine(); a=s.toCharArray();
      bloc(ok && oke);
    }

    if(estenu())
    {
      s=brin.readLine(); a=s.toCharArray();
      bloc(ok && !oke);
    }

    s=brin.readLine(); if(s!=null) a=s.toCharArray();
  }// blocif(...)

  static void blocatr(boolean ok) throws IOException
  {
    if(ok) x[a[0]-'a']=x[a[2]-'a'];

    // mai sunt si alte atribuiri consecutive ...
    s=brin.readLine(); if(s!=null) a=s.toCharArray();
    while((s!=null)&&(a[1]=='='))
```

```
    {
      if(ok) x[a[0]-'a']=x[a[2]-'a'];
      s=brin.readLine(); if(s!=null) a=s.toCharArray();
    }
  }// blocatr(...)

  static void blocelem(boolean ok) throws IOException
  {
    if(estefi()||esteda()||estenu()) return;
    if(a[1]=='=') blocatr(ok);
    if(esteif()) blocif(ok);
  }// blocelem(...)

  static void bloc(boolean ok) throws IOException// blocElementar + blocElementar +...
  {
    while(s!=null&&!estefi()&&!esteda()&&!estenu()) blocelem(ok);
  }// bloc(...)

public static void main(String[] args) throws IOException
{
    int k;
    brin=new BufferedReader(new FileReader("limbaj.in"));
    stin=new StreamTokenizer(brin);
    PrintWriter out=new PrintWriter(
      new BufferedWriter(new FileWriter("limbaj.out")));

    int i;
    for(i=0;i<26;i++) { stin.nextToken(); x[i]=(int)stin.nval; }
    brin.readLine();

    s=brin.readLine();
    if(s!=null) a=s.toCharArray();

    bloc(true);

    for(i=0;i<26;i++) out.print(x[i]+" ");
    out.println();
    out.close();
  }// main(...)
}// class
```

13.3 Panouri

autor

Pe autostrada "Soarele Estului" sunt aşezate de-a lungul şoselei, la distanţe egale, panouri publicitare ale unor firme. Aceeaşi firmă, poate să aibă mai multe panouri publicitare şi fiecare panou poate să apară în mai multe locuri. Panourile se identifică prin numere naturale, numărul total de panouri fiind N.

Firma "X Corporation" are panouri de T tipuri diferite. Firma a primit aprobarea construirii unui mare complex turistic în apropierea autostrăzii; de aceea, pentru alegerea locului, este interesată şi de următorul aspect: care este lungimea minimă de şosea, în care se pot întâlni, toate cele T tipuri de panouri publicitare ale firmei, indiferent de ordinea acestora, şi indiferent dacă între ele se mai interpun sau nu panouri ale altor firme.

Cerinţă

Cunoscând N - numărul total de panouri de la marginea autostrăzii şi ordinea amplasării lor, ca şi cele T tipuri de panouri amplasate de firmă, determinaţi numărul minim de intervale dintre două panouri între care firma "X Corporation" îşi regăseşte toate panourile sale.

Date de intrare

Fişierul de intrare **panouri.in** are pe prima linie numerele N şi T.

Pe următoarele N linii, sunt N numere naturale, nu neaparat diferite, câte unul pe linie, reprezentând panourile, iar începând cu linia $N + 2$, câte unul pe linie, cele T tipuri de panouri diferite al firmei.

Date de ieşire

Fişierul de ieşire **panouri.out** va conţine pe prima linie un singur număr întreg pozitiv L, reprezentând numărul cerut, sau -1 în caz că nu există soluţie.

Restricţii şi precizări

- $1 \leq N \leq 15000$
- $1 \leq T \leq 1000$
- Toate numerele reprezentând panouri sunt numere naturale din intervalul [1..1000].

Exemple

panouri.in	**panouri.out**	Explicaţie
6 2 1 2 3 **5** 3 **1** 5 1	**2**	Sunt $N = 6$ panouri : 1 2 3 5 3 1. Firma are $T = 2$ tipuri de panouri: 5 şi 1. Cel mai scurt interval care conţine elementele **5** şi **1**, este între panourile al 4 - lea şi al 6 -lea, şi conţine 2 intervale.
8 3 5 **1** **3** **3** 5 **4** 2 1 3 1 4	**4**	Sunt $N = 8$ panouri de tipurile: 5 1 3 3 5 4 2 1. Firma are $T = 3$ tipuri de panouri: 3, 1 şi 4. Cel mai scurt interval care conţine elementele 1, 3 şi 4, este între al 2 lea şi al 6-lea panou, şi conţine 4 intervale.

Timp maxim de execuţie/test: 1 secundă (Windows) 0.1 secunde (Linux)

13.3.1 Indicaţii de rezolvare - descriere soluţie *

Soluţia comisiei

Reformulăm problema: fiind dat un şir $a[1..n]$ şi o multime B cu m elemente, să se gasească două poziţii $start$ şi $final$, astfel încât toate elementele mulţimii B să fie conţinute în subsecvenţa $a[start, ..., final]$, cu proprietatea că diferenţa $final - start$ are valoare minimă.

Cu alte cuvinte, să se gasească subsecvenţa de lungime minimă $a[start..final]$ care conţine toate elementele mulţimii B.

Având în vedere valorile mici pentru tipurile de panouri ($1 \leq tip \leq 1000$), pentru operaţia de căutare în $O(1)$ a unui element în multimea B, definim şirul $b[]$, cu $b[i] = 1$ dacă i aparţine mulţimii B.

Definim de asemenea şrul frecvenţelor $fr[]$, cu proprietatea că $fr[i] = x$ daca i aparţine lui B şi i apare de x ori în subsecvenţa $a[start, ..final]$.

Fixăm $start$ şi $final$ la valoarea 1 şi incrementăm poziţia $final$ până când toate elementele mulţimii B se află în intervalul $a[start, ..final]$. Apoi incrementam $start$ până la valoarea maximă la care $a[start, ..final]$ mai conţine încă toate elementele mulţimii B.

În continuare, pentru fiecare incrementare a poziţiei $final$, mărim cât se poate de mult poziţia $start$, cu respectarea restricţiilor. În acest fel ne asigurăm că pentru fiecare poziţie $final$, avem o subsecvenţă de lungime minimă care conţine multimea B.

Algoritmul are complexitatea $O(n)$.

13.3.2 Rezolvare detaliată

13.3.3 Codul sursă *

Variantă care depăşeste timpul pentru ultimele 4 teste:

```
import java.io.*;  // OK ... dar .. depasire timp executie pentru
class Panouri1     // t7=1.07sec  t8=2.5sec  t9=4.9sec  t10=6.6sec
{
  static int n,t,min;
  static int[] x=new int[15001];
  static int[] pf=new int[1001];
  static int[] px=new int[1001];
  static int[] pu=new int[1001];

  static void calcul()
  {
    int i,j,npu,imin,jmin;

    min=15123;
    for(i=1;i<=n-t+1;i++)
    {
      imin=i;
      if(pf[x[i]]==0) continue;

      for(j=1;j<=1000;j++) pu[j]=0;

      npu=1;
      pu[x[i]]=1;
      jmin=-1;
      for(j=i+1;j<=n;j++)
      {
        if( (pf[x[j]]==1) && (pu[x[j]]==0) )
        {
          npu++;
```

```
          pu[x[j]]=1;
          if(npu==t) {jmin=j; break;}
        }
      }
      if(npu==t)
        if((jmin-imin)<min) min=jmin-imin;
    }
  }// calcul()

  public static void main(String[] args) throws IOException
  {
    int i,tipp;
    long t1,t2;
    t1=System.currentTimeMillis();
    StreamTokenizer st=new StreamTokenizer(
      new BufferedReader(new FileReader("panouri.in")));
    PrintWriter out=new PrintWriter(
      new BufferedWriter(new FileWriter("panouri.out")));

    st.nextToken(); n=(int)st.nval;
    st.nextToken(); t=(int)st.nval;
    for(i=1;i<=n;i++) { st.nextToken(); x[i]=(int)st.nval; px[x[i]]=1; }
    for(i=1;i<=t;i++) { st.nextToken(); tipp=(int)st.nval; pf[tipp]=1; }

    min=0;
    for(i=1;i<=1000;i++)
      if((px[i]==0)&&(pf[i]==1)) {min=-1; break;}

    if(min==0)
      if(t>1) calcul();

    out.println(min);
    out.close();
    t2=System.currentTimeMillis();
    System.out.println("Time = "+(t2-t1));
  }// main(...)
}// class
```

Variantă care se încadrează în timp pentru toate testele:

```
import java.io.*;  // OK ... fara depasire timp executie pentru
class Panouri2     // t7=.13sec  t8=0.15sec  t9=0.18sec  t10=0.19sec
{
  static int n,t,min;
```

```
static int[] x=new int[15001];
static int[] pf=new int[1001]; // panou firma
static int[] px=new int[1001];
static int[] pu=new int[1001]; // panou utilizat

static void calcul()
{
  int i,j,npu;

  i=1;
  while((i<=n)&&(pf[x[i]]==0)) i++; // i --> primul panou de firma
  pu[x[i]]++;     // panou utilizat (nr de ori)
  npu=1;          // nr panouri utilizate

  j=i+1;
  while((j<=n-t+1)&&(npu<t))  // ... toate panourile firmei ...
  {
    while(pf[x[j]]==0) j++;
    if(pu[x[j]]==0) npu++;
    pu[x[j]]++;              // panou utilizat (nr de ori)
    if(npu<t) j++;
  }
  min=j-i;                   // intre i ... j sunt toate ...

  while(j<=n)
  {
    while(npu==t)
    {
      while(pf[x[i]]==0) i++; // i -->
      if(pu[x[i]]==1) break; else {pu[x[i]]--; i++;}
    }
    if(j-i<min) min=j-i;

    // il scot pe i ...
    pu[x[i]]=0;
    npu=t-1;
    i++;

    j++; // j --> pana refac nr panouri = t
    while((j<=n)&&(npu<t))
    {
      while(pf[x[j]]==0) j++;  // ma plasez pe panou firma
      if(pu[x[j]]==0) npu++;
      pu[x[j]]++;              // panou utilizat (nr de ori)
```

```
        if(npu<t) j++;
      }
    }
  }// calcul()

  public static void main(String[] args) throws IOException
  {
    int i,tipp;
    long t1,t2;
    t1=System.currentTimeMillis();
    StreamTokenizer st=new StreamTokenizer(
      new BufferedReader(new FileReader("panouri.in")));
    PrintWriter out=new PrintWriter(
      new BufferedWriter(new FileWriter("panouri.out")));

    st.nextToken(); n=(int)st.nval;
    st.nextToken(); t=(int)st.nval;
    for(i=1;i<=n;i++) { st.nextToken(); x[i]=(int)st.nval; px[x[i]]=1; }
    for(i=1;i<=t;i++) { st.nextToken(); tipp=(int)st.nval; pf[tipp]=1; }

    min=0;
    for(i=1;i<=1000;i++)
      if((px[i]==0)&&(pf[i]==1)) {min=-1; break;}

    if(min==0)
      if(t>1) calcul();

    out.println(min);
    out.close();
    t2=System.currentTimeMillis();
    System.out.println("Time = "+(t2-t1));
  }// main(...)
}// class
```

13.4 Pereţi

autor

Localitatea Târgovişte este în plină modernizare. Primăria decide să inventarieze toate clădirile din oraş pentru a renova faţadele acestora. În acest sens analizează harta oraşului şi constată că toţi pereţii sunt aşezaţi doar pe direcţia Nord Sud sau Est Vest. Pereţii vizibili de către turişti sunt doar aceia la care se poate ajunge din exteriorul oraşului prin deplasarea pe cele două direcţii date, în oricare din cele 4 sensuri (N, E, S, V). Harta oraşului este întocmită pe un caroiaj

format din pătrate cu latură 1.

Cerinţă

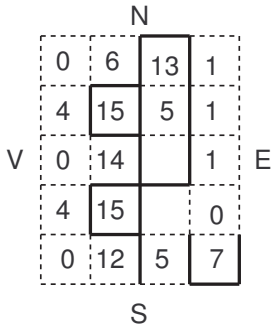
Cunoscându-se harta oraşului, determinaţi lungimea pereţilor vizibili ce urmează a fi zugrăviţi.

Date de intrare

Fişierul de intrare **pereti.in** are pe prima linie dimensiunile m (numărul de linii), n (numărul de coloane) ale hărţii. Pe fiecare dintre următoarele m linii există n numere naturale de la 0 la 15, separate prin câte un spaţiu, cu semnificaţia:

- reprezentarea binară a numărului pe 4 cifre semnifică, începând de la stânga spre dreapta, existenţa unui perete spre direcţiile N, E, S, V (1 - există perete, 0 - nu există perete; explicaţii în figura de mai jos).

De exemplu valoarea 13 se reprezintă în binar 1101, deci în mod corespunzător, de la stânga spre dreapta, vom avea pereţi spre N, E şi V.



Date de ieşire

Fişierul de ieşire **pereti.out** va conţine pe prima linie numărul natural k reprezentând lungimea pereţilor ce vor fi zugrăviţi.

Restricţii şi precizări

- $1 \leq m, n \leq 100$
- Pereţii aflaţi la marginea hărţii sunt pereţi vizibili.
- Datele de intrare sunt considerate corecte.

Exemplu

pereti.in	**pereti.out**	Explicaţie
5 4 0 6 13 1 4 15 5 1 0 14 7 1 4 15 9 0 0 12 5 7	22	Pentru poziţiile (5, 2) şi (5, 3) peretele dintre ele va fi zugrăvit pe ambele feţe. Peretele dinspre Nord al poziţiei (1,3) este perete exterior, chiar dacă se află pe marginea hărţii.

Timp maxim de execuţie/test: 1 secundă (Windows), 0.5 secunde (Linux)

13.4.1 Indicaţii de rezolvare - descriere soluţie *

Soluţia comisiei

Se bordează matricea cu valorile corespunzătoare, astfel încât toată harta se înconjoră cu ziduri exterioare. Se determină toate poziţiile (celulele) din matrice accesibile din pozitia $(0, 0)$ care este exterioară astfel:

Se construieşte un şir de perechi, (i, j) care vor marca pe rând poziţiile din matrice pe care le-am parcurs deja.

Se porneşte din $(0, 0)$, se determină toate poziţiile din matrice vecine cu acestea, în care se poate ajunge, şi se adaugă la şirul construit. Odată adăugate, se trece la următoarea poziţie din şir şi se adaugă din nou toţi vecinii accesibili din aceasta dar care nu figurează în şirul nostru. Procedura continuă până când marcăm toţi vecinii celulelor adăugate în şir.

Pentru a marca elementele din matrice deja adăugate în sir adunăm 16 fiecărei "celule". În acelaşi timp adăugăm numărul de pereţi vizibili din exterior, respectiv 1, 2 sau 3 în funcţie de valoarea celulei.

Verificarea vecinilor se realizează fără a folosi descompunerea binară a valorilor. Se foloseşte operaţia AND între valorile numerice întregi (de exemplu, 8 and 11 = 8 deci există perete spre N).

13.4.2 Rezolvare detaliată

13.4.3 Codul sursă *

```
import java.io.*;
class Pereti
{
  static final int nord=8, est=4, sud=2, vest=1;
  static int m,n,np,nz;
  static int[][] a=new int[101][101];
```

```
  static int[][] b=new int[101][101];

  static void fill(int i, int j)
  {
    b[i][j]=nz;

    if((a[i][j] & nord)!=0) np++;
    if((a[i][j] & est) !=0) np++;
    if((a[i][j] & sud) !=0) np++;
    if((a[i][j] & vest)!=0) np++;

    if((i-1>=1)&&(b[i-1][j]==0)&&((a[i][j]&nord)==0)) fill(i-1,j);// nord
    if((i+1<=m)&&(b[i+1][j]==0)&&((a[i][j]&sud)==0))  fill(i+1,j);// sud
    if((j-1>=1)&&(b[i][j-1]==0)&&((a[i][j]&vest)==0)) fill(i,j-1);// vest
    if((j+1<=n)&&(b[i][j+1]==0)&&((a[i][j]&est)==0))  fill(i,j+1);// est
  }// fill(...)

  static void intra()
  {
    int i,j;
    for(i=1;i<=m;i++) // de la vest
      if((b[i][1]==0)&&((a[i][1]&vest)==0)) { nz++; fill(i,1); }
    for(i=1;i<=m;i++) // de la est
      if((b[i][n]==0)&&((a[i][n]&est)==0)) { nz++; fill(i,n); }
    for(j=1;j<=n;j++) // de la nord
      if((b[1][j]==0)&&((a[1][j]&nord)==0)) { nz++; fill(1,j); }
    for(j=1;j<=n;j++) // de la sud
      if((b[m][j]==0)&&((a[m][j]&sud)==0)) { nz++; fill(m,j); }
  }// intra(...)

  static void exterior()
  {
    int i,j;
    for(i=1;i<=m;i++) if((a[i][1]&vest)!=0) np++; // vest
    for(i=1;i<=m;i++) if((a[i][n]&est) !=0) np++; // est
    for(j=1;j<=n;j++) if((a[1][j]&nord)!=0) np++; // nord
    for(j=1;j<=n;j++) if((a[m][j]&sud) !=0) np++; // sud
  }// exterior(...)

  public static void main(String[] args) throws IOException
  {
    int i,j;
    StreamTokenizer st=new StreamTokenizer(
      new BufferedReader(new FileReader("pereti.in")));
```

```
    PrintWriter out=new PrintWriter(
      new BufferedWriter(new FileWriter("pereti.out")));
    st.nextToken(); m=(int)st.nval;
    st.nextToken(); n=(int)st.nval;
    for(i=1;i<=m;i++)
      for(j=1;j<=n;j++) { st.nextToken(); a[i][j]=(int)st.nval; }
    np=0; // nr pereti
    nz=0; // nr zone (fill!)
    intra();
    exterior();
    out.println(np);
    out.close();
  }// main(...)
}// class
```

13.5 Şanţ

autor

Cei n deţinuţi ai unei închisori, numerotaţi de la 1 la n, trebuie să sape un şanţ dispus în linie dreaptă între două puncte A şi B, situate la distanţa de 250 km unul de celălalt, pe care există 251 borne kilometrice numerotate de la 0 la 250. Fiecare deţinut are însă o pretenţie, "e doar democraţie, nu?": el doreşte să sape doar undeva în zona dintre borna x şi borna y. Pe lângă aceste pretenţii închisoarea se confruntă şi cu o altă problemă: nu are suficienţi paznici angajaţi.

Cerinţă

Cunoscându-se numărul n de deţinuţi şi pretenţiile lor, să se determine locul (locurile) unde vor fi puşi deţinuţii să sape într-o zi de muncă, respectându-se pretenţiile lor, astfel încât numărul de paznici necesari pentru a păzi cei n deţinuţi să fie minim. Intervalul în care poate păzi un paznic nu poate conţine două sau mai multe zone disjuncte dintre cele exprimate de deţinuţi în preferinţele lor.

Date de intrare

Fişierul de intrare **sant.in** are pe prima linie numărul n de deţinuţi. Pe fiecare dintre următoarele n linii există câte două numere naturale, a_i b_i, separate printr-un spaţiu ($a_i \leq b_i$), care reprezintă pretenţia deţinutului. Mai exact pe linia $i+1$ ($1 \leq i \leq n$) este descrisă pretenţia deţinutului cu numărul de ordine i.

Date de ieşire

Fişierul de ieşire **sant.out** va conţine pe prima linie numărul natural k reprezentând numărul minim de paznici necesari pentru paza celor n deţinuţi scoşi la lucru. Pe următoarele $2k$ linii vor fi descrise locurile unde vor fi puşi să sape deţinuţii, astfel: fiecare pereche de linii $(2j, 2j+1)$ va conţine pe linia $2j$ trei numere naturale p x_j y_j, separate prin câte un spaţiu reprezentând numărul de ordine al paznicului şi bornele kilometrice x_j c si y_j unde va păzi paznicul p, iar pe linia

$2j + 1$ vor fi scrise numerele de ordine ale deţinuţilor care sapă în această zonă, separate prin câte un spaţiu, ordonate crescător.

Restricţii şi precizări

- $1 \leq n \leq 10000$
- $0 \leq a_i \leq b_i \leq 250$, pentru orice i, $1 \leq i \leq n$
- $0 \leq x_j \leq y_j \leq 250$, pentru orice j, $1 \leq j \leq k$
- un deţinut poate să sape şi într-un singur punct ("în dreptul bornei kilometrice numerotată cu x")
- în cazul în care există mai multe soluţii se va afişa una singură
- numerele de ordine ale paznicilor vor fi scrise în fişier în ordine crescătoare
- numerotarea paznicilor începe de la 1

Exemplu

.in	**.out**	Explicaţie
3 0 20 8 13 30 60	2 1 8 13 1 2 2 30 60 3	sunt necesari 2 paznici: paznicul 1 va păzi între borna 8 şi borna 13, iar deţinuţii păziţi sunt 1 şi 2; paznicul 2 va păzi între borna 30 şi borna 60, iar deţinutul păzit este 3
4 10 203 2 53 30 403 5 7 33	3 1 10 20 1 2 5 5 2 4 3 30 40 3	sunt necesari 3 paznici: paznicul 1 va păzi între borna 10 şi borna 20, iar deţinutul păzit este 1; paznicul 2 va păzi la borna 5, iar deţinuţii păziţi sunt 2 şi 4; paznicul 3 va păzi între borna 30 şi borna 40, iar deţinutul păzit este 3
5 10 30 30 32 0 30 27 30 27 28	2 1 30 30 1 2 3 4 2 27 28 5	sunt necesari 2 paznici: paznicul 1 va păzi la borna 30, iar deţinuţii păziţi sunt 1, 2, 3 şi 4; paznicul 2 va păzi între borna 27 şi borna 28, iar deţinutul păzit este 5 *!Soluţia nu este unică!*

Timp maxim de execuţie/test: 1 secundă (Windows), 0.5 secunde (Linux)

13.5.1 Indicaţii de rezolvare - descriere soluţie *

Soluţia comisiei

Problema cere, de fapt, determinarea numărului minim de intersecţii între segmentele determinate de kilometrul minim şi maxim între care sapă un deţinut.

Pentru a le determina procedez astfel:

- ordonez intervalele crescător după kilometrul minim şi descrescător după kilometrul maxim
- pun primul deţinut (deci cel cu intervalul de săpare cel mai mare) în grija primului paznic
- pentru toate celelalte
  - caut un paznic care mai păzeşte deţinuţi şi care poate păzi şi acest deţinut (adică intersecţia segmentelor să fie nevidă)
  - dacă găsesc
  - ajustez intervalul de săpare ca să poată săpa şi acest deţinut
  - altfel (dacă nu găsesc)
  - mai am nevoie de un paznic şi îl dau în grija acestuia

Datorită faptului că intervalele sunt sortate, crescător după capătul inferior, în momentul când un interval nu se mai intersectează cu cel deja determinat, este sigur că nici un alt interval ce îl urmează nu se mai intersectează, lucru ce asigură determinarea numărului minim de paznici.

13.5.2 Rezolvare detaliată

13.5.3 Codul sursă *

```
import java.io.*;
class Sant
{
  static int n;
  static int[] x1=new int[10001];
  static int[] x2=new int[10001];
  static int[]  o=new int[10001];

  static void qsort(int li,int ls)
  {
    int val,aux,i,j;

    i=li;
```

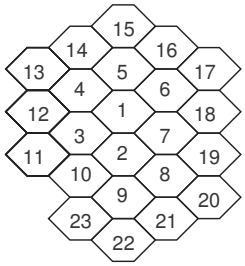
```
    j=ls;
    val=x2[(i+j)/2];

    while(i<j)
    {
      while(x2[i]<val) i++;
      while(x2[j]>val) j--;
      if(i<=j)
      {
        aux=x1[i]; x1[i]=x1[j]; x1[j]=aux;
        aux=x2[i]; x2[i]=x2[j]; x2[j]=aux;
        aux=o[i];  o[i]=o[j];    o[j]=aux;
        i++;
        j--;
      }
    }// while

    if(li<j) qsort(li,j);
    if(i<ls) qsort(i,ls);
  }// qsort(...)

  public static void main(String[] args) throws IOException
  {
    int k,np,xp;
    StreamTokenizer st=new StreamTokenizer(
      new BufferedReader(new FileReader("sant.in")));
    PrintWriter out=new PrintWriter(
      new BufferedWriter(new FileWriter("sant.out")));

    st.nextToken(); n=(int)st.nval;
    for(k=1;k<=n;k++)
    {
      st.nextToken(); x1[k]=(int)st.nval;
      st.nextToken(); x2[k]=(int)st.nval;
      o[k]=k;
    }

    qsort(1,n);

    np=1;
    xp=x2[1];
    for(k=2;k<=n;k++) if(x1[k]>xp) { np++; xp=x2[k]; }
    out.println(np);
```

```
    // inca o data pentru ...!
    np=1;
    xp=x2[1];
    out.println(np+" "+xp+" "+xp);
    out.print(o[1]+" ");
    for(k=2;k<=n;k++)
    {
      if(x1[k]>xp)
      {
        out.println();
        np++;
        xp=x2[k];
        out.println(np+" "+xp+" "+xp);
        out.print(o[k]+" ");
      }
      else out.print(o[k]+" ");
    }// for k
    out.close();
  }// main(...)
}// class
```

13.6 Zumzi

autor

Albinuţa **zumzi** locuieşte într-un stup format din N celule de formă hexagonală. Cele N celule numerotate de la 1 la N sunt dispuse sub formă de spirală (ca în figură).



Adică, celula din centrul stupului este numerotată cu 1. Plecând de la această celulă spre sud şi apoi în spirală, în sensul acelor de ceasornic, sunt numerotate celelalte celule.

Iniţial **zumzi** se găseşte în celula din centru (cea numerotată cu 1), şi doreşte să ajungă, trecând din celulă în celulă, la celula cu numărul de ordine X, unde se

găseşte prietenul ei. **zumzi** se poate deplasa dintr-o celulă în oricare dintre celulele vecine, fără a părăsi însă stupul.

Două celule sunt vecine dacă au o latură comună.

Unele celule ale stupului sunt ocupate de alte albine şi de aceea **zumzi** nu poate să treacă prin ele.

Cerinţă

Problema vă cere să determinaţi câte variante are **zumzi** ca după **exact** K paşi să ajungă la prietenul ei.

Date de intrare

Fişierul de intrare **zumzi.in** conţine pe prima sa linie valorile naturale N, M, K şi X separate printr-un spaţiu, având următoarea semnificaţie:

- N - numărul total de celule din stup;
- M - numărul de celule din stup ocupate de alte albine
- K - numărul de paşi pe care îi are la dispoziţie **zumzi**
- X - numărul de ordine a celulei în care se găseşte prietenul lui **zumzi**.

Următoarea linie a fişierului de intrare conţine M numere naturale separate printr-un spaţiu reprezentând numerele de ordine ale celulelor ocupate din stup.

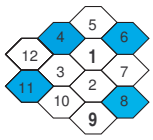
Date de ieşire

Fişierul text **zumzi.out** va conţine pe prima sa linie un singur număr natural reprezentând numărul de variante pe care le are **zumzi** la dispoziţie de a ajunge la prietenul ei.

Restricţii şi precizări

- $1 \leq M < N \leq 300$
- $X \neq 1$
- $K \leq 100$
- **zumzi** nu are posibilitatea de a părăsi stupul, iar în plus odată ajunsă la prietenul ei nu îl va mai părăsi.
- **zumzi** nu este o albină foarte inteligentă de aceea ea poate trece de mai multe ori printr-o celulă, cu excepţia celulei finale, în care se află prietenul ei, celulă în care va intra o singură dată şi nu o mai părăseşte.

Exemplu

zumzi.in	**zumzi.out**	Explicaţie
12 4 3 9 11 4 6 8	4	Variantele sunt: 1-2-10-9 1-3-2-9 1-3-10-9 1-7-2-9 
12 4 4 2 11 4 6 8	9	Variantele avute la dispoziţie sunt: 1-3-10-9-2 1-7-1-3-2 1-5-1-7-2 etc.

Timp maxim de execuţie/test: 1 secundă

13.6.1 Indicaţii de rezolvare - descriere soluţie *

Soluţia comisiei

Pentru memorarea "stupului" vom folosi o matrice pătratică T de ordinul $2 * k + 1$, unde valoarea lui k este data de relaţia:

$$3 * (k-1) * k + 1 < n \leq 3 * k * (k+1) + 1$$

adică k reprezintă numărul de cercuri concentrice din stup (fără a număra celula 1 ca un cerc).

$T[k+1, k+1] = 1$ adică reprezintă celula din centrul stupului.

Celulele vecine ale celulei de la coordonatele (i, j) din matricea T vor fi, în ordine:

$$(i+1, j), (i+1, j-1), (i, j-1), (i-1, j), (i-1, j+1), (i, j+1).$$

De exemplu pentru $N = 12$ matricea T va arăta astfel:

```
 0  0 0 0 0
 0  0 5 6 0
 0  4 1 7 0
12  3 2 8 0
11 10 9 0 0
```

iar pentru $N = 35$:

```
 0  0  0 31 32 33 34
 0  0 30 15 16 17 35
 0 29 14  5  6 18  0
28 13  4  1  7 19  0
```

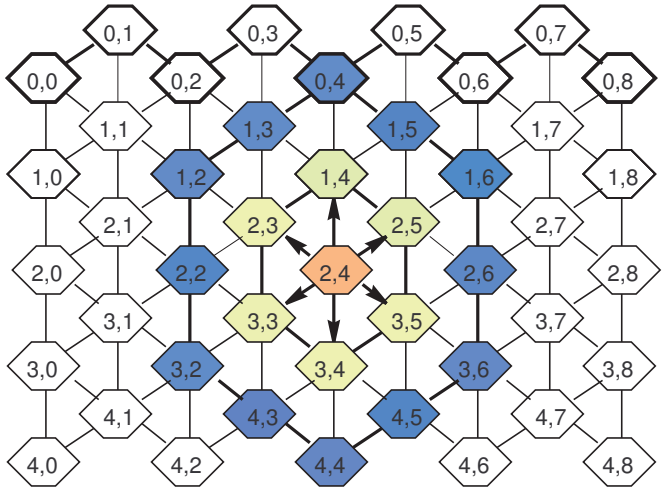
```
27 12  3  2  8 20  0
26 11 10  9 21  0  0
25 24 23 22  0  0  0
```

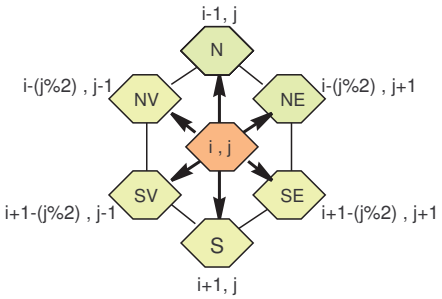
Vom mai folosi un vector A cu M componente, în care $A[i]$ este 0 dacă celula cu numărul de ordine i este liberă, şi respectiv 1 dacă celula este ocupată.

Se vor mai folosi doi vectori $p1$ şi $p2$ cu semnificaţia că la pasul j, $p1[i]$ reprezintă numărul de drumuri de lungime $j-1$ existente până la celula i, iar $p2[i]$ reprezintă numărul de drumuri de lungime j existente până la celula i. Se observă că $p2[i]$ va fi suma valorilor $p1[k]$ pentru toate celulele k vecine cu i.

Întrucât valorile solicitate pot fi foarte mari se va lucra pe numere mari.

13.6.2 Rezolvare detaliată





13.6.3 Codul sursă *

Variantă fără numere mari:

```
import java.io.*;
class Zumzi1
{
  static int n,m,np,x;   // np=nr pasi
  static int nv=0,nc=0;

  static int[] a=new int[301];
  static int[][] s=new int[23][23];  // stup
  static int[] is=new int[301];      // pozitie in stup: linia
  static int[] js=new int[301];      // pozitie in stup: coloana

  static int[][] nd=new int[301][2]; // nr drumuri

  static int i0=11, j0=11;           // centrul stupului

  static int in(int i,int j) { return i-1; } // i nord
  static int jn(int i,int j) { return j;   } // j nord
  static int is(int i,int j) { return i+1; } // i sud
  static int js(int i,int j) { return j;   } // j sud

  static int ine(int i,int j) { return i-j%2;   } // i nord_est
  static int jne(int i,int j) { return j+1;     } // j nord_est
  static int ise(int i,int j) { return i+1-j%2; } // i sud_est
  static int jse(int i,int j) { return j+1;     } // j sud_est

  static int inv(int i,int j) { return i-j%2;   } // i nord_vest
  static int jnv(int i,int j) { return j-1;     } // j nord_vest
  static int isv(int i,int j) { return i+1-j%2; } // i sud_vest
  static int jsv(int i,int j) { return j-1;     } // j sud_vest

  static void stupul()
  {
    int i,j,ic,jc,np,k; // k=nr celule
    k=1;
    s[i0][j0]=1; is[1]=i0; js[1]=j0; // zumzi (linie, coloana)
    i=i0;
    j=j0;
    while(k<n)
    {
      nc++;
```

```
// la sud 1 pas
ic=i; jc=j;
i=is(ic,jc); j=js(ic,jc);
s[i][j]=++k; is[k]=i; js[k]=j;
if(k==n) break;

// sud_vest nc-1 pasi
for(np=1;np<=nc-1&&k<n;np++)
{
  ic=i; jc=j;
  i=isv(ic,jc); j=jsv(ic,jc);
  s[i][j]=++k; is[k]=i; js[k]=j;
}
if(k==n) break;

// nord_vest nc pasi
for(np=1;np<=nc&&k<n;np++)
{
  ic=i; jc=j;
  i=inv(ic,jc); j=jnv(ic,jc);
  s[i][j]=++k; is[k]=i; js[k]=j;
}
if(k==n) break;

// nord nc pasi
for(np=1;np<=nc&&k<n;np++)
{
  ic=i; jc=j;
  i=in(ic,jc); j=jn(ic,jc);
  s[i][j]=++k; is[k]=i; js[k]=j;
}
if(k==n) break;

// nord_est nc pasi
for(np=1;np<=nc&&k<n;np++)
{
  ic=i; jc=j;
  i=ine(ic,jc); j=jne(ic,jc);
  s[i][j]=++k; is[k]=i; js[k]=j;
}
if(k==n) break;

// sud_est nc pasi
for(np=1;np<=nc&&k<n;np++)
```

```
    {
      ic=i; jc=j;
      i=ise(ic,jc); j=jse(ic,jc);
      s[i][j]=++k; is[k]=i; js[k]=j;
    }
    if(k==n) break;

    // sud nc pasi
    for(np=1;np<=nc&&k<n;np++)
    {
      ic=i; jc=j;
      i=is(ic,jc); j=js(ic,jc);
      s[i][j]=++k; is[k]=i; js[k]=j;
    }
  }// while
}// stupul()

static void calcul()
{
  int i,k,v;               // v=vecin
  nd[1][0]=1;              // zumzi
  for(k=1;k<=np;k++)       // pasul k
  {
    for(i=1;i<=n;i++)      // celula i
    {
      nd[i][k%2]=0;        // este suma vecinilor

      if(a[i]==1) continue;         // este ocupata
      if((i==x)&&(k<np)) continue;  // nu mai pleaca !

      v=s[in(is[i],js[i])][jn(is[i],js[i])];  // vecin la nord
      if(a[v]==0) nd[i][k%2]+=nd[v][(k+1)%2]; // celula libera

      v=s[is(is[i],js[i])][js(is[i],js[i])];  // vecin la sud
      if(a[v]==0) nd[i][k%2]+=nd[v][(k+1)%2]; // celula libera

      v=s[ine(is[i],js[i])][jne(is[i],js[i])];  // vecin la nord_est
      if(a[v]==0) nd[i][k%2]+=nd[v][(k+1)%2];   // celula libera

      v=s[ise(is[i],js[i])][jse(is[i],js[i])];  // vecin la sud_est
      if(a[v]==0) nd[i][k%2]+=nd[v][(k+1)%2];   // celula libera

      v=s[inv(is[i],js[i])][jnv(is[i],js[i])];  // vecin la nord_vest
      if(a[v]==0) nd[i][k%2]+=nd[v][(k+1)%2];   // celula libera
```

```
        v=s[isv(is[i],js[i])][jsv(is[i],js[i])];  // vecin la sud_vest
        if(a[v]==0) nd[i][k%2]+=nd[v][(k+1)%2];   // celula libera
      }// for i
    }// for k
  }// calcul()

  public static void main(String[] args) throws IOException
  {
    int i,j;
    StreamTokenizer st=new StreamTokenizer(
      new BufferedReader(new FileReader("zumzi.in")));
    PrintWriter out=new PrintWriter(
      new BufferedWriter(new FileWriter("zumzi.out")));
    st.nextToken(); n=(int)st.nval;
    st.nextToken(); m=(int)st.nval;
    st.nextToken(); np=(int)st.nval;
    st.nextToken(); x=(int)st.nval;

    for(i=1;i<=m;i++)  { st.nextToken(); j=(int)st.nval; a[j]=1; }

    stupul();       // numerotare celule
    calcul();

    out.println(nd[x][np%2]);
    out.close();
  }// main(...)
}// class
```

Variantă cu numere mari:

```
import java.io.*; // test2 = ? aici = 186979673890425070366353371 40
class Zumzi2      // in 2-zumzi.ok  = 187099299808834866109433599 69
{
  static int n,m,np,xz; // np=nr pasi
  static int nv=0,nc=0;

  static int[] a=new int[301];
  static int[][] s=new int[23][23]; // stup
  static int[] is=new int[301];     // pozitie in stup: linia
  static int[] js=new int[301];     // pozitie in stup: coloana

  static int[][][] nd=new int[301][2][1];  // nr drumuri
```

```
  static int i0=11, j0=11;                // centrul stupului

  static int in(int i,int j) { return i-1; } // i nord
  static int jn(int i,int j) { return j;   } // j nord
  static int is(int i,int j) { return i+1; } // i sud
  static int js(int i,int j) { return j;   } // j sud

  static int ine(int i,int j) { return i-j%2;   } // i nord_est
  static int jne(int i,int j) { return j+1;     } // j nord_est
  static int ise(int i,int j) { return i+1-j%2; } // i sud_est
  static int jse(int i,int j) { return j+1;     } // j sud_est

  static int inv(int i,int j) { return i-j%2;   } // i nord_vest
  static int jnv(int i,int j) { return j-1;     } // j nord_vest
  static int isv(int i,int j) { return i+1-j%2; } // i sud_vest
  static int jsv(int i,int j) { return j-1;     } // j sud_vest

  static void stupul()
  {
    int i,j,ic,jc,np,k; // k=nr celule
    k=1;
    s[i0][j0]=1; is[1]=i0; js[1]=j0; // zumzi (linie, coloana)
    i=i0;
    j=j0;
    while(k<n)
    {
      nc++;
      // la sud 1 pas
      ic=i; jc=j;
      i=is(ic,jc); j=js(ic,jc);
      s[i][j]=++k; is[k]=i; js[k]=j;
      if(k==n) break;

      // sud_vest nc-1 pasi
      for(np=1;np<=nc-1&&k<n;np++)
      {
        ic=i; jc=j;
        i=isv(ic,jc); j=jsv(ic,jc);
        s[i][j]=++k; is[k]=i; js[k]=j;
      }
      if(k==n) break;

      // nord_vest nc pasi
      for(np=1;np<=nc&&k<n;np++)
```

```
      {
        ic=i; jc=j;
        i=inv(ic,jc); j=jnv(ic,jc);
        s[i][j]=++k; is[k]=i; js[k]=j;
      }
      if(k==n) break;

      // nord nc pasi
      for(np=1;np<=nc&&k<n;np++)
      {
        ic=i; jc=j;
        i=in(ic,jc); j=jn(ic,jc);
        s[i][j]=++k; is[k]=i; js[k]=j;
      }
      if(k==n) break;

      // nord_est nc pasi
      for(np=1;np<=nc&&k<n;np++)
      {
        ic=i; jc=j;
        i=ine(ic,jc); j=jne(ic,jc);
        s[i][j]=++k; is[k]=i; js[k]=j;
      }
      if(k==n) break;

      // sud_est nc pasi
      for(np=1;np<=nc&&k<n;np++)
      {
        ic=i; jc=j;
        i=ise(ic,jc); j=jse(ic,jc);
        s[i][j]=++k; is[k]=i; js[k]=j;
      }
      if(k==n) break;

      // sud nc pasi
      for(np=1;np<=nc&&k<n;np++)
      {
        ic=i; jc=j;
        i=is(ic,jc); j=js(ic,jc);
        s[i][j]=++k; is[k]=i; js[k]=j;
      }
    }// while
  }// stupul()
```

```
  static int[] nrv(int nr)
  {
    int[] x;
    if(nr==0) { x=new int[1]; x[0]=0; }
    else
    {
      int nc, nrrez=nr;
      nc=0;
      while(nr!=0) {nc++; nr=nr/10;}

      x=new int[nc];
      nr=nrrez;
      nc=0;
      while(nr!=0) { x[nc]=nr%10; nc++; nr=nr/10;}
    }
    return x;
  }// nrv(...)

  static int[] suma(int[] x, int[] y)
  {
    int[] z=new int[(x.length>y.length) ? (x.length+1) : (y.length+1)];
    int k,t=0;
    for(k=0;k<=z.length-2;k++)
    {
      z[k]=t;
      if(k<x.length) z[k]+=x[k];
      if(k<y.length) z[k]+=y[k];
      t=z[k]/10;
      z[k]=z[k]%10;
    }
    z[z.length-1]=t;
    if(z[z.length-1]!=0) return z;
    else
    {
      int[] zz=new int[z.length-1];
      for(k=0;k<zz.length;k++) zz[k]=z[k];
      return zz;
    }
  }// suma(...)

  static void calcul()
  {
    int i,k,v;              // v=vecin
    nd[1][0]=nrv(1);        // zumzi
```

```
    for(k=1;k<=np;k++)      // pasul k
    {
      for(i=1;i<=n;i++)     // celula i
      {
        nd[i][k%2]=nrv(0);  // este suma vecinilor

        if(a[i]==1) continue;          // este ocupata
        if((i==xz)&&(k<np)) continue;  // nu mai pleaca !

        v=s[in(is[i],js[i])][jn(is[i],js[i])];      // vecin la nord
        if(a[v]==0) nd[i][k%2]=suma(nd[i][k%2],nd[v][(k+1)%2]);

        v=s[is(is[i],js[i])][js(is[i],js[i])];      // vecin la sud
        if(a[v]==0) nd[i][k%2]=suma(nd[i][k%2],nd[v][(k+1)%2]);

        v=s[ine(is[i],js[i])][jne(is[i],js[i])];     // vecin la nord_est
        if(a[v]==0) nd[i][k%2]=suma(nd[i][k%2],nd[v][(k+1)%2]);

        v=s[ise(is[i],js[i])][jse(is[i],js[i])];     // vecin la sud_est
        if(a[v]==0) nd[i][k%2]=suma(nd[i][k%2],nd[v][(k+1)%2]);

        v=s[inv(is[i],js[i])][jnv(is[i],js[i])];     // vecin la nord_vest
        if(a[v]==0) nd[i][k%2]=suma(nd[i][k%2],nd[v][(k+1)%2]);

        v=s[isv(is[i],js[i])][jsv(is[i],js[i])];     // vecin la sud_vest
        if(a[v]==0) nd[i][k%2]=suma(nd[i][k%2],nd[v][(k+1)%2]);
      }// for i
    }// for k
  }// calcul()

  public static void main(String[] args) throws IOException
  {
    int i,j;
    StreamTokenizer st=new StreamTokenizer(
      new BufferedReader(new FileReader("zumzi.in")));
    PrintWriter out=new PrintWriter(
      new BufferedWriter(new FileWriter("zumzi.out")));
    st.nextToken(); n=(int)st.nval;
    st.nextToken(); m=(int)st.nval;
    st.nextToken(); np=(int)st.nval;
    st.nextToken(); xz=(int)st.nval;

    for(i=1;i<=m;i++)  { st.nextToken(); j=(int)st.nval; a[j]=1; }
```

```
    stupul();          // numerotare celule
    calcul();

    for(i=nd[xz][np%2].length-1;i>=0;i--) out.print(nd[xz][np%2][i]);
    out.println();
    out.close();
  }// main(...)
}// class
```

Capitolul 14

ONI 2007 clasa a IX-a

14.1 Agitaţie - ONI 2007

O firmă producătoare de software organizează un interviu pentru ocuparea unui post de programator, la care s-au prezentat N candidaţi. Aceştia sunt ordonaţi în funcţiie de momentul la care şi-au trimis CV-ul şi numerotaţi cu numere consecutive de la 1 la N.

Fiecărui candidat i-a fost realizat în prealabil un profil psihologic şi i s-a determinat nivelul de agitaţie provocat de interviul care urmează să aibă loc, precum şi un sens (crescător sau descrescător) de modificare a acestui nivel. Astfel, la ora la care s-a anunţat începerea interviului (pe care o vom considera momentul 0), fiecare candidat are un nivel de agitaţie iniţial. Pentru fiecare unitate de timp după momentul 0 şi până în momentul în care candidatul este invitat pentru interviu (până atunci el trebuind să aştepte), nivelul său de agitaţie se modifică cu 1: pentru o parte din candidaţi nivelul creşte cu 1 unitate, iar pentru ceilalţi nivelul scade cu 1 unitate. Dacă nivelul de agitaţie a unui candidat ajunge la 0,

din acel moment, pentru fiecare unitate de timp următoare, nivelul de agitaţie va creşte cu 1 (se schimbă sensul de modificare a nivelului de agitaţie).

Firma va invita candidaţii la interviu în grupuri, în ordinea numerotării (toţi candidaţii având numere de ordine mai mici decât un candidat K vor fi invitaţi într-un grup anterior sau în acelaşi grup cu candidatul K). Înainte de a invita un grup, comisia ce conduce interviul poate decide să aştepte un număr întreg de unităţi de timp (zero sau mai multe). Pentru un grup, durata interviului se consideră neglijabilă (fiecare candidat trebuie doar să răspundă la câteva întrebări de tip grilă). Din momentul în care un candidat este invitat la interviu, nivelul de agitaţie a acestuia rămâne constant.

Deoarece firma doreşte ca, indiferent de rezultatul interviului, toţi candidaţii să rămână cu o părere bună, comisia va forma grupurile şi va alege timpii de aşteptare în aşa fel încât suma totală a nivelelor de agitaţie a candidaţilor la sfârşitul interviului să fie minimă.

Cerinţă

Să se scrie un program care să determine suma totală minimă a nivelelor de agitaţie a candidaţilor la sfârşitul interviului.

Date de intrare

Fişierul de intrare **agitatie.in** are pe prima linie numărul natural N, reprezentând numărul de candidaţi. Pe următoarele N linii se află câte două numere întregi A şi B, separate printr-un spaţiu. A reprezintă nivelul iniţial de agitaţie a candidatului, iar B reprezintă sensul de modificare a agitaţiei pentru fiecare unitate de timp în care acesta aşteaptă (dacă B este 1, atunci nivelul de agitaţie creşte, iar dacă B este -1, nivelul de agitaţie scade). Linia $k+1$ din fişier va conţine valorile corespunzătoare candidatului cu numărul k.

Date de ieşire

Fişierul de ieşire **agitatie.out** va conţine pe prima (şi singura) linie suma totală minimă a nivelelor de agitaţie a candidaţilor la sfârşitul interviului.

Restricţii şi precizări

- $1 \leq N \leq 3000$
- $1 \leq$ nivelul iniţial de agitaţie a fiecărui candidat ≤ 3000

Exemplu

agitatie.in	agitatie.out
6 10 1 3 -1 2 -1 1 -1 9 1 6 -1	23

Explicaţie:

Suma totală minimă este 23.

O posibilă soluţie este următoarea: Se formează 3 grupuri.

Primul grup este format doar din candidatul 1 şi aşteapt 0 unităţi de timp. Prin urmare, nivelul final de agitaţie a candidatului 1 va fi 10.

Al doilea grup va aştepta 2 unităţi de timp din momentul în care a terminat interviul primul grup (deci va începe interviul la momentul 2), iar din grup vor face parte candidaţii 2, 3, 4 şi 5. Nivelele finale de agitaţie a acestor candidaţi vor fi: 1, 0, 1 şi 11. Observaţi că nivelul de agitaţie a candidatului 4 a scăzut întâi până la 0, apoi a crescut la 1.

Al 3-lea grup va mai aştepta 4 unităţi de timp (deci va începe interviul la momentul 6), iar din grup va face parte doar candidatul 6. Nivelul final de agitaţie a acestuia va fi 0.

Timp maxim de execuţie/test(Windows/Linux): 0.6 secunde

14.1.1 Indicaţii de rezolvare - descriere soluţie *

Soluţia 1 (As. Mugurel Ionuţ Andreica)

Se va calcula o matrice $SMIN[i,t]$, reprezentând suma minimă a nivelelor finale de agitaţie a candidaţilor $i, i+1, .., N$, dacă interviul primilor $i-1$ candidaţi s-ar termina la momentul t. Pentru calculul lui $SMIN[i,t]$ avem 2 variante:

• să nu aşteptăm deloc înainte de a începe grupul ce-l conţine pe candidatul i (timp de aşteptare egal cu 0 implică apartenenţa la grupul anterior); deci, $SMIN[i,t]$ ar putea fi egal cu: (nivelul de agitatie al candidatului i la momentul t) $+SMIN[i+1,t]$

• se aşteaptă cel puţin 1 unitate de timp (se începe un grup nou de la candidatul i încolo) deci, $SMIN[i,t]$ ar putea fi egal cu $SMIN[i,t+1]$

Pentru $SMIN[i,t]$ se alege minimul din cele 2 variante. Rezultatul dorit se afla în $SMIN[1,0]$.

Complexitatea soluţiei este $O(N*TMAX)$, unde ambele valori sunt mai mici sau egale cu 3000. Pentru a se încadra în limita de memorie, observăm că pentru calculul matricei $SMIN$ avem nevoie, la orice moment, doar de 2 linii (liniile i şi $i+1$) ale matricii $SMIN$; prin urmare, cantitatea necesară de memorie este $O(TMAX)$ şi nu $O(N*TMAX)$.

Soluţia 2 (stud. Adrian Paul Diaconu)

Se calculează vectorul $SUM[i]$ ca fiind suma sensurilor de modificare a agitaţiei pentru candidaţii $i, i+1, .., N$. Se alege valoarea minimă din acest vector.

Să presupunem că această valoare se obţine pentru poziţia k din vector. Dacă $SUM[k]$ este mai mică decât 0, atunci introducând o perioadă de aşteptare mai mare decât 0 înaintea candidatului k, suma totală a nivelelor de agitaţie va scădea.

Perioada de timp cu care se măreşte timpul de aşteptare dinaintea candidatului k este mărită cu valoarea minimă a nivelului de agitaţie dintre toţi candidaţii

$j \geq k$ având sensuri negative de modificare a agitaţiei. Se modifică corespunzător valorile nivelelor de agitaţie a tuturor candidaţilor $j \geq k$, iar pentru toţi candidaţii care ating nivelul de agitaţie 0, se schimbă şi sensul de modificare a nivelului de agitaţie (din -1 în $+1$).

Acest algoritm se repetă până în momentul în care valoarea minimă din vectorul SUM este mai mare sau egală cu 0. Complexitatea acestei soluţii este $O(N^2)$.

14.1.2 Rezolvare detaliată

14.1.3 Codul sursă *

```
import java.io.*;
class agitatie
{
  static StreamTokenizer st;
  static PrintWriter out;
  static int n,tmax=-1;
  static int[] a;
  static int[] b;
  static int[][] smin;

  public static void main(String[] args) throws IOException
  {
    int i,t,na;
    st=new StreamTokenizer(new BufferedReader(new FileReader("agitatie.in")));
    out=new PrintWriter(new BufferedWriter(new FileWriter("agitatie.out")));
    st.nextToken(); n=(int)st.nval;
    a=new int[n+1];
    b=new int[n+1];
    for(i=1;i<=n;i++)
    {
      st.nextToken(); a[i]=(int)st.nval;
      st.nextToken(); b[i]=(int)st.nval;
      if(a[i]>tmax) tmax=a[i];
    }
    smin=new int[n+2][tmax+2];

    for(i=0;i<=n;i++) smin[i][tmax+1]=9999999;
```

```
    for(i=n;i>=1;i--)
    {
      for(t=tmax;t>=0;t--)
      {
        if(b[i]==1) na=a[i]+t;
        else
        {
          if(t<=a[i]) na=a[i]-t; else na=t-a[i];
        }
        smin[i][t]=min(na+smin[i+1][t],smin[i][t+1]);
      }// for t
    }// for i
    //afissmin();
    System.out.println(smin[1][0]);
    out.println(smin[1][0]);
    out.close();
  }// main

  static int min(int a, int b)
  {
    if(a<b) return a; else return b;
  }

  static void afissmin()
  {
    int i,t;
    for(i=0;i<=n;i++)
    {
      for(t=0;t<=tmax;t++) System.out.print(smin[i][t]+"\t");
      System.out.println();
    }
    System.out.println();
  }// afissmin(...)
}// class
```

14.2 Coduri - ONI 2007

Întorcându-se de la şcoală în ziua în care a aflat cum se face înmulţirea numerelor, Gigel a auzit la televizor următoarea afirmaţie: "Pentru a face avere, nu trebuie să aduni bani în viaţă, ci trebuie s-i înmulţeşti".

Toate acestea l-au pus pe gânduri, aşa că s-a hotărât să inventeze propriul "sistem de codificare" pentru numere reale mai mari decât 0 care să aibă

următoarele proprietăţi:

- fiecare număr va fi codificat sub forma unui şir de valori întregi (pozitive şi/sau negative)

- dacă un număr real x are codul c_x şi un număr real y are codul c_y, atunci numărul real rezultat prin înmulţirea lui x şi y trebuie să aibă codul obţinut prin "adunarea" codurilor $c - x$ şi c_y.

- dacă un număr real x se poate scrie ca produs de numere y_1, y_2, ..., y_k, atunci codul lui x se obţine prin "adunarea" codurilor numerelor $y_1, y_2, ..., y_k$.

Considerăm un cod c_1 format din n_1 valori $a_{n_1}...a_1$ şi un cod c_2 format din n_2 valori $b_{n_2}...b1$, atunci codul c_3 obţinut prin "adunarea" codurilor c_1 şi c_2 va avea n_3 valori $d_{n_3}...d_1$, cu proprietăţile următoare:

•n_3 este maximul dintre n_1 şi n_2

$$\bullet d_i = \begin{cases} a_i + b_i & \text{pentru } i = 1, ..., min(n_1, n_2) \\ a_i & \text{pentru } i = minim(n_1, n_2) + 1, ..., n_1 \text{ dacă } minim(n_1, n_2) = n_2 \\ b_i & \text{pentru } i = minim(n_1, n_2) + 1, ..., n_2 \text{ dacă } minim(n_1, n_2) = n_1 \end{cases}$$

Cerinţă

Dându-se N numere reale mai mari strict decât 0, să se scrie codificarea acestora în sistemul inventat de Gigel.

Date de intrare

Fişierul de intrare **coduri.in** va conţine:

- pe prima linie din fişier se află numărul N de numere reale

- pe următoarele N linii cele N numere reale, fiecare pe câte o linie.

Date de ieşire

Fişierul de ieşire **coduri.out** va conţine N linii:

- pe linia i (i între 1 şi N) : numărul de valori folosite pentru codificarea numărului cu indicele i din fişierul de intrare, urmat de un spaţiu şi apoi valorile ce alcătuiesc codul numărului, separate două câte două printr-un singur spaţiu.

Restricţii şi precizări

• $2 \leq N \leq 18$

• Separatorul între partea întreagă şi partea zecimală este virgula.

• Orice număr are după virgulă cel mult 5 cifre.

• Valorile din codurile numerelor din fişierele de test trebuie să fie cuprinse în intervalul $[-106, 106]$.

• Partea întreagă a fiecărui număr real este o valoare mai mică sau egală cu 20000.

• Toate numerele din fişierele de test sunt strict pozitive şi distincte două câte două.

• Numărul maxim de valori utilizat pentru codificarea unui număr este 2500.

• Dacă există mai multe soluţii de codificare, se va afişa una singură.
• Nu trebuie să existe două numere diferite cu aceeaşi codificare.
• 40% din teste vor conţine numai numere întregi, 30% din teste vor conţine numere întregi şi numere reale fără perioadă şi 30% din teste vor conţine numere întregi şi numere reale cu şi fără perioadă.

Exemplu

coduri.in	coduri.out	Explicaţii
8	2 1 1	
10	3 -1 0 1	10=2*5 iar suma codurilor pentru 2 şi 5
2	3 1 1 0	determină codul lui 10
5	3 2 1 0	
0,3	3 -1 2 1	2,1=7*0,3 iar suma codurilor pentru 7 şi 0,3
7	3 1 3 1	determină codul lui 2,1
2,1	2 1 11	
1,(7)	2 1 2	
1,2(34)		

Timp maxim de execuţie/test(Windows/Linux): 0.2 secunde

14.2.1 Indicaţii de rezolvare - descriere soluţie *

Descrierea soluţiei (Prof. Roxana Tîmplaru)

O soluţie de codificare porneşte de la ideea că orice număr întreg se descompune în factori primi, sub forma $2^{k_1} \cdot 3^{k_2} \cdot 5^{k_3} \cdot 7^{k_4} \cdot ...a^{k_p}$. Codificarea acestui număr va avea k_p valori, iar acestea vor fi coeficienţii la care apar factorii primi în descompunerea numărului, adică $k_p...k_3k_2k_1$.

Numărul rezultat prin înmulţirea a două numere va avea coeficienţii în descompunere suma coeficienţilor celor două numere, deci codificarea numărului care va rezulta ca produs a celor două, va rezulta din suma codurilor acestor numere.

Pentru numerele cu virgulă, se foloseşte scrierea sub formă de fracţie, se descompune numărătorul şi numitorul, iar codificarea se va obţine scăzând din coeficienţii de la numărător pe cei de la numitor.

14.2.2 Rezolvare detaliată

14.2.3 Codul sursă *

Rezultatele nu corespund cu cele date în exemplu dar ... pare corect aici!

```
import java.io.*;
```

```
class coduri
{
  static BufferedReader br;
  static PrintWriter out;
  static int n;
  static String nr;
  static int[] p=new int[9593]; // nr prime
  static int[] f1=new int[9593];
  static int[] f2=new int[9593];
  static int iuf, iuf1, iuf2;

  public static void main(String[] args) throws IOException
  {
    int i,j,k;
    int nr1,nr2,nr3,aux,n10,x,y;

    br=new BufferedReader(new FileReader("coduri.in"));
    out=new PrintWriter(new BufferedWriter(new FileWriter("coduri.out")));

    nrprime();
    n=Integer.parseInt(br.readLine());
    for(i=1;i<=n;i++)
    {
      nr=br.readLine();
      nr1=0;
      for(j=0;j<nr.length();j++)
        if(nr.charAt(j)==',') break;
        else nr1=nr1*10+nr.charAt(j)-'0';

      //System.out.print(i+" : "+nr+" --> ");

      if(j==nr.length()) // intreg
      {
        x=nr1;
        y=1;

        //System.out.println("x = "+x+"   y = "+y);

        if(x==1) // 1 nu se mai desxcompune ... are codul "0"
        {
          out.println("1 0");
          continue;
        }
```

```
  descfact(x,f1);
  iuf1=iuf;

  out.print(iuf1+" ");
  for(k=iuf1;k>=2;k--) out.print(f1[k]+" ");
  out.println(f1[1]);

  continue;
}// if intreg

nr2=0;
for(k=j+1;k<nr.length();k++)
  if(nr.charAt(k)=='(') break;
  else nr2=nr2*10+nr.charAt(k)-'0';

if(k==nr.length()) // real neperiodic
{
  x=nr1;
  y=1;
  aux=nr2;
  while(aux!=0) { x*=10; y*=10; aux/=10;}
  x+=nr2;

  descfact(x,f1);
  iuf1=iuf;

  descfact(y,f2);
  iuf2=iuf;

  iuf=max(iuf1,iuf2);
  out.print(iuf+" ");
  for(k=iuf;k>=2;k--)
  {
    if(k<=iuf1 && k<=iuf2) out.print((f1[k]-f2[k])+" "); else
    if(k<=iuf1)
      out.print(f1[k]+" ");
    else
      out.print(-f2[k]+" ");
  }
  out.println(f1[1]-f2[1]);

  continue;
}// if real neperiodic
```

```
      // real periodic (nr1-nr2)/nr3
      aux=nr2;
      n10=1;
      while(aux!=0) {nr1*=10; aux/=10; n10=n10*10;}
      nr1+=nr2;
      nr2=nr1;
      nr3=0;
      for(j=k+1;j<nr.length()-1;j++)
      {
        nr1=nr1*10+nr.charAt(j)-'0';
        nr3=nr3*10+9;
      }
      nr3=nr3*n10;
      x=nr1-nr2;
      y=nr3;

      descfact(x,f1);
      iuf1=iuf;
      descfact(y,f2);
      iuf2=iuf;

      iuf=max(iuf1,iuf2);
      out.print(iuf+" ");
      for(k=iuf;k>=2;k--)
      {
        if(k<=iuf1 && k<=iuf2) out.print((f1[k]-f2[k])+" "); else
        if(k<=iuf1)
          out.print(f1[k]+" ");
        else
          out.print(-f2[k]+" ");
      }
      out.println(f1[1]-f2[1]);

    }// for i

    out.close();
  }// main(...)

  static void descfact(int nr, int[] f)
  {
    int i;
    i=1;
    f[1]=0;
    while(nr>1)
```

```
    {
      while(nr%p[i]==0) {f[i]++; nr/=p[i];}
      i++;
      f[i]=0;
    }
    iuf=i-1;
  }// descfact(...)

  static void nrprime()
  {
    int i,j;
    boolean ok;
    int k;
    p[1]=2;  p[2]=3; p[3]=5; p[4]=7; p[5]=11;
    p[6]=13; p[7]=17; p[8]=19; p[9]=23; // gata ...
    k=9;
    for(i=29;i<=99999;i=i+2)
    {
      ok=true;
      for(j=2; j<=k && p[j]*p[j]<=i && ok; j++)
        if(i%p[j]==0) ok=false;
      if(ok)
      {
        p[++k]=i;
      }
    }// for
  }// nrprime(...)

  static int max(int a, int b)
  {
    if(a>b) return a; else return b;
  }// max(...)
}// class
```

14.3 Lacuri - ONI 2007

Pe un teren de formă pătrată sunt zone de uscat şi lacuri. Harta terenului este memorată într-un tablou bidimensional $n * n$ cu valori 1 (apă) sau 0 (uscat). Liniile sunt numerotate de la 1 la n, de sus în jos şi coloanele de la 1 la n, de la stânga la dreapta. Fiecare lac este înconjurat de o suprafaţă de teren uscat. Excepţie fac doar lacurile situate la marginea terenului care sunt înconjurate de uscat doar prin interiorul terenului nu şi prin afara acestuia.

Cerinţă

Se doreşte să se traverseze terenul pe uscat, de la poziţia $(1, 1)$ la poziţia (n, n), pe un drum de lungime minimă, mergând pas cu pas. La un pas, se ajunge de la un pătrăţel la altul învecinat cu el spre nord, est, sud sau vest.

Să se scrie un program care:

a) Determină numărul lacurilor care sunt de formă pătrată şi în acelaşi timp sunt "pline de 1".

b) În cazul în care toate lacurile sunt de formă pătrată şi în acelaşi timp "pline de 1", determinaţi un drum cu proprietatea de mai sus.

Date de intrare

Fişierul de intrare **lacuri.in** are pe prima linie numărul n, iar pe următoarele n linii este harta terenului (fiecare linie are n valori 0 sau 1, separate de câte un spaţiu).

Date de ieşire

Fişierul de ieşire **lacuri.out** va conţine:

• pe prima linie: numărul de lacuri ce sunt de formă pătrată şi în acelaşi timp "pline de 1";

• în cazul în care toate lacurile sunt de formă pătrată şi în acelaşi timp "pline de 1", urmează liniile ce descriu drumul de lungime minimă ales. Fiecare linie din fişier conţine câte o pereche de coordonate ale poziţiilor succesive prin care trece drumul (linia şi coloana, separate cu un spaţiu), începând cu $(1, 1)$ şi terminând cu (n, n).

Restricţii şi precizări

• $2 \leq n \leq 100$

• Poziţiile $(1, 1)$ şi (n, n) sunt sigur libere (cu valoarea 0).

• Dacă există mai multe soluţii se va afişa oricare dintre ele.

• Pot fi cel mult 100 de lacuri şi cel puţin unul; dacă un lac este de formă pătrată, atunci $1 \leq$ latura $\leq n - 1$.

• Indiferent de forma lor, lacurile sunt sigur "izolate", adică nu se "ating" deloc de alt lac. De exemplu, dacă un lac conţine poziţia $(3, 3)$, atunci un alt lac nu poate conţine vreuna din poziţiile învecinate cu $(3, 3)$, adică: $(2, 3)$, $(2, 4)$, $(3, 4)$, $(4, 4)$, $(4, 3)$, $(4, 2)$, $(3, 2)$ şi $(2, 2)$.

• Pentru cerinţa *a*) se acordă 30% din punctaj, iar pentru cerinţa b) se acordă 70% din punctaj.

Exemplu

lacuri.in	lacuri.out
6	4
0 0 0 0 0 0	1 1
0 1 0 1 1 1	1 2
0 0 0 1 1 1	1 3
0 0 0 1 1 1	2 3
1 1 0 0 0 0	3 3
1 1 0 0 1 0	4 3
	5 3
	5 4
	5 5
	5 6
	6 6

Explicaţie:

a) Prima linie conţine 4 (sunt 4 lacuri de formă pătrată şi "pline de 1")

b) Deoarece toate cele 4 lacuri sunt de formă pătrată şi "pline de 1", se scrie şi drumul ales: de la $(1,1)$, $(1,2)$, $(1,3)$, $(2,3)$, $(3,3)$, ..., $(6,6)$.

Observaţii:

1) Dacă în poziţia $(3,5)$ ar fi fost un 0, atunci lacul cu latura 3 nu ar mai fi fost "plin de 1" şi atunci prima linie ar fi conţinut doar valoarea 3 (ar fi fost doar 3 lacuri pătrate şi "pline de 1").

2) în exemplul iniţial, dacă în poziţia $(6,1)$ ar fi fost valorea 0, atunci nu ar fi fost toate lacurile pătrate (cel din stânga-jos nu ar fi fost pătrat) şi s-ar fi afişat doar un 3 în fi?ierul de ieşire.

3) în exemplul iniţial, dacă în poziţia $(5,2)$ ar fi fost valoarea 0, atunci s-ar fi afişat doar un 3, pentru că lacul din stânga-jos nu ar fi un lac pătrat şi "plin de 1".

Timp maxim de execuţie/test: 0.25 secunde

14.3.1 Indicaţii de rezolvare - descriere soluţie *

Descrierea soluţiei (Prof. Dan Grigoriu)

a. Prima cerinţă: se parcurge tabloul pe linii şi pentru fiecare valoare 1 întâlnită se verifică dacă este colţul stânga-sus al unui pătrat ; dacă da, atunci se numără şi se şterge acel pătrat; dacă nu, atunci se reţine acest lucru. Apoi se reia căutarea unei eventuale alte valori 1.

b. A doua cerinţă: dacă toate lacurile sunt pătrate şi pline de 1, atunci se încearcă un drum apropiat de diagonală de la poziţia $(1,1)$ la poziţia (n,n); sigur va exista un asemenea drum deoarece lacurile sunt izolate şi pot fi ocolite.

Fie poziţia curentă (i,i). Cazuri:

1. poziţia $(i+1, i+1)$ este ocupată.

În acest caz, se va căuta poziţia liberă imediat următoare de pe diagonală. Fie ea (j, j). Drumul de la poziţia (i, i) la poziţia (j, j) este sigur liberă (pentru că lacurile sunt sub formă pătrată şi sunt izolate) fie pe "deasupra" lacului întâlnit, fie pe sub acesta. Verificăm una dintre rute şi dacă e liberă, atunci o folosim, altfel o alegem pe cealaltă. Observaţie: în final, dacă $j = n$, atunci se termină algoritmul.

2. poziţia $(i + 1, i + 1)$ este liberă.

În acest caz, se verifică dacă se poate ajunge la poziţia $(i + 1, i + 1)$ pe "deasupra" şi atunci se trece prin poziţia intermediară $(i, i + 1)$, iar dacă această poziţie intermediară este ocupată, se va ajunge la poziţia $(i+1, i+1)$ prin poziţia intermediară $(i+1, i)$, care este liberă sigur (tot din cauză că lacurile sunt izolate).

14.3.2 Rezolvare detaliată

14.3.3 Codul sursă *!

```
import java.io.*; // margine ... ??? = T6, T7
class lacuri
{
  static StreamTokenizer st;
  static PrintWriter out;
  static int n;
  static int np=0;
  static int[][] a;
  static int[][] b;

  static final int dimq=10000;
  static int[] q=new int[dimq]; // coada circulara

  public static void main(String[] args) throws IOException
  {
    st=new StreamTokenizer(new BufferedReader(new FileReader("lacuri.in")));
    out=new PrintWriter(new BufferedWriter(new FileWriter("lacuri.out")));

    int i,j,k;

    st.nextToken(); n=(int)st.nval;
    a=new int[n+2][n+2];
    b=new int[n+2][n+2];

    for(i=1;i<=n;i++)
```

```
      for(j=1;j<=n;j++)
      {
        st.nextToken(); a[i][j]=b[i][j]=(int)st.nval;
      }

    if(!suntnumaipatrate())
    {
      System.out.println("NU ... "+np);
      out.println(np);
    }
    else
    {
      out.println(np);
      System.out.println("DA ... "+np);
      out.println("1 1");
      i=1; // casuta pusa
      j=2; // prima casuta mai departe (libera sau ocupata)
      while(j<=n)
      {
        while(b[j][j]==1) j++;
        System.out.println("i = "+i+"  j = "+j);

        if(j==i+1) // casute apropiate
        {
          if(b[i][i+1]==0)
            out.println(i+" "+(i+1));
          else
            out.println((i+1)+" "+i);
          out.println(j+" "+j);
          i=j;
          j=i+1;
        }
        else // casute departate i=pus j=nepus
        {
          if(b[i+1][i]==0) // in jos, apoi dreapta
          {
            for(k=i+1;k<=j;k++) out.println(k+" "+i);
            for(k=i+1;k<=j;k++) out.println(j+" "+k);
          }
          else // dreapta, apoi in jos
          {
            for(k=i+1;k<=j;k++) out.println(i+" "+k);
            for(k=i+1;k<=j;k++) out.println(k+" "+j);
          }
```

```
          i=j;
          j=j+1;
        }// else casute departate
      }// while
    }// else, toate patratele sunt pline de 1
    out.close();
  }// main

  static boolean suntnumaipatrate()
  {
    int i,j,i0,j0,i1,j1;
    boolean ok,okfin;

    okfin=true;
    for(i0=1;i0<=n;i0++)
      for(j0=1;j0<=n;j0++)
        if(a[i0][j0]==1)
        {
          ok=true;
          for(j1=j0+1;j1<=n+1;j1++) if(a[i0][j1]==0) break;
          j1--;
          for(i1=i0+1;i1<=n+1;i1++) if(a[i1][j0]==0) break;
          i1--;
          System.out.print("patrat posibil: "+i0+" "+j0+"   "+i1+" "+j1);

          // verificare interior
          for(i=i0+1;i<=i1-1 && ok;i++)
            for(j=j0+1;j<=j1-1 && ok;j++)
              if(a[i][j]==0) ok=false;

          // verificare stanga
          for(i=i0;i<=i1 && ok;i++)
            if(a[i][j0-1]==1) ok=false;

          // verificare dreapta
          for(i=i0;i<=i1 && ok;i++)
            if(a[i][j1+1]==1) ok=false;

          // verificare sus
          for(j=j0;j<=j1 && ok;j++)
            if(a[i0-1][j]==1) ok=false;

          // verificare jos
          for(j=j0;j<=j1 && ok;j++)
```

```
            if(a[i1+1][j]==1) ok=false;

          fill(i0,j0);
          if(!ok) okfin=false; else np++;

          if(ok) System.out.println(" DA"); else System.out.println(" NU");
        }// for i0 j0 if ...

    return okfin;
  }// suntnumaipatrate(...)

  static void fill(int i0, int j0)
  {
    // recursiv depaseste stiva ... in Java !!!
    int ic,sc,k,i,j;
    a[i0][j0]=0;
    k=(i0-1)*n+j0;
    q[0]=k;
    ic=0;
    sc=1;
    while(ic!=sc)
    {
      k=q[ic];
      ic=(ic+1)%dimq;
      i=(k+n-1)/n;
      j=k%n;
      if(j==0) j=n;
      //System.out.println(n+" "+k+" "+i+" "+j);
      if(a[i-1][j]==1) {a[i-1][j]=0; q[sc]=(i-2)*n+j; sc=(sc+1)%dimq;}
      if(a[i+1][j]==1) {a[i+1][j]=0; q[sc]=(i)*n+j; sc=(sc+1)%dimq;}
      if(a[i][j-1]==1) {a[i][j-1]=0; q[sc]=(i-1)*n+j-1; sc=(sc+1)%dimq;}
      if(a[i][j+1]==1) {a[i][j+1]=0; q[sc]=(i-1)*n+j+1; sc=(sc+1)%dimq;}
    }// while
  }// fill(...)
}// class
```

14.4 Secv - ONI 2007

Se dă un şir de N numere întregi $A_1, A_2, ..., A_N$. Asupra acestui şir se poate efectua următoarea operaţie: se împarte şirul în 3 secvenţe nevide, se calculează valoarea maximă din fiecare secvenţă şi apoi se face suma acestor valori. Cu alte cuvinte se aleg doi indici $0 < i < j < N$ şi se calculează valorile

$X = \max\{A_k | 1 \leq k \leq i\}$,
$Y = \max\{A_k | i+1 \leq k \leq j\}$,
$Z = \max\{A_k | j+1 \leq k \leq N\}$
şi suma $S = X + Y + Z$.

Cerinţă

Calculaţi valoarea minimă a lui S care se poate obţine în urma unei astfel de operaţii şi determinaţi cei doi indici care separă secvenţele pentru a obţine această valoare.

Date de intrare

Prima linie a fişierului de intrare **secv.in** conţine un număr natural N reprezentând numărul de elemente al şirului de intrare, iar a doua linie conţine numerele întregi $A_1, A_2, ..., A_N$ separate prin câte un spaţiu.

Date de ieşire

Fi?şerul de ieşire **secv.out** va conţine:
- pe prima linie: valoarea minimă a sumei;
- pe a doua linie: două numere naturale i, j separate printr-un spaţiu, reprezentând indicii pentru care se obţine valoarea minimă pentru S prin aplicarea operaţiei descrise mai sus.

Restricţii şi precizări

- $3 \leq N \leq 30000$
- $A_1, A_2, ..., A_N$ sunt numere întregi din intervalul $[-10000, 10000]$
- în cazul în care există mai multe soluţii se poate afişa oricare dintre ele.

Exemplu

secv.in	secv.out	Explicaţii
7 3 2 1 5 6 3 2	10 2 3	Prima secvenţă : 3 2 - maximul este 3 A doua secvenţă : 1 - maximul este 1 A treia secvenţă : 5 6 3 2 - maximul este 6 Suma: 10

Timp maxim de execuţie/test: 0.1 secunde

14.4.1 Indicaţii de rezolvare - descriere soluţie *

Descrierea soluţiei (Adrian Diaconu)

Vom nota cu X maximul elementelor din şir. X va apărea ca maxim într-una din cele 3 secvenţe. Se disting trei cazuri:

a) X aparţine primei secvenţe. Acum se poate observa că a doua secvenţă se poate forma dintr-un singur element pe cazul optim deoarece, în caz contrar celealte elemente se pot lipi primei secvenţe (lucru care nu ar mări costul). De aceea vom considera că ultima secvenţă este pe rând formată din ultimele i elemente,

secvenţa 2 este formată din elementul $N-i$ şi prima secvenţă din primele $N-i-1$ şi alegem minimul dintre toate aceste posibilităţi de secc tionare.

b) X aparţine celei de a doua secvenţe, caz în care prima şi ultima secvenţă le vom considera formate dintr-un singur element dintr-un raţionament asemănător cu punctul a).

c) X aparţine celei de a treia secvenţe caz simetric cu cel de la punctul a).

Se alege minim global al tuturor celor trei cazuri.

Complexitate: $O(n)$

14.4.2 Rezolvare detaliată

14.4.3 Codul sursă *

```
import java.io.*;
class secv
{
  static StreamTokenizer st;
  static PrintWriter out;
  static int n;
  static int[] a;
  static final int oo=300000001;

  public static void main(String[] args) throws IOException
  {
    int i,j,x,y,z,max,imax=0,smin;
    int s1=oo,s2=oo,s3=oo;
    int i1=0,j1=0,i2=0,j2=0,i3=0,j3=0;

    st=new StreamTokenizer(new BufferedReader(new FileReader("10-secv.in")));
    out=new PrintWriter(new BufferedWriter(new FileWriter("secv.out")));

    st.nextToken(); n=(int)st.nval;
    a=new int[n+1];

    max=-oo;
    for(i=1;i<=n;i++)
    {
      st.nextToken(); a[i]=(int)st.nval;
      if(a[i]>max) { max=a[i]; imax=i;}
    }
```

```
// max in prima secventa (daca imax <= n-2)
if(imax<=n-2)
{
  x=max;
  s1=oo; // minim
  for(i=imax+1;i<=n-1;i++)
  {
    y=a[i];
    z=-oo;
    for(j=i+1;j<=n;j++) if(a[j]>z) { z=a[j]; j1=j;}
    if(x+y+z<s1) { s1=x+y+z; i1=i; j1=i1+1;}
  }// for
}// if

// x in a doua secventa
if((imax>1)&&(imax<n))
{
  s2=max+a[1]+a[n];
  i2=1;
  j2=n-1;
}

// max in a treia secventa
if(imax>=3)
{
  z=max;
  s3=oo; // minim
  for(j=2;j<=imax-1;j++)
  {
    y=a[j];
    x=0;
    for(i=1;i<=j;i++) if(a[i]>x) {x=a[i]; i3=i;}
    if(x+y+z<s3) { s3=x+y+z; j3=j;}
  }// for
  i3=j3-1;
}// if

smin=min(min(s1,s2),s3);
out.println(smin);
if(smin==s1) out.println(i1+" "+j1); else
if(smin==s2) out.println(i2+" "+j2); else
             out.println(i3+" "+j3);
out.close();
```

```
  }// main

  static int min(int a, int b)
  {
    if(a<b) return a; else return b;
  }// min(...)
}// class
```

14.5 Şotron - ONI 2007

Pe asfalt este desenat cu cretă un şotron, caroiaj format din $n * n$ căsuţe având aceleaşi dimensiuni (câte n căsuţe pe fiecare din cele n rânduri).

În fiecare căsuţă este scris câte un număr întreg din intervalul $[-100, 100]$. Fiecare jucător are câte o piatră pe care o aruncă într-o căsuţă a şotronului, şi sărind într-un picior, împinge piatra din căsuţă în căsuţă, pe un anumit traseu astfel încât punctajul obţinut din suma numerelor de pe traseul parcurs să fie cât mai mare.

Numerele din căsuţele şotronului sunt scrise cu două culori albastru şi alb, astfel încât să nu existe două căsuţe alăturate (pe cele patru direcţii Nord, Est, Sud, Vest) având numere scrise cu aceeaşi culoare. Întotdeauna, prima căsuţă din primul rând al şotronului are înscris un număr de culoare albastră.

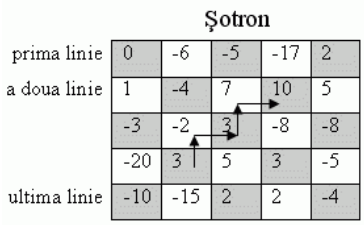
Se stabilesc apoi, următoarele reguli ale jocului:

- la începutul jocului, piatra poate fi aruncată în oricare căsuţă a şotronului. Din poziţia respectivă jucătorul îşi conduce piatra până la sfârşitul traseului stabilit de el;

- dintr-o căsuţă în care numărul este scris cu albastru, piatra poate fi deplasată doar în căsuţa vecină pe direcţia Nord;

- dintr-o căsuţă în care numărul este scris cu alb, piatra poate fi deplasată doar în căsuţa vecină pe direcţia Est;

- jucătorul poate alege orice căsuţă (inclusiv cea în care a aruncat piatra) pentru a încheia jocul, atâta timp cât piatra nu iese din şotron.



Cerinţă

Să se scrie un program care să determine cel mai mare punctaj care se poate obţine jucând şotron după regulile stabilite.

Date de intrare

Fişierul de intrare **sotron.in** are pe prima linie dimensiunea n a şotronului, iar pe următoarele n linii câte n numere separate de câte un spaţiu, reprezentând numerele scrise în şotron.

Date de ieşire

Fişierul de ieşire **sotron.out** va conţine pe prima linie un singur număr reprezentând punctajul maxim care se poate obţine jucând şotron.

Restricţii şi precizări

$1 \leq n \leq 240$

Exemplu

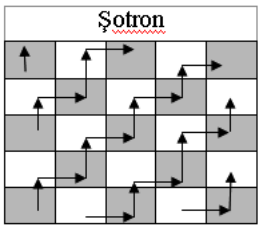
secv.in	secv.out	Explicaţii
5 0 -6 -5 -17 2 1 -4 7 10 5 -3 -2 3 -8 -8 -20 3 5 3 -5 -10 -15 2 2 -4	21	Punctajul obţinut este 3+(-2)+3+7+10=21

Timp maxim de execuţie/test(Windows/Linux): 0.1 secunde

14.5.1 Indicaţii de rezolvare - descriere soluţie *

Descrierea soluţiei (Prof. Cristina Bărbieru)

Se deplasează piatra în şotron din căsuţă în căsuţă, conform regulilor din enunţ, identificându-se astfel n trasee(în formă de zig-zag) de lungime (număr de căsuţe) maximă posibilă. Aceste trasee pornesc din fiecare căsuţă în care e scris un număr cu albastru, de pe prima coloană a şotronului sau dintr-o căsuţă cu număr scris cu alb de pe ultima linie a şotronului.



Pentru a calcula punctajul maxim obţinut de un jucător, se calculează pentru fiecare astfel de traseu suma maximă a numerelor unei succesiuni de căsuţe vecine.

Numerele scrise într-un traseu determinant la un moment dat se pot memora într-un vector, astfel, acestă problemă este echivalentă cu determinarea unei secvenţe de sumă maximă dintr-un vector.

14.5.2 Rezolvare detaliată

14.5.3 Codul sursă *

```
import java.io.*;
class Sotron
{
  static StreamTokenizer st;
  static PrintWriter out;
  static int n,smaxsol=-101;
  static int[][] a;
  static int[] x;

  public static void main(String[] args) throws IOException
  {
    int i,j,ii,jj,nc,smaxc;

    st=new StreamTokenizer(new BufferedReader(new FileReader("sotron.in")));
    out=new PrintWriter(new BufferedWriter(new FileWriter("sotron.out")));

    st.nextToken(); n=(int)st.nval;
    a=new int[n+1][n+1];
    x=new int[2*n];

    for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)
      {
        st.nextToken(); a[i][j]=(int)st.nval;
      }

    // prima coloana
    for(i=1;i<=n;i++)
    {
      if(i%2==0) continue;

      nc=0;
```

```
      x[++nc]=a[i][1];
      ii=i-1;
      jj=1;
      while(ii>=1)
      {
        x[++nc]=a[ii][jj]; // sus
        x[++nc]=a[ii][++jj]; // dreapta
        ii--;
      }
      smaxc=secvsummax(nc);
      if(smaxc>smaxsol) smaxsol=smaxc;
}// for i

    // ultima linie
    for(j=1;j<=n;j++)
    {
      if(n%2==0 && j%2==0) continue;
      if(n%2==1 && j%2==1) continue;

      nc=0;
      x[++nc]=a[n][j];
      ii=n;
      jj=j+1;
      while(jj<=n)
      {
        x[++nc]=a[ii][jj]; // dreapta
        x[++nc]=a[--ii][jj]; // sus
        jj++;
      }
      smaxc=secvsummax(nc);
      if(smaxc>smaxsol) smaxsol=smaxc;
    }// for j

    out.println(smaxsol);
    out.close();
  }// main(...)

  static int secvsummax(int n)
  {
    int i,i1,i2,ic,sc,smax;
    i1=i2=ic=1;
    smax=sc=x[1];
    for(i=2;i<=n;i++)
    {
```

```
      if(sc<=0) { ic=i; sc=x[i]; continue; } else sc=sc+x[i];
      if(sc>=smax) { smax=sc; i1=ic; i2=i; }
    }
    //System.out.println("smax = "+smax+"   x["+i1+"..."+i2+"]");
    return smax;
  }// secvsummax(...)
}// class
```

14.6 Triunghi - ONI 2007

În comuna **Triunghi** din România sunt n ţărani codificaţi prin numerele $1, 2, ..., n$. După anul 1990 a început retrocedarea suprafeţelor de pământ deţinute înainte de colectivizare. Fiecare ţăran are un document prin care dovedeşte că este proprietar pe o singură suprafaţă de teren de formă triunghiulară. Din păcate, documentele dau bătaie de cap primarului (care se ocupă de retrocedarea suprafeţelor de pământ), pentru că sunt porţiuni din suprafeţele de pământ care se regăsesc pe mai multe documente.

În această comună există o fântână cu apă, fiind posibil ca ea să fie revendicată de mai mulţi ţărani. O suprafaţă de pământ este dată prin coordonatele celor trei colţuri, iar fântâna este considerată punctiformă şi dată prin coordonatele punctului.

Cerinţă

Să se scrie un program care să determine:

a) Codurile ţăranilor care au documente cu suprafeţe de pământ ce conţin în interior sau pe frontieră fântâna.

b) Codul ţăranului ce deţine un document cu suprafaţa de teren, care include toate celelalte suprafeţe.

Date de intrare

Fişierul de intrare **triunghi.in** are pe prima linie numărul n de ţărani, pe următoarele n linii câte 6 valori numere întregi separate prin câte un spaţiu, în formatul: $x1$ $y1$ $x2$ $y2$ $x3$ $y3$, ce reprezintă coordonatele celor trei colţuri ale suprafeţei triunghiulare deţinute de un ţăran ($x1$, $x2$, $x3$ abscise, iar $y1$, $y2$, $y3$ ordonate). Pe linia $i+1$ se află coordonatele colţurilor suprafeţei de teren triunghiulare deţinute de ţăranul i, $i = 1, 2, ..., n$. Ultima linie a fişierului (linia $n+2$) va conţine coordonatele fântânii în formatul x y, cu un spaţiu între ele (x abscisă, iar y ordonată).

Date de ieşire

Fişierul de ieşire **triunghi.out** va conţine pe prima linie răspunsul de la punctul *a*), adică: numărul de ţărani care îndeplinesc condiţia din cerinţă şi apoi codurile lor (în ordine crescătoare), cu un spaţiu între ele. Dacă nu există ţărani cu condiţia din cerinţă, pe prima linie se va scrie cifra 0. Pe linia a doua se va scrie

răspunsul de la punctul b), adică: codul ţăranului cu proprietatea cerută, sau cifra 0, dacă nu există un astfel de ţăran.

Restricţii şi precizări

• $2 \leq n \leq 65$

• coordonatele colţurilor suprafeţelor de pământ şi ale fântânii sunt numere întregi din intervalul $[-3000, 3000]$

• cele trei colţuri ale fiecărei suprafeţe de pământ sunt distincte şi necoliniare

• nu există doi ţărani care să deţină aceeaşi suprafaţă de pământ

• nu se acordă punctaje parţiale.

Exemplu

secv.in	**secv.out**
3 10 0 0 10 10 10 0 100 100 0 -100 0 0 0 10 0 0 10 10 5	2 1 2 2

Explicaţie:

La punctul a), sunt doi ţărani care deţin suprafeţe de pământ ce au în interior sau pe frontieră fântâna, cu codurile 1 şi 2.

La punctul b), ţăranul cu codul 2 deţine o suprafaţă de teren care include, suprafeţele de pământ deţinute de ceilalţi ţărani (cu codurile 1 şi 3).

Timp maxim de execuţie/test: 0.1 secunde

14.6.1 Indicaţii de rezolvare - descriere soluţie *

Descrierea soluţiei (Prof. Doru Popescu Anastasiu)

Notăm cu $T_1, T_2, ..., T_n$ triunghiurile corespunzătoare suprafeţelor şi cu I punctul unde se găseşte fântâna.

$$T_i = A_iB_iC_i, \;\; i = 1, 2, ..., n.$$

a)

$nr = 0$

Pentru $i = 1, ..., n$ verificăm dacă I este interior sau pe frontiera lui T_i, în caz afirmativ $nr = nr + 1$ şi $sol[nr] = i$. Afişăm nr şi vectorul sol.

Pentru a verifica dacă I este interior sau pe frontiera unui triunghi T_i este suficient să verificăm dacă:

$$aria(A_iB_iC_i) = aria(IA_iB_i) + aria(IA_iC_i) + aria(IB_iC_i)$$

O altă variantă ar fi să folosim poziţia unui punct faţă de o dreaptă.

b)

Dacă există un asemenea triunghi atunci el este de arie maximă. Astfel determinăm triunghiul p de arie maximă. Pentru acest triunghi verificăm dacă toate celelalte $n-1$ triunghiuri sunt interioare sau pe frontiera lui T_p (adică dacă au toate vârfurile în interiorul sau pe frontiera lui T_p). În caz afirmativ se afişează p, altfel 0.

14.6.2 Rezolvare detaliată

14.6.3 Codul sursă *

```
import java.io.*;
class Triunghi
{
  static int n,x0,y0;
  static int smax=-1,imax=-1,nr=0;
  static int[] x1=new int[66];
  static int[] y1=new int[66];
  static int[] x2=new int[66];
  static int[] y2=new int[66];
  static int[] x3=new int[66];
  static int[] y3=new int[66];
  static int[] sol=new int[66];

  public static void main(String[] args) throws IOException
  {
    int i,j,s,s1,s2,s3;
    boolean ok;

    StreamTokenizer st=new StreamTokenizer(
      new BufferedReader(new FileReader("14-triunghi.in")));
    PrintWriter out=new PrintWriter(
      new BufferedWriter(new FileWriter("triunghi.out")));
    st.nextToken(); n=(int)st.nval;
    for(i=1;i<=n;i++)
    {
      st.nextToken(); x1[i]=(int)st.nval;
      st.nextToken(); y1[i]=(int)st.nval;
      st.nextToken(); x2[i]=(int)st.nval;
      st.nextToken(); y2[i]=(int)st.nval;
      st.nextToken(); x3[i]=(int)st.nval;
      st.nextToken(); y3[i]=(int)st.nval;
```

```
}
st.nextToken(); x0=(int)st.nval;
st.nextToken(); y0=(int)st.nval;
for(i=1;i<=n;i++)
{
  s=aria(x1[i],y1[i],x2[i],y2[i],x3[i],y3[i]);
  if(s>smax) {smax=s; imax=i;}
  s1=aria(x1[i],y1[i],x2[i],y2[i],x0,y0);
  s2=aria(x2[i],y2[i],x3[i],y3[i],x0,y0);
  s3=aria(x1[i],y1[i],x3[i],y3[i],x0,y0);
  if(s==s1+s2+s3) {nr++; sol[nr]=i;}
  //System.out.println("i = "+i+" --> "+s+" "+s1+" "+s2+" "+s3);
}
if(nr>0)
{
  out.print(nr+" ");
  for(i=1;i<=nr;i++)
    if(i!=nr) out.print(sol[i]+" "); else out.println(sol[i]);
}
else out.println(0);

//System.out.println("imax = "+imax);
ok=true;
for(i=1;i<=n;i++)
{
  if(i==imax) continue;

  s1=aria(x1[imax],y1[imax],x2[imax],y2[imax],x1[i],y1[i]);
  s2=aria(x2[imax],y2[imax],x3[imax],y3[imax],x1[i],y1[i]);
  s3=aria(x1[imax],y1[imax],x3[imax],y3[imax],x1[i],y1[i]);
  if(smax!=s1+s2+s3) { ok=false; break; }

  s1=aria(x1[imax],y1[imax],x2[imax],y2[imax],x2[i],y2[i]);
  s2=aria(x2[imax],y2[imax],x3[imax],y3[imax],x2[i],y2[i]);
  s3=aria(x1[imax],y1[imax],x3[imax],y3[imax],x2[i],y2[i]);
  if(smax!=s1+s2+s3) { ok=false; break; }

  s1=aria(x1[imax],y1[imax],x2[imax],y2[imax],x3[i],y3[i]);
  s2=aria(x2[imax],y2[imax],x3[imax],y3[imax],x3[i],y3[i]);
  s3=aria(x1[imax],y1[imax],x3[imax],y3[imax],x3[i],y3[i]);
  if(smax!=s1+s2+s3) { ok=false; break; }
}
if(ok) out.println(imax); else out.println(0);
out.close();
```

```
  }// main(...)

  static int aria(int x1, int y1, int x2, int y2, int x3, int y3) // dubla ...
  {
    int s=x1*y2+x2*y3+x3*y1-y1*x2-y2*x3-y3*x1;
    if(s<0) s=-s;
    return s;
  }
}// class
```