

# ALGORITMI ȘI STRUCTURI DE DATE 3

Note de Laborator

(uz intern - draft v1.1)

Adrian Răbăea



# Cuprins

<b>1</b>	<b>OJI 2002 clasa a XI-a</b>	<b>1</b>
1.1	Urgența - OJI 2002 . . . . .	1
1.1.1	Indicații de rezolvare - descriere soluție . . . . .	2
1.1.2	Rezolvare detaliată . . . . .	2
1.1.3	Codul sursă * . . . . .	2
1.2	Nunta - OJI 2002 . . . . .	6
1.2.1	Indicații de rezolvare - descriere soluție . . . . .	7
1.2.2	Rezolvare detaliată . . . . .	7
1.2.3	Codul sursă * !!! val max = ??? . . . . .	7
<b>2</b>	<b>OJI 2003 clasa a XI-a</b>	<b>11</b>
2.1	Compus - OJI 2003 . . . . .	11
2.1.1	Indicații de rezolvare - descriere soluție * . . . . .	12
2.1.2	Rezolvare detaliată . . . . .	14
2.1.3	Codul sursă * . . . . .	14
2.2	Zmeu - OJI 2003 . . . . .	16
2.2.1	Indicații de rezolvare - descriere soluție * . . . . .	17
2.2.2	Rezolvare detaliată . . . . .	18
2.2.3	Codul sursă * . . . . .	18
<b>3</b>	<b>OJI 2004 clasa a XI-a</b>	<b>23</b>
3.1	Moșia - OJI 2004 . . . . .	23
3.1.1	Indicații de rezolvare - descriere soluție * . . . . .	24
3.1.2	Rezolvare detaliată . . . . .	25
3.1.3	Codul sursă * . . . . .	25
3.2	Lanterna - OJI 2004 . . . . .	29
3.2.1	Indicații de rezolvare - descriere soluție * . . . . .	31
3.2.2	Rezolvare detaliată . . . . .	32
3.2.3	Codul sursă * . . . . .	32

<b>4</b>	<b>OJI 2005 clasa a XI-a</b>	<b>39</b>
4.1	Lanț - OJI 2005 . . . . .	39
4.1.1	Indicații de rezolvare - descriere soluție *	40
4.1.2	Rezolvare detaliată . . . . .	41
4.1.3	Codul sursă *	41
4.2	Scara - OJI 2005 . . . . .	48
4.2.1	Indicații de rezolvare - descriere soluție *	49
4.2.2	Rezolvare detaliată . . . . .	50
4.2.3	Codul sursă *	50
<b>5</b>	<b>OJI 2006 clasa a XI-a</b>	<b>59</b>
5.1	Graf - OJI 2006 . . . . .	59
5.1.1	Indicații de rezolvare - descriere soluție *	60
5.1.2	Rezolvare detaliată . . . . .	61
5.1.3	Codul sursă *	61
5.2	Cifru - OJI 2006 . . . . .	74
5.2.1	Indicații de rezolvare - descriere soluție *	75
5.2.2	Rezolvare detaliată . . . . .	77
5.2.3	Codul sursă *	77
<b>6</b>	<b>OJI 2007 clasa a XI-a</b>	<b>81</b>
6.1	Numere - OJI 2007 . . . . .	81
6.1.1	Indicații de rezolvare - descriere soluție *	82
6.1.2	Rezolvare detaliată . . . . .	83
6.1.3	Codul sursă *	83
6.2	Cezar - OJI 2007 . . . . .	87
6.2.1	Indicații de rezolvare - descriere soluție *	89
6.2.2	Rezolvare detaliată . . . . .	90
6.2.3	Codul sursă *	90
<b>7</b>	<b>ONI 2000 clasa a XI-a</b>	<b>99</b>
7.1	Arbore - ONI 2000 . . . . .	99
7.1.1	Indicații de rezolvare - descriere soluție . . . . .	100
7.1.2	Rezolvare detaliată . . . . .	100
7.1.3	Codul sursă . . . . .	101
7.2	Moara - ONI 2000 . . . . .	101
7.2.1	Indicații de rezolvare - descriere soluție . . . . .	102
7.2.2	Rezolvare detaliată . . . . .	102
7.2.3	Codul sursă . . . . .	102
7.3	Puncte - ONI 2000 . . . . .	102
7.3.1	Indicații de rezolvare - descriere soluție *	103
7.3.2	Rezolvare detaliată . . . . .	103
7.3.3	Codul sursă . . . . .	103
7.4	SICN - ONI 2000 . . . . .	103

7.4.1	Indicații de rezolvare - descriere soluție . . . . .	104
7.4.2	Rezolvare detaliată . . . . .	104
7.4.3	Codul sursă . . . . .	104
7.5	Spioni - ONI 2000 . . . . .	104
7.5.1	Indicații de rezolvare - descriere soluție . . . . .	106
7.5.2	Rezolvare detaliată . . . . .	106
7.5.3	Codul sursă . . . . .	106
7.6	Tezaur - ONI 2000 . . . . .	106
7.6.1	Indicații de rezolvare - descriere soluție . . . . .	108
7.6.2	Rezolvare detaliată . . . . .	108
7.6.3	Codul sursă . . . . .	108
<b>8</b>	<b>ONI 2001 clasa a XI-a</b>	<b>109</b>
8.1	Comparări - ONI 2001 . . . . .	109
8.1.1	Indicații de rezolvare - descriere soluție . . . . .	113
8.1.2	Rezolvare detaliată . . . . .	114
8.1.3	Codul sursă . . . . .	114
8.2	Relee - ONI 2001 . . . . .	114
8.2.1	Indicații de rezolvare - descriere soluție . . . . .	115
8.2.2	Rezolvare detaliată . . . . .	115
8.2.3	Codul sursă . . . . .	115
8.3	Telecomanda - ONI 2001 . . . . .	115
8.3.1	Indicații de rezolvare - descriere soluție . . . . .	117
8.3.2	Rezolvare detaliată . . . . .	117
8.3.3	Codul sursă . . . . .	117
8.4	Entries - ONI 2001 . . . . .	117
8.4.1	Indicații de rezolvare - descriere soluție . . . . .	118
8.4.2	Rezolvare detaliată . . . . .	118
8.4.3	Codul sursă . . . . .	118
8.5	Robot - ONI 2001 . . . . .	118
8.5.1	Indicații de rezolvare - descriere soluție . . . . .	120
8.5.2	Rezolvare detaliată . . . . .	120
8.5.3	Codul sursă . . . . .	120
8.6	Text mare - ONI 2001 . . . . .	120
8.6.1	Indicații de rezolvare - descriere soluție . . . . .	121
8.6.2	Rezolvare detaliată . . . . .	121
8.6.3	Codul sursă . . . . .	121
<b>9</b>	<b>ONI 2002 clasa a XI-a</b>	<b>123</b>
9.1	Arbore - ONI 2002 . . . . .	123
9.1.1	Indicații de rezolvare - descriere soluție * . . . . .	124
9.1.2	Rezolvare detaliată . . . . .	125
9.1.3	Codul sursă . . . . .	125
9.2	Decod - ONI 2002 . . . . .	125

9.2.1	Indicații de rezolvare - descriere soluție *	127
9.2.2	Rezolvare detaliată	127
9.2.3	Codul sursă	127
9.3	Seti - ONI 2002	128
9.3.1	Indicații de rezolvare - descriere soluție *	129
9.3.2	Rezolvare detaliată	129
9.3.3	Codul sursă	129
9.4	Suma divizorilor - ONI 2002	129
9.4.1	Indicații de rezolvare - descriere soluție *	130
9.4.2	Rezolvare detaliată	131
9.4.3	Codul sursă	131
9.5	Sistem - ONI 2002	131
9.5.1	Indicații de rezolvare - descriere soluție *	132
9.5.2	Rezolvare detaliată	132
9.5.3	Codul sursă	132
9.6	Comitat - ONI 2002	132
9.6.1	Indicații de rezolvare - descriere soluție *	134
9.6.2	Rezolvare detaliată	135
9.6.3	Codul sursă	135
<b>10</b>	<b>ONI 2003 clasa a XI-a</b>	<b>137</b>
10.1	Asmin - ONI 2003	137
10.1.1	Indicații de rezolvare - descriere soluție *	139
10.1.2	Rezolvare detaliată	140
10.1.3	Codul sursă	140
10.2	Căutare - ONI 2003	140
10.2.1	Indicații de rezolvare - descriere soluție *	142
10.2.2	Rezolvare detaliată	144
10.2.3	Codul sursă	144
10.3	A007 - ONI 2003	144
10.3.1	Indicații de rezolvare - descriere soluție *	145
10.3.2	Rezolvare detaliată	146
10.3.3	Codul sursă	147
10.4	Inter - ONI 2003	147
10.4.1	Indicații de rezolvare - descriere soluție *	147
10.4.2	Rezolvare detaliată	148
10.4.3	Codul sursă	148
10.5	Număr - ONI 2003	148
10.5.1	Indicații de rezolvare - descriere soluție *	149
10.5.2	Rezolvare detaliată	150
10.5.3	Codul sursă	151
10.6	Proc - ONI 2003	151
10.6.1	Indicații de rezolvare - descriere soluție *	152
10.6.2	Rezolvare detaliată	154

10.6.3	Codul sursă . . . . .	154
<b>11</b>	<b>ONI 2004 clasa a XI-a</b>	<b>155</b>
11.1	BASE3 - ONI 2004 . . . . .	155
11.1.1	Indicații de rezolvare - descriere soluție *	156
11.1.2	Rezolvare detaliată . . . . .	156
11.1.3	Codul sursă . . . . .	156
11.2	Coach - ONI 2004 . . . . .	157
11.2.1	Indicații de rezolvare - descriere soluție *	158
11.2.2	Rezolvare detaliată . . . . .	159
11.2.3	Codul sursă . . . . .	159
11.3	Color - ONI 2004 . . . . .	159
11.3.1	Indicații de rezolvare - descriere soluție *	160
11.3.2	Rezolvare detaliată . . . . .	160
11.3.3	Codul sursă . . . . .	160
11.4	Magic - ONI 2004 . . . . .	160
11.4.1	Indicații de rezolvare - descriere soluție *	161
11.4.2	Rezolvare detaliată . . . . .	162
11.4.3	Codul sursă . . . . .	162
11.5	Pătrate - ONI 2004 . . . . .	162
11.5.1	Indicații de rezolvare - descriere soluție *	163
11.5.2	Rezolvare detaliată . . . . .	164
11.5.3	Codul sursă . . . . .	164
11.6	Turnuri - ONI 2004 . . . . .	164
11.6.1	Indicații de rezolvare - descriere soluție *	165
11.6.2	Rezolvare detaliată . . . . .	165
11.6.3	Codul sursă . . . . .	166
<b>12</b>	<b>ONI 2005 clasa a XI-a</b>	<b>167</b>
12.1	Mașina - ONI 2005 . . . . .	167
12.1.1	Indicații de rezolvare - descriere soluție *	169
12.1.2	Rezolvare detaliată . . . . .	169
12.1.3	Codul sursă . . . . .	169
12.2	Matrice - ONI 2005 . . . . .	169
12.2.1	Indicații de rezolvare - descriere soluție *	171
12.2.2	Rezolvare detaliată . . . . .	172
12.2.3	Codul sursă . . . . .	172
12.3	Ziduri - ONI 2005 . . . . .	172
12.3.1	Indicații de rezolvare - descriere soluție *	173
12.3.2	Rezolvare detaliată . . . . .	174
12.3.3	Codul sursă . . . . .	174
12.4	Lsort - ONI 2005 . . . . .	174
12.4.1	Indicații de rezolvare - descriere soluție *	175
12.4.2	Rezolvare detaliată . . . . .	176

12.4.3	Codul sursă . . . . .	176
12.5	Pătrat - ONI 2005 . . . . .	176
12.5.1	Indicații de rezolvare - descriere soluție * . . . . .	177
12.5.2	Rezolvare detaliată . . . . .	177
12.5.3	Codul sursă * . . . . .	178
12.6	Cșir - ONI 2005 . . . . .	179
12.6.1	Indicații de rezolvare - descriere soluție * . . . . .	180
12.6.2	Rezolvare detaliată . . . . .	181
12.6.3	Codul sursă . . . . .	181
<b>13</b>	<b>ONI 2006 clasa a XI-a</b>	<b>183</b>
13.1	borg - ONI 2006 . . . . .	183
13.1.1	Indicații de rezolvare - descriere soluție * . . . . .	184
13.1.2	Rezolvare detaliată . . . . .	185
13.1.3	Codul sursă . . . . .	185
13.2	diamant - ONI 2006 . . . . .	185
13.2.1	Indicații de rezolvare - descriere soluție * . . . . .	186
13.2.2	Rezolvare detaliată . . . . .	186
13.2.3	Codul sursă . . . . .	186
13.3	matrice - ONI 2006 . . . . .	187
13.3.1	Indicații de rezolvare - descriere soluție * . . . . .	187
13.3.2	Rezolvare detaliată . . . . .	188
13.3.3	Codul sursă . . . . .	188
13.4	Petrom - ONI 2006 . . . . .	188
13.4.1	Indicații de rezolvare - descriere soluție * . . . . .	189
13.4.2	Rezolvare detaliată . . . . .	190
13.4.3	Codul sursă . . . . .	190
13.5	ratina - ONI 2006 . . . . .	190
13.5.1	Indicații de rezolvare - descriere soluție * . . . . .	191
13.5.2	Rezolvare detaliată . . . . .	192
13.5.3	Codul sursă . . . . .	192
13.6	vitale - ONI 2006 . . . . .	192
13.6.1	Indicații de rezolvare - descriere soluție * . . . . .	193
13.6.2	Rezolvare detaliată . . . . .	194
13.6.3	Codul sursă . . . . .	194
<b>14</b>	<b>ONI 2007 clasa a XI-a</b>	<b>195</b>
14.1	Descompunere - ONI 2007 . . . . .	195
14.1.1	Indicații de rezolvare - descriere soluție * . . . . .	196
14.1.2	Rezolvare detaliată . . . . .	197
14.1.3	Codul sursă . . . . .	197
14.2	Felinare - ONI 2007 . . . . .	197
14.2.1	Indicații de rezolvare - descriere soluție * . . . . .	198
14.2.2	Rezolvare detaliată . . . . .	199



14.2.3	Codul sursă . . . . .	200
14.3	Joc - ONI 2007 . . . . .	200
14.3.1	Indicații de rezolvare - descriere soluție * . . . . .	201
14.3.2	Rezolvare detaliată . . . . .	201
14.3.3	Codul sursă . . . . .	202
14.4	Logaritmi - ONI 2007 . . . . .	202
14.4.1	Indicații de rezolvare - descriere soluție * . . . . .	203
14.4.2	Rezolvare detaliată . . . . .	204
14.4.3	Codul sursă . . . . .	204
14.5	Maxq - ONI 2007 . . . . .	204
14.5.1	Indicații de rezolvare - descriere soluție * . . . . .	205
14.5.2	Rezolvare detaliată . . . . .	206
14.5.3	Codul sursă . . . . .	206
14.6	Tric - ONI 2007 . . . . .	206
14.6.1	Indicații de rezolvare - descriere soluție * . . . . .	207
14.6.2	Rezolvare detaliată . . . . .	207
14.6.3	Codul sursă . . . . .	208



# Capitolul 1

## OJI 2002 clasa a XI-a

### 1.1 Urgența - OJI 2002

Autoritățile dintr-o zonă de munte intenționează să stabilească un plan de urgență pentru a reacționa mai eficient la frecvențele calamități naturale din zonă. În acest scop au identificat  $N$  puncte de interes strategic și le-au numerotat distinct de la 1 la  $N$ . Punctele de interes strategic sunt conectate prin  $M$  căi de acces având priorități în funcție de importanță. Între oricare două puncte de interes strategic există cel mult o cale de acces ce poate fi parcursă în ambele sensuri și cel puțin un drum (format din una sau mai multe căi de acces) ce le conectează.

În cazul unei calamități unele căi de acces pot fi temporar întrerupte și astfel între anumite puncte de interes nu mai există legătură. Ca urmare pot rezulta mai multe grupuri de puncte în așa fel încât între oricare două puncte din același grup să existe măcar un drum și între oricare două puncte din grupuri diferite să nu existe drum.

Autoritățile estimează gravitatea unei calamități ca fiind suma priorităților căilor de acces distruse de aceasta și doresc să determine un scenariu de gravitate maximă, în care punctele de interes strategic să fie împărțite într-un număr de  $K$  grupuri.

#### Date de intrare

Fișierul de intrare URGENTA.IN are următorul format:

$N M K$

$i_1 j_1 p_1$  - între punctele  $i_1$  și  $j_1$  există o cale de acces de prioritate  $p_1$

$i_2 j_2 p_2$  - între punctele  $i_2$  și  $j_2$  există o cale de acces de prioritate  $p_2$

...

$i_M j_M p_M$  - între punctele  $i_M$  și  $j_M$  există o cale de acces de prioritate  $p_M$

**Date de ieșire**

Fișierul de ieșire URGENTA.OUT va avea următorul format:

*gravmax* - gravitatea maximă

*C* - numărul de căi de acces întrerupte de calamitate

*k<sub>1</sub> h<sub>1</sub>* - între punctele *k<sub>1</sub>* și *h<sub>1</sub>* a fost întreruptă calea de acces

*k<sub>2</sub> h<sub>2</sub>* - între punctele *k<sub>2</sub>* și *h<sub>2</sub>* a fost întreruptă calea de acces

...

*k<sub>C</sub> h<sub>C</sub>* - între punctele *k<sub>C</sub>* și *h<sub>C</sub>* a fost întreruptă calea de acces

**Restricții și precizări**

$0 < N < 256$

$N - 2 < M < 32385$

$0 < K < N + 1$

Prioritățile căilor de acces sunt întregi strict pozitivi mai mici decât 256.

Un grup de puncte poate conține între 1 și *N* puncte inclusiv.

Dacă există mai multe soluții, programul va determina una singură.

**Exemplu**

URGENTA.IN	URGENTA.OUT
7 11 4	27
1 2 1	8
1 3 2	1 3
1 7 3	1 7
2 4 3	2 4
3 4 2	3 4
3 5 1	3 7
3 6 1	4 5
3 7 5	5 6
4 5 5	6 7
5 6 4	
6 7 3	

**Timp maxim de executare:** 1 secundă / test

**1.1.1 Indicații de rezolvare - descriere soluție****1.1.2 Rezolvare detaliată****1.1.3 Codul sursă \***

```
import java.io.*; // arbore minim de acoperire: algoritmul lui Prim O(n^2)
```

```
class Urgenta    // sortare O(n^2) ... si una slaba merge!
{
    static final int oo=0x7fffffff;
    static int n,m,ncc,gravmax,costmax,nrm; // ncc = nr componente conexe
    static int[][] cost;
    static boolean[] esteInArbore;
    static int[] p;           // predecesor in arbore
    static int[] d;           // distante de la nod catre arbore
    static int[] a1;          // a1[k]=varful 1 al muchiei k din arbore
    static int[] a2;          // a2[k]=varful 2 al muchiei k din arbore
    static int[] ac;          // a1[k]=costul muchiei k din arbore

    public static void main(String[]args) throws IOException
    {
        int nodStart=3;       // nod start
        int i, j, k, costArbore=0,min,jmin=0,aux;

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

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

        cost=new int[n+1][n+1];
        esteInArbore=new boolean [n+1];
        p=new int[n+1];
        d=new int[n+1];
        a1=new int[n];
        a2=new int[n];
        ac=new int[n];

        for(i=1;i<=n;i++) for(j=1;j<=n;j++) cost[i][j]=oo;
        for(i=1;i<=n;i++) d[i]=oo;

        costmax=0;
        for(k=1;k<=m;k++)
        {
            st.nextToken(); i=(int)st.nval;
            st.nextToken(); j=(int)st.nval;
            st.nextToken(); cost[i][j]=cost[j][i]=(int)st.nval;
            costmax+=cost[i][j];
        }
    }
}
```

```
}

// alg Prim
d[nodStart]=0;

for(k=1;k<=n;k++) // O(n)
{
    min=oo;
    for(j=1;j<=n;j++) // O(n)
    {
        if(esteInArbore[j]) continue;
        if(min>d[j]) { min=d[j]; jmin=j; }
    }//for j

    esteInArbore[jmin]=true;
    d[jmin]=0;
    costArbore+=min;

    for(j=1;j<=n;j++) // actualizez distantele nodurilor // O(n)
    {
        if(esteInArbore[j]) continue; // j este deja in arbore
        if(cost[jmin][j]<oo) // am muchia (jmin,j)
            if(d[jmin]+cost[jmin][j]<d[j])
            {
                d[j]=d[jmin]+cost[jmin][j];
                p[j]=jmin;
            }
    }
} //for k

k=0;
for(i=1;i<=n;i++)
    if(p[i]!=0)
    {
        //System.out.println(i+" "+p[i]+" --> "+cost[i][p[i]]);
        k++;
        a1[k]=i;
        a2[k]=p[i];
        ac[k]=cost[i][p[i]];
    }
//System.out.println("cost="+costArbore);

gravmax=costmax-costArbore; // deocamdata, ...
```

```
//System.out.println("gravmax =" + gravmax);

// trebuie sa adaug la gravmax primele ncc-1 costuri mari (sort!)
// din arborele minim de acoperire

// sortez descrescator ac (pastrand corect a1 si a2)
// care are n-1 elemente
for(k=1;k<=n-1;k++) // de n-1 ori (bule)
{
    for(i=1;i<=n-2;i++)
        if(ac[i]<ac[i+1])
        {
            aux=ac[i]; ac[i]=ac[i+1]; ac[i+1]=aux;
            aux=a1[i]; a1[i]=a1[i+1]; a1[i+1]=aux;
            aux=a2[i]; a2[i]=a2[i+1]; a2[i+1]=aux;
        }
}

// primele ncc-1 ...
for(i=1;i<=ncc-1;i++) gravmax+=ac[i];

// sterg muchiile ramase in arbore ...
for(i=ncc;i<=n-1;i++)
{
    cost[a1[i]][a2[i]]=cost[a2[i]][a1[i]]=oo; //sterg
}

out.println(gravmax);

// determin numarul muchiilor ...
nrm=0;
for(i=1;i<=n-1;i++)
    for(j=i+1;j<=n;j++)
        if(cost[i][j] < oo)
            nrm++;
out.println(nrm);

// afisez muchiile ...
for(i=1;i<=n-1;i++)
    for(j=i+1;j<=n;j++)
        if(cost[i][j] < oo)
            out.println(i+" "+j);

out.close();
```

```

} //main
} //class

```

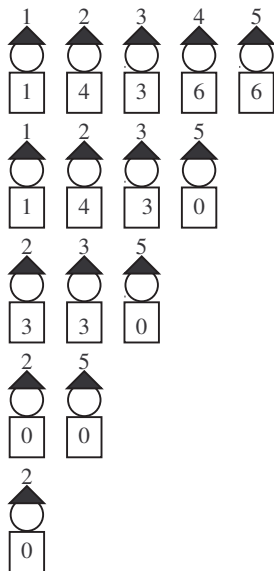
## 1.2 Nunta - OJI 2002

În fața palatului Prințesei Mofturoase se află  $N$  pețitori așezați la coadă, unul în spatele celuilalt. Fiecare poartă sub mantie un număr de pietre prețioase pe care dorește să le ofere prințesei ca dar de nuntă. Pentru a nu semăna vrajbă în rândurile lor, prințesa a decis să-i determine ca  $N - 1$  dintre ei să renunțe în chip pașnic, pețitorul rămas devenind alesul prințesei (indiferent de numărul de pietre prețioase deținute de acesta).

Doi pețitori vecini la coadă se pot înțelege între ei astfel: cel care are mai puține pietre prețioase pleacă de la coadă primind de la celălalt un număr de pietre astfel încât să plece acasă cu un număr dublu de pietre față de câte avea. Dacă doi pețitori au același număr de pietre, unul din ei (nu contează care) pleacă luând toate pietrele vecinului său.

Un pețitor se poate înțelege la un moment dat cu unul singur dintre cei doi vecini ai săi. După plecarea unui pețitor, toți cei din spatele lui avansează.

De exemplu: pentru configurația alăturată de 5 pețitori, un șir posibil de negocieri care conduc la reducerea cozii la un singur pețitor este: se înțeleg vecinii 4 cu 5 și pleacă 4, se înțeleg apoi 1 cu 2 și pleacă 1, se înțeleg apoi 3 cu 2 și pleacă 3, se înțeleg 2 cu 5 și pleacă 5. Astfel pețitorul 2 câștigă mâna preafrumoasei prințese, oferindu-i 0 pietre prețioase ca dar de nuntă.





Fie  $P$  numărul de pietre prețioase pe care le are pețitorul care va deveni alesul printesei. Se cer valorile distincte ale lui  $P$  la care se poate ajunge prin toate succesiunile de negocieri posibile.

#### Date de intrare

Fișierul de intrare **nunta.in** conține:

- pe prima linie numărul de pețitori:  $n$  ( $1 \leq n \leq 50$ ).
- pe a doua linie,  $n$  numere naturale din intervalul  $[0, 20]$ , reprezentând numărul de pietre prețioase pe care le dețin pețitorii, în ordinea în care stau la coadă.

#### Date de ieșire

Fișierul de ieșire **nunta.out** va conține:

- pe prima linie numărul  $m$  de valori distincte ce pot fi obținute
- pe a doua linie cele  $m$  valori ordonate crescător, reprezentând valorile care se pot obține.

#### Exemplu:

nunta.in	nunta.out
4	3
1 4 2 6	1 3 5

**Timp maxim de executare:** 1 secundă / test

### 1.2.1 Indicații de rezolvare - descriere soluție

### 1.2.2 Rezolvare detaliată

### 1.2.3 Codul sursă \* !!! val max = ???

```
import java.io.*; // nu rezolva testele: 2, 4, 5
class Nunta // val_max mai mic ... !!!
{
    // t2: 6 -> 1 5 7 11 13 15 corect: 3 --> 1 5 7
    // t4: 10 -> 1 3 5 7 9 11 13 15 17 19 corect: 9 -> 1 3 5 7 9 11 13 15 17
    // t5: 10 -> 0 2 4 6 8 10 12 14 16 18 corect: 9 -> 0 2 4 6 8 10 12 14 16

    static long[] a=new long[2001]; // a[i] i = -1000 ... 1000
    static long[] b=new long[2001]; // sir auxiliar

    static int[] p=new int[51]; // p[i] = nr pietre pretioase ...
    static int n,m,max,smax; // m = nr valori distincte
```

```
public static void main(String[] args) throws IOException
{
    //long t1,t2;
    //t1=System.currentTimeMillis();
    citesteDatele();
    determinaSolutia();
    scrieSolutia();
    //t2=System.currentTimeMillis();
    //System.out.println("\nTIMP = "+(t2-t1)+" milisecunde\n");
} // main()

static void citesteDatele() throws IOException
{
    int i;

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

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

    max=0;
    smax=0;
    for(i=1;i<=n;i++)
    {
        st.nextToken(); p[i]=(int)st.nval;
        smax+=p[i];
        if(p[i]>max) max=p[i];
    }
} // citesteDate()

static void scrieSolutia() throws IOException
{
    int i;
    PrintWriter out = new PrintWriter(
        new BufferedWriter(new FileWriter("nunta.out")));
    m=0;
    for(i=0;i<=max;i++)
        if(a[1000+i] != 0) m++;
    out.println(m);

    for(i=0;i<=max;i++)
        if(a[1000+i] == 1) out.print(i+" ");
}
```

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

    static void determinaSolutia()
    {
        int i,j,k;
        a[1000+0] = 1; // initial suma posibila este zero
        for(i=1;i<=n;i++)
        {
            for(j=-1000;j<=1000;j++)
            if (a[1000+j]!=0)
            {
                b[1000+j+p[i]] = b[1000+j-p[i]] = 1;
            }

            //System.out.print(i+ " : ");
            //for(j=-1000;j<=1000;j++)
            // if (b[1000+j]!=0)
            // System.out.print(j+ " ");
            //System.out.println();

            // copiezi b --> a si faci b=0
            for (j=-1000;j<=1000;j++)
            {
                a[1000+j] = b[1000+j];
                b[1000+j] = 0;
            }
        }
    }// determinaSolutia()
}// class
```



## Capitolul 2

# OJI 2003 clasa a XI-a

### 2.1 Compus - OJI 2003

La ultima expediție pe Marte a fost descoperit un compus organic necunoscut. Acest compus este acum studiat în laboratoarele NASA. Cercetătorii au descoperit că acest compus este constituit numai din atomi de hidrogen ( $H$ ), igrigen ( $I$ ) și carbin ( $C$ ) și are masa moleculară  $M$ .

Se știe că regulile de formare a compușilor organici pe Marte sunt următoarele:

- un atom de carbin se poate lega de oricare dintre atomii de  $C$ ,  $H$  și  $I$  cu oricâte dintre cele 4 legături pe care le are (astfel, în combinația  $H - C = C$  primul atom de carbin se leagă prin două legături de alt atom de carbin și cu o legătură de alt atom de hidrogen)

- un atom de hidrogen se poate lega numai de un atom de carbin cu singura legătură pe care o posedă

- un atom de igrigen se poate lega numai de atomi de carbin cu cele două legături pe care le posedă

- un compus este un ansamblu cu proprietatea că toți atomii de carbin sunt legați conex între ei și nu există vreun atom cu una sau mai multe legături libere (nelegate de un alt atom).

Combinația  $H - C = C$  nu este un compus deoarece atomii de carbin mai au legături libere.

Cercetătorii au în vedere studiul categoriilor de compuși, făcând distincție între doi compuși numai dacă aceștia diferă prin numărul de atomi de carbin, de igrigen sau de hidrogen.

#### **Cerință**

Scrieți un program care să determine câți compuși distincți formați din atomi de carbin, hidrogen și igrigen (cel puțin unul din fiecare) și care au masa moleculară  $M$  există.

**Date de intrare**

Fișierul de intrare **compus.in** conține pe prima linie masa moleculară a compusului.

**Date de ieșire**

Fișierul de ieșire **compus.out** conține o singură linie pe care se află numărul de compuși determinat.

**Restricții și precizări**

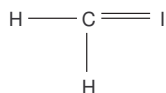
$$30 \leq M \leq 1000000$$

Masa atomului de *H* este 1, masa atomului de *C* este 5, iar masa atomului de *I* este 3. Masa moleculară a unui compus este egală cu suma maselor atomilor din care este constituit compusul respectiv.

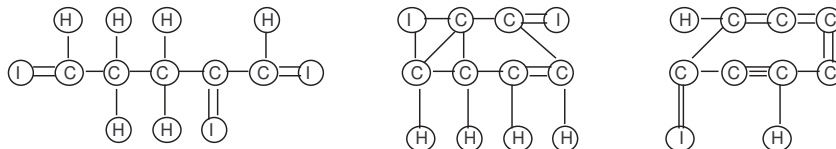
Ordinea în care sunt "utilizate" legăturile unui atom nu contează. De asemenea, nici ordinea atomilor sau legăturile interne dintre ei nu contează atâta timp cât respectă regulile de formare enunțate.

**Exemple**

Există un singur compus cu masa moleculară 10: cel format cu un atom de *C*, doi atomi de *H* și un atom de *I* ( $5 + 2 * 1 + 3 = 10$ ), compus ale cărui legături pot fi reprezentate astfel:



Se pot obține 3 compuși cu masa moleculară 40: (5*C*, 6*H*, 3*I*), (6*C*, 4*H*, 2*I*), (7*C*, 2*H*, 1*I*):



Reprezentarea cu legături a oricăruia dintre compuși nu este unică. Orice altă combinație corespunzătoare aceluiași triplet nu se consideră un compus distinct.

**Exemple**

compus.in	compus.out	compus.in	compus.out
40	3	125	28

**Timp maxim de executare/test:** 1 secundă

**2.1.1 Indicații de rezolvare - descriere soluție \***

*Soluția optimă și explicațiile pentru problema compus au fost realizate de Irina Dumitrașcu (studentă, Automatică, Politehnica București)*

Idei de bază:

Fiecare legătură consumă 2 capete de legături posibile  $\Rightarrow$  grupul de atomi de c va avea totdeauna un număr par de legături de oferit.

I consumă 2 capete  $\Rightarrow$  nu pot să am număr impar de H.

Numărul de legături ale atomilor de carbon:  $4c$

Numărul minim de legături între atomii de carbon:  $2c - 2$

$\Rightarrow$  Avem:

maxim  $2c + 2$  capete de legături disponibile pentru o și h

minim = 4 capete (trebuie să punem obligatoriu un I și 2 H)

### Ideea 1

c între  $c_{\text{Min}} = (m - 3) / 8$  și  $c_{\text{Max}} = (m - 4) / 5$

### Ideea 2

$M = 5 * c + 3 * i + h = 5 * c + 3 * i' + 3 + h' + 2$

(am scos separat un I și 2 H care trebuie să existe oricum în orice compus)

acum  $i' \geq 0$ ,  $h' \geq 0$

$M = 5 * c + 3 * i' + h' + 5$

Presupunem c cunoscut. Atunci

$3 * i' + h' = M - 5 * c - 5 = N$  (notație)

dar din condițiile impuse de legăturile oferite de grupul conex de C,

$$4 \leq 2 * i + h \leq 2 * c + 2$$

$$4 \leq 2 * i' + 2 + h' + 2 \leq 2 * c + 2$$

$$0 \leq 2 * i' + h' \leq 2 * c - 2$$

dar  $3 * i' + h' = N$ , deci

$$0 \leq N - i' \leq 2 * c - 2$$

Deci putem stabili limitele pentru i din inegalitățile

$$N - 2 * c + 2 \leq i' \leq N$$

$$\text{Notam } o_{\text{Min}} = \max(0, N - 2 * c + 2)$$

$$o_{\text{Max}} = \min(N, c)$$

Algoritmul devine

```
c între cMin si cMax
daca n > 0
  i între oMin si oMax
  verific daca h = M - 5c - 3i este par si pozitiv
```

### Ideea 3 - compus1.java

$h = N - 3 * i$  trebuie să fie par;

deci N are aceeași paritate cu  $3 * i \Rightarrow$  N are aceeași paritate cu i; pot să stabilesc paritatea la început și apoi să cresc din 2 în 2; așa nu trebuie să mai verific paritatea lui h

```
if ((N mod 2) <> (i mod 2)) then inc(i);
h := N - i * 3;
```

```

while ((h >= 0) and (i <= oMax)) do
begin
  inc(sol);
  i := i + 2;
  h := h - 6;
end;

```

#### Ideea 4 - compus2.java

În momentul ăsta pot să observ că while-ul e de fapt degeaba, și se reduce la

```

if ((N mod 2) <> (i mod 2)) then inc(o);
h := N - o * 3;

if ((h >= 0) and (i <= oMax)) then
begin
  sol := sol + min ( h div 6 + 1, (oMax - i + 2) div 2);
end

```

Am ajuns în  $O(N)$  !!

### 2.1.2 Rezolvare detaliată

#### 2.1.3 Codul sursă \*

Prima variantă:

```

import java.io.*;
class compus
{
  static int m, n, c, h, o, oMax, sMax, sol;

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

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

    sol=0;

```



```

for(c=(m-3)/8+1; c<=(m-4)/5;c++)
{
    n=m-c*5-5;
    if(n>=0)
    {
        o=max(0,n-2*c+2);
        oMax=min(c, n);
        sMax=2*(c-1);

        if((n%2)!=o%2) o++;
        h=n-o*3;

        while((h>=0)&&(o<=oMax)) { sol++; o=o+2; h=h-6; }
    }
}

out.println(sol);
out.close();
} // main

static int max(int a, int b)
{
    if(a>b) return a; else return b;
}

static int min(int a, int b)
{
    if(a<b) return a; else return b;
}
} // class

```

A doua variantă:

```

import java.io.*;
class compus
{
    static int m, n, c, h, o, oMax, sol;

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

```

```

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

sol=0;
for(c=(m-3)/8+1; c<=(m-4)/5;c++)
{
    n=m-c*5-5;
    if(n>=0)
    {
        o=max(0,n-2*c+2);
        oMax=min(c, n);

        if((n%2)!=o%2) o++;
        h=n-o*3;

        if((h>=0)&&(o<=oMax)) sol=sol+min(h/6+1,(oMax-o)/2+1);
    }
}

out.println(sol);
out.close();
} // main

static int max(int a, int b)
{
    if(a>b) return a; else return b;
}

static int min(int a, int b)
{
    if(a<b) return a; else return b;
}
} // class

```

## 2.2 Zmeu - OJI 2003

*Prof. Dana Lica - Ploiești*

Un zmeu cu  $n$  capete călătorește din poveste în poveste, iar în poveștile tradiționale întâlnește câte un Făt Frumos care-l mai scurtează de câteva capete, în timp ce în poveștile moderne salvează omenirea mâncând în timp record, cu toate capetele lui, insecte ucigașe apărute prin mutații genetice. Într-o seară, el își planifică o secesiune de povești cărora să le dea viață. El știe  $p$  povești numerotate de la 1 la  $p$ , durata fiecăreia și numărul de capete pe care le pierde în fiecare poveste.

Mai știe o mulțime de  $k$  perechi de povești, semnificând faptul că a doua poveste din pereche nu poate fi spusă după prima poveste din pereche.

### Cerință

Știind că trebuie să înceapă cu povestea 1 și să încheie succesiunea cu povestea  $p$ , ajutați bietul zmeu să aleagă una sau mai multe povești intermediare astfel încât durata totală să fie minimă și să rămână cu cel puțin un cap la sfârșitul tuturor poveștilor.

### Date de intrare

Fișierul de intrare **zmeu.in** conține pe prima linie numerele  $n$ ,  $p$  și  $k$  despărțite prin câte un spațiu. Pe fiecare din următoarele  $p$  linii se află câte o pereche de numere  $d_i$  și  $c_i$  (separate prin câte un spațiu) ce reprezintă durata și numărul de capete tăiate pentru fiecare poveste. Iar pe ultimele  $k$  linii se află câte o pereche de numere  $p_i$  și  $p_j$  (separate prin câte un spațiu) ce semnifică faptul că povestea  $p_j$  nu poate fi spusă după povestea  $p_i$ .

### Date de ieșire

Fișierul de ieșire **zmeu.out** conține o singură linie pe care se află un număr natural reprezentând durata (minimă) a succesiunii de povești sau valoarea  $-1$  dacă nu există o astfel de succesiune.

### Restricții și precizări

$$2 \leq N \leq 500$$

$$1 \leq P \leq 200$$

$$1 \leq k \leq 30000$$

Valorile reprezentând duratele și numărul de capete sunt numere naturale (duratele fiind strict pozitive), nedepășind valoarea 10.

### Exemple

zmeu.in	zmeu.out
10 4 2	9
2 6	
4 0	
1 3	
3 3	
3 2	
4 3	

**TimP maxim de executare/test:** 1 secundă

## 2.2.1 Indicații de rezolvare - descriere soluție \*

*Problemă propusă de prof. Dana Lica (Ploiești)*

Se lucrează pe un graf orientat în care nodurile reprezintă povești și din care s-au eliminat niște arce (date în **zmeu.in**,  $i \rightarrow 1$ ,  $p \rightarrow i$  și  $1 \rightarrow p$ )

Rezolvarea are la bază un algoritm de tip Lee cu memorarea în fiecare nod a tuturor timpilor minimi cu care se ajunge în nodul respectiv cu 1, 2, ..., c capete netăiate.

Pentru testele mari sunt necesare tehnici de alocare și optimizări legate de parcurgerea în lățime a grafului.

## 2.2.2 Rezolvare detaliată

### 2.2.3 Codul sursă \*

```
import java.io.*;
class zmeu // Bellman Ford (cu coada)
{
    static StreamTokenizer st;
    static PrintWriter out;

    static final int CMAX=10001; // dimensiune coada circulara
    static final int oo=999999999;

    static int[] [] g1; // graf - "matrice" de adiacenta
    static int[] [] g2; // graf - "liste" de adiacenta

    static int[] d; // d[i]=durata in nodul (povestea) i;
    static int[] c; // c[i]=capete taiate in nodul (povestea) i;

    static int[] [] dpc; // dpc[i][j]=durata ... ajunge in povestea i cu j capete taiate
    static boolean[] [] nuesteutil;

    static int[] cp=new int[CMAX]; // coada poveste
    static int[] cc=new int[CMAX]; // coada capete taiate deja

    static int n,p,k,dmin;

    static int nri=0,nre=0, nrmax=0; // testare ... dim coada ... !!!

    public static void main(String[] args) throws IOException
    {
        citire();
        sol();
        scrieSOL();
    }
}
```

```

    System.out.println("nrmax = "+nrmax);
}

static void sol()
{
    // daca c[1]>=n ==> nu are solutie (pierde toate capetele in povestea 1 ... !!!)
    if(c[1]>=n)
    {
        dmin=-1;
        return;
    }
    dmin=distmin(n-1);    // sa ramana cel putin un cap in fiecare poveste
} // sol()

static int distmin(int cmax)
{
    //System.out.println("--> disdmin  cmax = "+cmax);

    int ic,sc,i,j,i0,j0,ii,vi0,cvi0,dvi0;

    for(i=1;i<=p;i++)
        for(j=0;j<=cmax;j++)
        {
            dpc[i][j]=oo;
        }
    dpc[1][c[1]]=d[1];    // start din 1 cu c[1] capete taiate si durata=d[1]

    dmin=oo;    // initializare distanta minima ...

    cp[0]=1;    // nodul 1 in coada
    cc[0]=c[1];    // capete taiate la start
    ic=0;
    sc=1;    // prima pozitie libera

    nri=0; nre=0; ++nri;

    while(ic!=sc)    // coada nevida
    {
        //System.out.println("ic = "+ic+"    "+"sc = "+sc);
        i0=cp[ic];    // scot un nod din coada
        j0=cc[ic];
        ic=(1+ic)%CMAX;    // coada circulara !!!

        //System.out.println("  q --> "+i0+" "+j0);
    }
}

```

```

nre--;

if(nuesteutil[i0][j0])
{
    //System.out.println("Inutil sa propage ... "+i0+","+j0+");
    continue;
}

// propag catre vecinii lui i0
for(ii=1;ii<=g2[i0][0];ii++)
{
    vi0=g2[i0][ii];          // vecinul lui i0
    cvi0=j0+c[vi0];         // capete taiate ... ajuns in vi0
    dvi0=dpc[i0][j0]+d[vi0];

    if(cvi0>cmax)
    {
        //System.out.println("Nu ajung capetele ...");
        continue;
    }

    if(dvi0>dmin)
    {
        //System.out.println("Deja este o distanta prea mare ...");
        continue;
    }

    if(vi0==p)              // dmin ca sa nu cau min pe linia p
    {
        if(dvi0<dmin) dmin=dvi0;
        dpc[vi0][cvi0]=dvi0; // pentru depanare !
        //System.out.println("Am ajuns la destinatie ... pe un drum ...");
    }
    else
    { // vi0 != n
        if(dvi0<dpc[vi0][cvi0]) // distanta mai mica ...
        {
            //System.out.print("Distanta mai mica ... ");
            dpc[vi0][cvi0]=dvi0;
            cp[sc]=vi0;          // pun in coada vi0
            cc[sc]=cvi0;
            sc=(1+sc)%CMAX;
        }
    }
}

```

```

        //System.out.println("    q <-- "+vi0+" "+cvi0);

        nri++;
        if(nri-nre>nrmax) nrmax=nri-nrmax;

        nusteutil[vi0][cvi0]=false;

        // optimizare ...
        for(j=cvi0+1;j<=cmax;j++)
            if(dpc[vi0][j]>dvi0)
                {
                    dpc[vi0][j]=dvi0;
                    // il marchez "inutil"...
                    // daca este in coada ... sa nu mai propage aiurea !
                    nusteutil[vi0][j]=true;
                }
            else
                {
                    break;
                }
        }// if
    else
        {
            //System.out.println("Distanta mai mare sau egala ...");
        }
    }// j!=n
}//for ii
}//while

return dmin;
}//timpmin

static void citire() throws IOException
{
    int i,j,ij;

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

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

    g1=new int[p+1][p+1];
    g2=new int[p+1][p+1];

```

```
d=new int[p+1];
c=new int[p+1];
dpc=new int[p+1][n+1];
nuesteutil=new boolean[p+1][n+1];

for(i=1;i<=p-1;i++) for(j=i+1;j<=p;j++) g1[i][j]=g1[j][i]=1;

for(i=1;i<=p;i++)
{
    st.nextToken(); d[i]=(int)st.nval;
    st.nextToken(); c[i]=(int)st.nval;
}

// graf cu matrice de adiacenta
for(ij=1;ij<=k;ij++)
{
    st.nextToken(); i=(int)st.nval;
    st.nextToken(); j=(int)st.nval;
    g1[i][j]=0;
}

// nu este permis traseu direct 1 --> p
g1[1][p]=0;

// sterg si arcele de iesire din p si arcele de intrare in 1
for(i=1;i<=p;i++) g1[p][i]=g1[i][1];

// graf cu liste de adiacenta
for(i=1;i<=p;i++)
    for(j=1;j<=p;j++)
        if(g1[i][j]==1)
            {
                g2[i][0]++;
                g2[i][g2[i][0]]=j;
            }
}

} // citire(...)

static void scrieSOL() throws IOException
{
    out=new PrintWriter(new BufferedWriter(new FileWriter("zmeu.out")));
    out.println(dmin);
    out.close();
}
} // class
```



## Capitolul 3

# OJI 2004 clasa a XI-a

### 3.1 Moșia - OJI 2004

Păcală a primit, așa cum era învoiala, un petec de teren de pe moșia boierului. Terenul este împrejmuțit complet cu segmente drepte de gard ce se sprijină la ambele capete de câte un par zdravăn. La o nouă prinsoare, Păcală iese iar în câștig și primește dreptul să strămute niște pari, unul câte unul, cum i-o fi voia, astfel încât să-și extindă suprafața de teren. Dar învoiala prevede că fiecare par poate fi mutat în orice direcție, dar nu pe o distanță mai mare decât o valoare dată (scrisă pe fiecare par) și fiecare segment de gard, fiind cam șubred, poate fi rotit și prelungit de la un singur capăt, celălalt rămânând nemișcat.

Cunoscând pozițiile inițiale ale parilor și valoarea înscrisă pe fiecare par, se cere suprafața maximă cu care poate să-și extindă Păcală proprietatea. Se știe că parii sunt dați într-o ordine oarecare, pozițiile lor inițiale sunt date prin numere întregi de cel mult 3 cifre, distanțele pe care fiecare par poate fi deplasat sunt numere naturale strict pozitive și figura formată de terenul inițial este un poligon neconvex.

#### **Date de intrare**

Fișierul MOSIA.IN conține  $n + 1$  linii cu următoarele valori:

$n$  - numărul de pari

$x_1 y_1 d_1$  - coordonatele inițiale și distanța pe care poate fi mutat parul 1

$x_2 y_2 d_2$  - coordonatele inițiale și distanța pe care poate fi mutat parul 2

...

$x_n y_n d_n$  - coordonatele inițiale și distanța pe care poate fi mutat parul  $n$

#### **Date de ieșire**

În fișierul MOSIA.OUT se scrie un număr real cu 4 zecimale ce reprezintă suprafața maximă cu care se poate mări moșia.

**Restricții și observații:**

$3 < N \leq 200$  număr natural

$-1000 < x_i, y_i < 1000$  numere întregi

$0 < d_i \leq 20$  numere întregi

poligonul neconcav se definește ca un poligon convex cu unele vârfuri coliniare  
pozițiile parilor sunt date într-o ordine oarecare

poligonul obținut după mutarea parilor poate fi concav

pozițiile finale ale parilor nu sunt în mod obligatoriu numere naturale

**Exemplu**

Pentru fișierul de intrare

```
4
-3 0 2
3 0 3
0 6 2
0 -6 6
```

se va scrie în fișierul de ieșire valoarea 30.0000

**Explicație:** prin mutarea parilor 1 și 2 cu câte 2 și respectiv 3 unități, se obține un teren având suprafața cu 30 de unități mai mare decât terenul inițial.

**Timp limită de executare:** 1 sec./test

**3.1.1 Indicații de rezolvare - descriere soluție \***

*Mihai Stroe, Ginfo nr 14/4 - aprilie 2004*

Această problemă se poate împărți în trei subprobleme mai mici.

Prima subproblemă constă în ordonarea punctelor în sens trigonometric sau anti-trigonometric pentru ca șirul lor să descrie un poligon.

Pentru aceasta vom alege cel mai de jos punct și îl vom pune pe prima poziție, iar apoi vom sorta celelalte puncte în funcție de unghiurile formate de segmentul care are ca și capete punctul ales la început și un punct dintre cele nesortate și dreapta orizontală care trece prin punctul ales la început.

După ce prima subproblemă a fost rezolvată se poate trece la rezolvarea celei de-a doua subprobleme.

A doua subproblemă constă în a determina surplusul de suprafață pe care îl aduce mutarea fiecăruia dintre pari și acesta este egal cu produsul dintre distanța maximă pe care poate fi mutat un par și distanța dintre cei doi pari vecini ai săi împărțită la 2.

La acest rezultat se ajunge prin câteva calcule simple din geometrie ținându-se cont de faptul că dacă mutăm un par, nu vom mai putea muta vecinii săi datorită restricțiilor impuse în enunț. Aceste surplusuri (costuri) sunt reținute într-un vector.

A treia subproblemă constă în a determina suprafața maximă cu care poate fi mărită moșia ținând cont de restricții.

Această subproblemă se rezolvă prin metoda programării dinamice și este asemănătoare problemei "Oo" care a fost propusă spre rezolvare la concursul de programare Stelele Informaticii al cărei enunț poate fi găsit în GInfo 8/2003, iar rezolvarea în GInfo 1/2004.

Diferența dintre aceste două probleme este dată de faptul că la problema Oo, fermierul Ion poate aduna ouăle din două sectoare și nu mai poate aduna ouăle din sectoarele vecine celor din care a adunat. În cazul acestei subprobleme va trebui să construim un tablou unidimensional A cu N elemente. Elementul de pe o poziție i indică suprafața maximă cu care poate fi mărită moșia prin mutarea parului i. Soluția este dată de cea mai mare valoare din acest vector.

#### **Analiza complexității**

Ordinul de complexitate al operației de citire a datelor de intrare este  $O(N)$ .

Ordinul de complexitate a primei subprobleme este dat de ordinul de complexitate al operației de sortare utilizate și este  $O(N \log N)$ , dacă se folosește sortarea rapidă.

Ordinul de complexitate a celei de-a doua și a treia subprobleme este  $O(N)$ .

Operația de scriere a rezultatelor se realizează în  $O(1)$ .

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este  $O(N \log N)$ .

### 3.1.2 Rezolvare detaliată

### 3.1.3 Codul sursă \*

```
import java.io.*;
class mosia
{
    static final int NMAX=200;

    static double smax=-1,smx1=-1,smx2=-1;
    static int n,k,vs; // vs=varf stiva
    static int[] x=new int [NMAX+1];
    static int[] y=new int [NMAX+1];
    static int[] d=new int [NMAX+1];
    static int[] dd=new int [NMAX+1];

    static int[] xs=new int [NMAX+1]; // sortate
    static int[] ys=new int [NMAX+1];
    static int[] os=new int [NMAX+1];
}
```

```

static int[] xc=new int[NMAX+1];           // in convex
static int[] yc=new int[NMAX+1];
static int[] oc=new int[NMAX+1];

static boolean[] liber=new boolean[NMAX+1];
static int[] st=new int[NMAX+1];          // stiva pentru convex
static boolean[] fm1=new boolean[NMAX+1]; // folosit in sm1(1..n-1)
static boolean[] fm2=new boolean[NMAX+1]; // folosit in sm2(2..n)

static double[] dc=new double[NMAX+1];
static double[] a=new double[NMAX+1];
static double[] sm1=new double[NMAX+1];
static double[] sm2=new double[NMAX+1];

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

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

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

    for(i=1;i<=n;i++) os[i]=i; // inainte de sortare !
    qsort(1,n); // ordonez punctele dupa: 1. y=crescator; 2. x=crescator;

    convex();

    for(i=2;i<=n-1;i++) dc[i]=dist(xc[i-1],yc[i-1],xc[i+1],yc[i+1]);
    dc[1]=dist(xc[n],yc[n],xc[2],yc[2]);
    dc[n]=dist(xc[n-1],yc[n-1],xc[1],yc[1]);
    for(i=1;i<=n;i++) dd[i]=d[oc[i]];
    for(i=1;i<=n;i++) a[i]=dc[i]*dd[i]/2.0;

    dinamic(a,sm1,fm1,1,n-1);
    dinamic(a,sm2,fm2,2,n);
}

```

```

smax1=sm1[n-1]; if(!fm1[1]&&!fm1[n-1]) smax1+=x[n];
smax2=sm2[n]; if(!fm2[2]&&!fm2[n]) smax2+=x[1];
if(smax1>smax2) smax=smax1; else smax=smax2;

int smaxi=(int)(smax+0.0000001);
int smaxf=(int)((smax+0.0000001)-smaxi)*1000000);
out.println(smaxi+"."+smaxf); // cu 6 zecimale exacte si ...
out.close();
} // main

static void qsort(int l, int r)
{
    int i,j,mx,my,ax,aux;
    i=l; j=r; my=y[(l+r)/2]; mx=x[(l+r)/2];
    do
    {
        while((y[i]<my)||((y[i]==my)&&(x[i]<mx))) i++;
        while((y[j]>my)||((y[j]==my)&&(x[j]>mx))) j--;
        if(i<=j)
        {
            aux=x[i];x[i]=x[j];x[j]=aux;
            aux=y[i];y[i]=y[j];y[j]=aux;
            aux=os[i];os[i]=os[j];os[j]=aux;
            i++; j--;
        }
    } while(i<=j);
    if(l<j) qsort(l, j);
    if(i<r) qsort(i, r);
} // qsort()

static int orient(int x1,int y1,int x2,int y2,int x3,int y3)
{
    // modul(z3/2)=aria_triunghiului !!!
    long z1,z2,z3;
    int sens;
    z1=x1*y2+x2*y3+x3*y1;
    z2=y1*x2+y2*x3+y3*x1;
    z3=z1-z2;
    if(z3<0) sens=-1; // sens ace ceasornic
    else if(z3>0) sens=1; // sens trigonometric
    else sens=0; // coliniare
    return sens;
} // orient(...)

```

```

static void dinamic(double[] x,double[] sm,boolean[] fm,int pozi,int pozs)
{
    int i;

    for(i=pozi;i<=pozs;i++) fm[i]=false;

    sm[pozi]=x[pozi];
    fm[pozi]=true;

    if(x[pozi+1]>x[pozi]) { sm[pozi+1]=x[pozi+1]; fm[pozi+1]=true; }
        else sm[pozi+1]=x[pozi];

    for(i=pozi+2;i<=pozs;i++)
        if(sm[i-2]+x[i]>sm[i-1]){ sm[i]=sm[i-2]+x[i]; fm[i]=true; }
            else sm[i]=sm[i-1];
} // dinamic(...)

static double dist(int x1,int y1,int x2,int y2)
{
    double r1,r2,r3;
    r1=(double)(x2-x1)*(x2-x1);
    r2=(double)(y2-y1)*(y2-y1);
    r3=Math.sqrt(r1+r2);
    return r3;
} // dist(...)

static void convex()    / pune in stiva pucele poligonului convex
{
    int i;

    for(i=1;i<=n;i++) liber[i]=true;    // i nu este in infasuratoare
    st[1]=1;
    st[2]=2;
    liber[1]=false;
    liber[2]=false;

    vs=2;
    for(i=3;i<=n;i++)    // agat i la infasuratoare (pe dreapta)
    {
        while(orient(x[st[vs-1]],y[st[vs-1]],
                    x[st[vs]],y[st[vs]],
                    x[i],y[i])!=-1)    // invers trigonometric
        {
            liber[st[vs]]=true;

```

```

    vs--;
}
vs++;
st[vs]=i;
liber[i]=false;
} // n este sigur agatat la infasuratoare si este in stiva

for(i=n-1;i>=1;i--) // agat nodul i la infasuratoare (pe stanga)
if(liber[i])
{
    while(orient(x[st[vs-1]],y[st[vs-1]],
                x[st[vs]],y[st[vs]],
                x[i],y[i])==-1) // invers trigonometric
    {
        liber[st[vs]]=true;
        vs--;
    }
    vs++;
    st[vs]=i;
    liber[i]=false;
} // 1 este sigur agatat la infasuratoare si este in stiva
// 1 este si primul si ultimul in stiva !!!

for(i=1;i<=vs;i++)
{
    xc[i]= x[st[i]]; // de exemplu:
    yc[i]= y[st[i]]; // st[i]=3 si os[3]=5 ==> oc[i]=5
    oc[i]=os[st[i]]; // punctul i din infasuratoare este "5 initial"
}
} // convex
} // class

```

## 3.2 Lanterna - OJI 2004

Un agent secret are o hartă pe care sunt marcate  $N$  obiective militare. El se află, inițial, lângă obiectivul numerotat cu 1 (baza militară proprie) și trebuie să ajungă la obiectivul numerotat cu  $N$  (baza militară inamică). În acest scop, el va folosi drumurile existente, fiecare drum legând 2 obiective distincte. Fiind o misiune secretă, deplasarea agentului va avea loc noaptea; de aceea, el are nevoie de o lanternă. Pentru aceasta, el are de ales între  $K$  tipuri de lanterne - o lanternă de tipul  $W$  ( $1 \leq W \leq K$ ) are baterii care permit consumul a  $W$  wați, după consumul acestor wați, lanterna nu mai luminează. Din fericire, unele dintre obiective sunt

baze militare prietene, astfel că, o dată ajuns acolo, el își poate reîncărca bateriile complet. Agentul trebuie să aibă grijă ca, înainte de a merge pe un drum între două obiective, cantitatea de wași pe care o mai poate consuma să fie mai mare sau egală cu cantitatea de wași pe care o va consuma pe drumul respectiv.

Cunoscând drumurile dintre obiective și, pentru fiecare drum, durata necesară parcurgerii drumului și numărul de wași consumați de lanternă, determinați tipul de lanternă cu numărul cel mai mic, astfel încât durata deplasării să fie minimă (dintre toate tipurile de lanternă cu care se poate ajunge în timp minim la destinație, interesează lanternă cu consumul cel mai mic).

#### **Date de intrare**

Pe prima linie a fișierului **lanterna.in** se află numerele întregi  $N$  și  $K$ , separate printr-un spațiu.

Pe următoarea linie se află  $N$  numere întregi din mulțimea  $\{0, 1\}$ . Dacă al  $i$ -lea număr este 1, aceasta înseamnă că obiectivul cu numărul  $i$  este o bază militară prietenă (adică agentul își poate reîncărca bateriile lanternei dacă ajunge la acest obiectiv); dacă numărul este 0, agentul nu își va putea reîncărca bateriile. Primul număr din linie este 1, iar ultimul este 0.

Pe cea de-a treia linie a fișierului se află numărul  $M$  de drumuri dintre obiective.

Fiecare din următoarele  $M$  linii conține câte 4 numere întregi separate prin spații:  $a b T W$ , având semnificația că există un drum bidirecțional între obiectivele  $a$  și  $b$  ( $a \neq b$ ), care poate fi parcurs într-un timp  $T$  și cu un consum de  $W$  wași.

#### **Date de ieșire**

În fișierul **lanterna.out** se vor afișa două numere întregi, separate printr-un spațiu:  $Tmin$  și  $Wmin$ .  $Tmin$  reprezentând durata minimă posibilă a deplasării de la obiectivul 1 la obiectivul  $N$ , iar  $Wmin$  reprezintă tipul de lanternă cu numărul cel mai mic pentru care se obține acest timp.

#### **Restricții și precizări**

$$2 \leq N \leq 50$$

$$1 \leq K \leq 1000$$

$$1 \leq M \leq N * (N - 1) / 2$$

Între două obiective diferite poate exista maximum un drum direct.

Pentru fiecare drum, durata parcurgerii este un număr întreg între 1 și 100, iar numărul de wași consumați este un număr întreg între 0 și 1000

Se garantează că există cel puțin un tip de lanternă pentru care deplasarea să fie posibilă.

Punctajul pentru un test se va acorda în felul următor:

- 30% : dacă este determinat corect  $Tmin$

- 100% : dacă sunt determinate corect atât  $Tmin$ , cât și  $Wmin$

#### **Exemplu:**



lanterna.in	lanterna.out
7 10	27 6
1 0 1 0 0 0 0	
7	
1 2 10 3	
1 4 5 5	
2 3 10 3	
4 3 15 1	
3 6 4 3	
6 5 2 2	
5 7 1 0	

**Timp maxim de executare:** 1 secundă/test

### 3.2.1 Indicații de rezolvare - descriere soluție \*

*Mihai Stroe, Ginfo nr 14/4 - aprilie 2004*

Se cere tipul de lanternă cu consumul cel mai mic, dintre cele cu care se poate ajunge la destinație în timp minim.

Cerința sugerează ideea folosirii căutării binare. Astfel, inițial se încearcă ajungerea la destinație cu  $W_{min} = K$ . Dacă se reușește ajungerea la destinație cu timpul minim, se va încerca cu un  $W_{min}$  mai mic; dacă nu se reușește, sau timpul obținut este mai mare decât timpul minim, se va încerca un  $W_{min}$  mai mare. Evident, timpul minim  $T_{min}$  este obținut pentru  $W_{min} = K$ .

Acestea fiind stabilite, avem de rezolvat următoarea subproblemă: cum determinăm timpul minim în care putem ajunge la destinație pentru un anumit  $W_{min}$ ?

Construim o matrice  $A$  cu  $N$  linii și  $W_{min} + 1$  coloane (de la 0 la  $W_{min}$ ), în care  $A[i,j]$  semnifică timpul minim în care se poate ajunge din punctul de plecare în obiectivul  $i$ , având încă  $j$  Wați disponibili în lanternă pentru a continua mai departe.

Regulile de completare sunt următoarele:

- dacă  $i$  este un obiectiv prieten, de fiecare dată se va completa numai  $A[i, W_{min}]$ , deoarece lanternă poate fi reîncărcată;
- dacă am completat  $A[i,j]$  cu o valoare  $V$ , atunci pentru orice drum de la  $i$  la un alt obiectiv  $p$ , de lungime  $L$  și consum  $C$ , am putea pune valoarea  $V + L$  în  $A[p, j+C]$  dacă nu există deja o valoare mai mică. Evident, dacă  $j + C > W_{min}$  această opțiune nu este luată în considerare. Dacă  $p$  este un obiectiv prieten, se folosește regula de mai sus.

Pentru a nu suprascrise prea des valori, vom completa elementele matricei  $A$  în ordinea timpului în care se poate ajunge în orașe. Pentru aceasta folosim o coadă de priorități, care va conține elemente de tipul (obiectiv, wațiRămăși, timpDeAjungere), sortată după timpDeAjungere. Se extrage repetat elementul cu

timpul de ajungere minim și, folosind regulile prezentate, se adaugă noi elemente în coadă, dacă este cazul, completându-se și pozițiile corespunzătoare din matricea A.

Putem implementa coada de priorități cu ajutorul unui heap.

Dacă dorim să nu suprascriem deloc valori, combinăm acest algoritm cu algoritmul lui Dijkstra. Lăsăm această îmbunătățire ca exercițiu pentru cititor.

În final se vor afișa valorile cerute.

### Analiza complexității

Se execută  $\log K$  pași.

La fiecare pas se completează  $O(W \min N)$  poziții; pentru fiecare completare a unei poziții se execută  $O(\log W \min N)$  operații. Pentru acest calcul, considerăm împreună operațiile de introducere, respectiv extragere ale unui anumit element în/din coada de priorități.

În concluzie, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este  $O(\log K \log(W \min N)(W \min N))$ .

În practică, nu toate pozițiile din matrice sunt completate, ceea ce face ca algoritmul să ruleze foarte repede.

## 3.2.2 Rezolvare detaliată

### 3.2.3 Codul sursă \*

```
import java.io.*;
class lanterna // Cautare binara + Bellman Ford (cu coada)
{
    static StreamTokenizer st;
    static PrintWriter out;

    static final int NMAX=50;
    static final int KMAX=1000;
    static final int CMAX=1002; // dim coada pentru test3 !!!
    static final int oo=999999999;

    static int[] [] g=new int[NMAX+1][NMAX+1]; // graf - "liste" de adiacenta
    static int[] [] t=new int[NMAX+1][NMAX+1]; // timp cu "matrice" de adiacenta
    static int[] [] w=new int[NMAX+1][NMAX+1]; // wati cu "matrice" de adiacenta

    static int[] p=new int[NMAX+1]; // p[i]=1 ==> i=prieten

    static int[] [] tm=new int[NMAX+1][KMAX+1];
```

```
static boolean[][] nuesteutil=new boolean[NMAX+1][KMAX+1];

static int[] cn=new int[CMAX];           // coada nod
static int[] cw=new int[CMAX];           // coada wati consumati

static int n,k,m;
static int tmink,wmin;

static int nri=0,nre=0, nrmax=0;

public static void main(String[] args) throws IOException
{
    citire();
    sol();
    scrieSOL();
    //System.out.println("tmink = "+tmink+"  wmin = "+wmin);
}

static void scrieSOL() throws IOException
{
    out=new PrintWriter(new BufferedWriter(new FileWriter("lanterna.out")));
    out.println(tmink+" "+wmin);
    out.close();
}

static int timpmin(int wmax)
{
    int ic,sc,i,j,i0,j0,ii,vi0,wvi0;
    int tvi0,tmin=oo;

    for(i=0;i<=n;i++)
        for(j=0;j<=wmax;j++)
        {
            tm[i][j]=oo;
        }
    tm[1][0]=0; // start din 1

    cn[0]=1; // nodul 1 in coada
    cw[0]=0; // consum 0 wati la start
    ic=0;
    sc=1; // prima pozitie libera

    nri=0; nre=0; ++nri;
```

```
while(ic!=sc)          // coada nevida
{
    i0=cn[ic];         // scot un nod din coada
    j0=cw[ic];
    ic=(1+ic)%CMAX;   // coada circulara !!!

    nre--;

    if(nuesteutil[i0][j0])
    {
        //System.out.println("Inutil sa propage ... "+i0+","+j0+");
        continue;
    }

    // propag catre vecinii lui i0
    for(ii=1;ii<=g[i0][0];ii++)
    {
        vi0=g[i0][ii];           // vecinul lui i0
        wvi0=j0+w[i0][vi0];
        tvi0=tm[i0][j0]+t[i0][vi0];

        if(wvi0>wmax)
        {
            //System.out.println("Nu ajung watii ...");
            continue;
        }

        if(tvi0>tmin)
        {
            //System.out.println("Deja este un timp prea mare ...");
            continue;
        }

        if(p[vi0]==1)
        {
            wvi0=0;
            //System.out.println("Este prieten ...");
        }

        if(vi0==n) // tmin ca sa nu cau min pe linia n
        {
            if(tvi0<tmin) tmin=tvi0;
            tm[vi0][wvi0]=tvi0;           // pentru depanare !
            //System.out.println("Am ajuns la destinatie ... pe un drum ...");
        }
    }
}
```

```

    }
    else
    { // vi0 != n
      if(tvi0<tm[vi0][wvi0]) // timp mai mic ...
      {
        //System.out.print("Timp mai mic ... ");
        tm[vi0][wvi0]=tvi0;
        cn[sc]=vi0; // pun in coada vi0
        cw[sc]=wvi0;
        sc=(1+sc)%CMAX;

        nri++;
        if(nri-nre>nrmax) nrmax=nri-nrmax;

        nuesteutil[vi0][wvi0]=false;

        // optimizare ...
        for(j=wvi0+1;j<=wmax;j++)
          if(tm[vi0][j]>tvi0)
          {
            tm[vi0][j]=tvi0;
            // il marchez "inutil"...
            // daca este in coada ... sa nu mai propage aiurea !
            nuesteutil[vi0][j]=true;
          }
          else
          {
            break;
          }

        }// if
        else
        {
          //System.out.println("Timp mai mare sau egal ...");
        }
        }// j!=n
    }//for ii

  }//while

  return tmin;
}//timpmin

static void citire() throws IOException

```

```

{
  int i,j,ij,tt,ww;

  st=new StreamTokenizer(new BufferedReader(new FileReader("lant10.in")));

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

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

  st.nextToken(); m=(int)st.nval;
  for(ij=1;ij<=m;ij++)
  {
    st.nextToken(); i=(int)st.nval;
    st.nextToken(); j=(int)st.nval;
    st.nextToken(); tt=(int)st.nval;
    st.nextToken(); ww=(int)st.nval;

    ++g[i][0]; g[i][g[i][0]]=j;          // lista de adiacenta
    ++g[j][0]; g[j][g[j][0]]=i;

    t[i][j]=t[j][i]=tt;                // matrice de adiacenta
    w[i][j]=w[j][i]=ww;
  }
} // citire(...)

static void sol()
{
  int wi,ws,wm;

  tmink=timpmin(k);

  wmin=k;
  wi=1;
  ws=k-1;
  while(wi<=ws)          // cautare binara
  {
    wm=(wi+ws)>>1;      // impartire la 2 ...
    if(timpmin(wm)==tmink)
    {
      wmin=wm;
    }
  }
}

```

```
        ws=wm-1;
    }
    else wi=wm+1;
}
} // sol()
} // class
```





## Capitolul 4

# OJI 2005 clasa a XI-a

### 4.1 Lanț - OJI 2005

*Emanuela Cerchez*

Ion este un lingvist pasionat. Recent el a descoperit un text scris într-o limbă necunoscută. Textul este scris pe mai multe linii și este format din cuvinte scrise cu litere mici din alfabetul latin, separate prin spații sau/și semne de punctuație (,; !? -).

Ion a fost frapat că există multe similitudini între cuvintele din text. Fiind foarte riguros, Ion definește similitudinea a două cuvinte după cum urmează.

Fie  $c_1$  și  $c_2$  două cuvinte. Cuvântul  $c_1$  poate fi obținut din cuvântul  $c_2$  printr-o succesiune de operații elementare. Operațiile elementare ce pot fi folosite sunt:

Operația	Efect	Exemplu
$move(c_1, c_2)$	Mută primul caracter din $c_1$ la sfârșitul lui $c_2$	Dacă $c_1 = \text{"alba"}$ și $c_2 = \text{"neagra"}$ , după efectuarea operației $c_1$ va fi $\text{"lba"}$ , iar $c_2$ va fi $\text{"neagraa"}$
$insert(c_1, x)$	Inserează caracterul $x$ la începutul lui $c_1$	Dacă $c_1 = \text{"alba"}$ și $x = \text{"b"}$ , după efectuarea operației $c_1$ va fi $\text{"balba"}$
$delete(c_1)$	Șterge primul caracter din $c_1$	Dacă $c_1 = \text{"alba"}$ după efectuarea operației $c_1$ va fi $\text{"lba"}$

Definim similitudinea dintre  $c_1$  și  $c_2$  ca fiind numărul **minim** de operații  $insert$  și  $delete$  ce trebuie să fie executate pentru a transforma cuvântul  $c_1$  în cuvântul  $c_2$  (operațiile  $move$  nu se numără).

Fie  $c_0$  primul cuvânt din text. Începând cu  $c_0$  putem construi lanțuri de  $k$ -similitudine.

Un lanț de  $k$ -similitudine este o succesiune de cuvinte distincte din text cu următoarele proprietăți:

- dacă cuvântul  $x$  apare în lanț înaintea cuvântului  $y$ , atunci prima apariție a lui  $x$  în text precedă prima apariție a lui  $y$  în text;
- dacă  $x$  și  $y$  sunt cuvinte consecutive în lanț (în ordinea  $x y$ ), atunci similitudinea dintre  $x$  și  $y$  este  $k$ ;
- lanțul este maximal (adică nu putem adăuga încă un cuvânt la sfârșitul acestui lanț, astfel încât să fie respectate proprietățile precedente).

**Cerință**

Scrieți un program care să determine numărul de lanțuri de  $k$ -similitudine care încep cu  $c0$ .

**Date de intrare**

Fișierul de intrare **lant.in** conține pe prima linie valoarea  $k$ . Pe următoarele linii se află textul dat.

**Date de ieșire**

Fișierul de ieșire **lant.out** va conține o singură linie pe care va fi scris numărul de lanțuri de  $k$ -similitudine care încep cu  $c0$ .

**Restricții**

Lungimea unei linii din text nu depășește 1000 de caractere.

Lungimea unui cuvânt nu depășește 30 de caractere.

Numărul total de cuvinte  $\leq 150$ .

Pentru datele de test, numărul de lanțuri de  $k$ -similitudine care încep cu  $c0$  va fi  $\leq 2.000.000.000$ .

**Exemplu**

lant.in	lant.out	Explicație
5 ana are mere, banane, pere si castane.	6	Lanțurile de 5-similitudine care se pot forma sunt: ana are mere pere ana are pere ana are banane castane ana are si ana banane castane ana si

**Timp maxim de execuție/test:** 1 secundă.

**4.1.1 Indicații de rezolvare - descriere soluție \****Soluția oficială*

1. Se citește textul și se memorează cuvintele distincte din text într-un tablou  $c$ . Fie  $nc$  numărul de cuvinte distincte determinate. Fiecare cuvânt este numerotat

de la 0 la  $nc-1$  (indicii din tabloul  $c$ ). Observați că numerotarea cuvintelor respectă ordinea primei apariții în text a cuvintelor.

2. Asociem problemei un graf orientat astfel:

– nodurile grafului sunt cuvintele distincte din text;

– există arc de la nodul  $i$  la nodul  $j$  ( $i < j$ ) dacă numărul minim de operații

**insert** și **delete** necesare pentru a transforma cuvântul  $c[i]$  în cuvântul  $c[j]$  este  $\leq k$ .

Observați că grafurile asociate problemei nu conțin circuite.

Pentru a determina arcele grafului trebuie să rezolvăm următoarea sub-problemă: să se determine numărul minim de operații **delete** și **insert** necesare pentru a transforma cuvântul  $x$  în cuvântul  $y$ .

Rezolvăm această subproblemă prin *programare dinamică*.

Fie  $d[i][j]$  = numărul minim de operații **insert** și **delete** necesare pentru a transforma sufixul lui  $x$  care începe la poziția  $i$  în sufixul lui  $y$  care începe la poziția  $j$ .

Fie  $n$  = lungimea cuvântului  $x$  și  $m$  = lungimea cuvântului  $y$ .

•  $d[n][j] = m - j$ , pentru orice  $j = 0, m$

•  $d[i][m] = n - i$ , pentru orice  $i = 0, n$

•  $d[i][j] = \min\{d[i+1][j+1], \text{dacă } p[i] == q[j]; - \text{move}$   
 $1 + d[i][j+1] - \text{insert}$   
 $1 + d[i+1][j] - \text{delete} \}$

Soluția este  $d[0][0]$ .

3. Numărul de lanțuri de  $k$ -similitudine este egal cu numărul de drumuri care încep cu nodul 0 și se termină într-un nod terminal al grafului (nod cu gradul exterior 0).

Să notăm:  $nr[i]$  = numărul de lanțuri de  $k$ -similitudine care încep cu cuvântul  $i$ .

Determinăm numărul folosind următoarea relație de recurență:

•  $nr[i] = 1$ , dacă nodul  $i$  este terminal

•  $nr[i] = nr[i_1] + nr[i_2] + \dots + nr[i_k]$ , unde  $i_1, i_2, \dots, i_k$  sunt noduri din graf cu proprietatea că există arc de la  $i$  la  $i_j$ , pentru  $j = 1, k$ .

#### 4.1.2 Rezolvare detaliată

#### 4.1.3 Codul sursă \*

Prima variantă:

```
import java.io.*;
class Lant1
{
```

```
static final int NMAX=150;
static String[] c=new String[NMAX+1];
static int k,nc=0,nlks=0;
static int[][] cs=new int[NMAX+1][NMAX+1]; // costuri editare
static int[] x=new int[NMAX+1];           // solutie in recursivitate

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

    citire();
    solutie();
    scrieSol();

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

static void citire() throws IOException
{
    StreamTokenizer st=new StreamTokenizer(
        new BufferedReader(new FileReader("lant.in")));

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

    st.whitespaceChars(' ',' ',' ');
    st.whitespaceChars(':',':',' ');
    st.whitespaceChars(';',' ',' ');
    st.whitespaceChars('.','.',' ');
    st.whitespaceChars('!',' ','!');
    st.whitespaceChars('?',' ','?');
    st.whitespaceChars('-', '-');

    while(StreamTokenizer.TT_EOF!=st.nextToken())
    {
        c[++nc]=st.sval;
        if(c[nc]==null) {--nc;continue;} // "EOL" returneaza "null"
        if(primaPozitie(c[nc])<nc) nc--;
        else System.out.println(nc+" : "+c[nc]);
    }
    System.out.println();
} // citire()
```

```

static void solutie()
{
    int i,j;

    for(i=1;i<=nc-1;i++)
        for(j=i+1;j<=nc;j++)
            cs[i][j]=dist(c[i],c[j]);

    afism(cs);      // matricea costurilor similitudine

    x[1]=1;        // lanturi care incep cu primul cuvant
    for(j=2;j<=nc;j++)
        if(cs[1][j]<=k) cuvant(2,j);    //(pozitie,cuvant)

    System.out.println();
}// solutie()

static int dist(String x, String y)
{
    int i,j,n,m;
    n=x.length();
    m=y.length();
    int [][] d=new int[n+1][m+1]; // este initializata cu zero
    for(i=0;i<=n;i++) d[i][m]=n-i;
    for(j=0;j<=m;j++) d[n][j]=m-j;

    for(i=n-1;i>=0;i--)
        for(j=m-1;j>=0;j--)
        {
            d[i][j]=min(1+d[i][j+1],1+d[i+1][j]);
            if(x.charAt(i)==y.charAt(j)) d[i][j]=min(d[i][j],d[i+1][j+1]);
        }
    return d[0][0];
}// dist()

static void cuvant(int pozi, int cuvi) // plasez cuvi pe pozi si ...
{
    int i,cuvj;
    boolean esteMaximal=true;

    x[pozi]=cuvi;
    for(cuvj=cuvi+1;cuvj<=nc;cuvj++)
        if(cs[cuvi][cuvj]<=k)
            {

```

```
        esteMaximal=false;
        cuvant(pozi+1,cuvj);
    }

    if(esteMaximal)
    {
        nlks++;
        System.out.print(nlks+" : ");
        for(i=1;i<=pozi;i++) System.out.print(c[x[i]]+" ");
        System.out.println();
    }
} // cuvant(...)

static int primaPozitie(String s) // in care apare s in text
{
    int i,j,pp=-1,ns=s.length();
    boolean ok=false;
    for(i=1;i<nc;i++)
    {
        pp=i;
        if(c[i].length()!=ns) continue; // difera prin lungime
        ok=true;
        for(j=0;j<ns;j++) // s[0] ... s[ns-1]
            if(c[i].charAt(j)!=s.charAt(j)) { ok=false; break;}
        if(ok) break;
    }
    if(ok) return pp; else return nc;
} // primaPozitie(...)

static int min(int a, int b) { if(a<b) return a; else return b; }

static void afism(int[][] a)
{
    int i,j;
    for(i=1;i<=nc;i++)
    {
        for(j=1;j<=nc;j++) System.out.print(a[i][j]+" ");
        System.out.println();
    }
    System.out.println();
} // afism(...)

static void scrieSol() throws IOException
{
```

```

    PrintWriter out = new PrintWriter(
        new BufferedWriter(new FileWriter("lant.out")));
    out.println(nlks);
    out.close();
} // scrieSol()
} // class
/*-----
Lant.in      5
              ana are mere,banane,
              pere si castane.

Lant.out:    6

Ecran:

1 : ana
2 : are
3 : mere
4 : banane
5 : pere
6 : si
7 : castane

0 1 0 3 0 0 2      0 4 7 3 7 5 6      1 : ana are mere pere
0 0 2 2 2 0 2      0 0 3 5 3 5 6      2 : ana are banane castane
0 0 0 1 3 0 1      0 0 0 8 2 6 9      3 : ana are pere
0 0 0 0 1 0 4      0 0 0 0 8 8 5      4 : ana are si
0 0 0 0 0 0 1      0 0 0 0 0 6 9      5 : ana banane castane
0 0 0 0 0 0 1      0 0 0 0 0 0 7      6 : ana si
0 0 0 0 0 0 0      0 0 0 0 0 0 0
-----*/

```

A doua variantă:

```

import java.io.*;
class Lant2
{
    static final int NMAX=150;
    static String[] c=new String[NMAX+1];
    static int k,nc=0;
    static int[][] cs=new int[NMAX+1][NMAX+1]; // costuri editare
    static int[] x=new int[NMAX+1]; // x[i]=nr cuvinte care incep cu c[i]

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

```

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

citire();
solutie();
scrieSol();

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

static void citire() throws IOException
{
    StreamTokenizer st=new StreamTokenizer(
        new BufferedReader(new FileReader("lant.in")));

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

    st.whitespaceChars(' ',' ',' ');
    st.whitespaceChars(':',':',' ');
    st.whitespaceChars(';',' ',' ');
    st.whitespaceChars('.','.',' ');
    st.whitespaceChars('!',' ',' ');
    st.whitespaceChars('?',' ',' ');
    st.whitespaceChars('-', '-',' ');

    while(StreamTokenizer.TT_EOF!=st.nextToken())
    {
        c[++nc]=st.sval;
        if(c[nc]==null) {--nc;continue;} // "EOL" returneaza "null"
        if(primaPozitie(c[nc])<nc) nc--;
    }
} // citire()

static void solutie()
{
    int i,j;
    boolean esteTerminal;

    for(i=1;i<=nc-1;i++)
        for(j=i+1;j<=nc;j++)
            cs[i][j]=dist(c[i],c[j]);

    for(i=2;i<=nc;i++)
```



```

    {
        esteTerminal=true;
        for(j=i+1;j<=nc;j++)
            if(cs[i][j]<=k) {esteTerminal=false; break;}
        if(esteTerminal) x[i]=1;
    }

    for(i=nc-1;i>=1;i--)
        for(j=i+1;j<=nc;j++)
            if(cs[i][j]<=k) x[i]+=x[j];
} // solutie()

static int dist(String x, String y)
{
    int i,j,n,m;
    n=x.length();
    m=y.length();
    int[][] d=new int[n+1][m+1]; // este initializata cu zero
    for(i=0;i<=n;i++) d[i][m]=n-i;
    for(j=0;j<=m;j++) d[n][j]=m-j;

    for(i=n-1;i>=0;i--)
        for(j=m-1;j>=0;j--)
        {
            d[i][j]=min(1+d[i][j+1],1+d[i+1][j]);
            if(x.charAt(i)==y.charAt(j)) d[i][j]=min(d[i][j],d[i+1][j+1]);
        }
    return d[0][0];
} // dist(...)

static int primaPozitie(String s) // in care apare s in text
{
    int i,j,pp=-1,ns=s.length();
    boolean ok=false;
    for(i=1;i<nc;i++)
    {
        pp=i;
        if(c[i].length()!=ns) continue; // difera prin lungime
        ok=true;
        for(j=0;j<ns;j++) // s[0] ... s[ns-1]
            if(c[i].charAt(j)!=s.charAt(j)) { ok=false; break;}
        if(ok) break;
    }
    if(ok) return pp; else return nc;
}

```

```

} // primaPozitie(...)

static int min(int a, int b) { if(a<b) return a; else return b; }

static void scrieSol() throws IOException
{
    PrintWriter out = new PrintWriter(
        new BufferedWriter(new FileWriter("lant.out")));
    out.println(x[1]);
    out.close();
} // scrieSol()
} // class

```

## 4.2 Scara - OJI 2005

*Stelian Ciurea*

Domnul  $G$  are de urcat o scară cu  $n$  trepte. În mod normal, la fiecare pas pe care îl face, el urcă o treaptă. Pe  $k$  dintre aceste trepte se află câte o sticlă cu un număr oarecare de decil litri de apă, fie acesta  $x$ . Dacă bea toată apa dintr-o astfel de sticlă, forța și mobilitatea lui  $G$  cresc, astfel încât, la următorul pas el poate urca până la  $x$  trepte, după care, dacă nu bea din nou ceva, revine la "normal". Sticlele cu apă nu costă nimic. Cantitatea de apă conținută de aceste sticle poate să difere de la o treaptă la alta.

Pe  $j$  trepte se află câte o sticlă cu băutură energizantă. Și pentru aceste sticle, cantitatea de băutură energizantă poate să difere de la o treaptă la alta. Să presupunem că într-una dintre aceste sticle avem  $y$  decil litri de băutură energizantă. Dacă bea  $q$  ( $q \leq y$ ) decil litri dintr-o astfel de sticlă, la următorul pas  $G$  poate urca până la  $2q$  trepte, după care și în acest caz, dacă nu bea din nou ceva, el revine la "normal". Însă băutura energizantă costă: pentru o cantitate de  $q$  decil litri consumați,  $G$  trebuie să plătească  $q$  lei grei.

Pot exista trepte pe care nu se află nici un pahar, dar și trepte pe care se află atât o sticlă cu apă cât și una cu băutură energizantă. În astfel de situații, nu are rost ca  $G$  să bea ambele băuturi deoarece efectul lor nu se cumulează; el poate alege să bea una dintre cele două băuturi sau poate să nu bea nimic.

### Cerință

Determinați  $p$ , numărul minim de pași pe care trebuie să îi facă  $G$  pentru a urca scara, precum și suma minimă pe care trebuie să o cheltuiască  $G$  pentru a urca scara în  $p$  pași.

### Date de intrare

Fișierul text de intrare **scara.in** conține:

- pe prima linie un număr natural  $n$ , reprezentând numărul total de trepte;

- pe cea de a doua linie un număr natural  $k$ , reprezentând numărul de trepte pe care se află sticle cu apă;

- pe fiecare dintre următoarele  $k$  linii câte două numere naturale separate printr-un spațiu, reprezentând numărul de ordine al treptei pe care se află o sticlă cu apă și respectiv cantitatea de apă din acea sticlă exprimată în decilitri;

- pe următoarea linie un număr natural  $j$ , reprezentând numărul de trepte pe care se află sticle cu băutură energizantă;

- pe fiecare dintre următoarele  $j$  linii câte două numere naturale separate printr-un spațiu, reprezentând numărul de ordine al treptei pe care se află o sticlă cu băutură energizantă și respectiv cantitatea de băutură energizantă din acea sticlă exprimată în decilitri.

### Date de ieșire

Fișierul text de ieșire **scara.out** va conține o singură linie pe care vor fi scrise două numere naturale  $p$   $c$  separate printr-un spațiu,  $p$  reprezentând numărul minim de pași, iar  $c$  suma minimă cheltuită.

### Restricții

$$n \leq 120$$

$$0 \leq k \leq n$$

$$0 \leq j \leq n$$

Cantitatea de apă aflată în oricare sticlă este  $1 \leq x \leq 100$

Cantitatea de băutură energizantă aflată în oricare sticlă este  $1 \leq y \leq 100$

### Exemple

scara.in	scara.out	scara.in	scara.out
6	3 2	6	4 1
1		1	
1 2		1 2	
2		2	
4 1		4 1	
1 2		1 1	

**Timp maxim de execuție:** 1 secundă/test.

#### 4.2.1 Indicații de rezolvare - descriere soluție \*

##### Soluția oficială

Se construiește un graf orientat cu  $n$  noduri (care corespund celor  $n$  trepte) la arcele căruia se atașează costuri urmărind ca:

– între două noduri costul să fie cu atât mai mic cu cât nodurile sunt ”mai depărtate”

– costul pentru salturi efectuate după ce se bea băutura energizantă să fie mai mare decât costul salturilor făcute după ce se bea apă între aceleași noduri, dar acest cost să nu depășească costul saltului între două noduri mai apropiate.

În calculul costurilor între două noduri am folosit formulele:

$$\text{cost}[i][i+1] = 999000$$

$$\text{cost}[i][j] = 1000000 - (j - i) \text{ pentru salturi când se bea apă } j > i$$

$$\text{cost}[i][j] = 1000000 - (j - i) + q \text{ pentru salturi când se bea energizantă } j > i,$$

$$1 \leq q \leq y$$

Pe acest graf se aplică apoi un algoritm de aflare a drumului minim de sursă unică (Dijkstra) sau având în vedere caracterul aciclic al grafului rezultat, algoritmul Dijkstra adaptat pentru astfel de grafuri.

## 4.2.2 Rezolvare detaliată

### 4.2.3 Codul sursă \*

Varianta 1a:

```
import java.io.*;    // numai nr pasi !
class Scarala
{
    static int n;
    static int[] a;    // apa
    static int[] e;    // energizant
    static int[] p;    // nr pasi
    static int[] c;    // cost

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

        int k,j;
        int i,treapta, cantitate;
        int pmin;
        StringTokenizer st=new StringTokenizer(
            new BufferedReader(new FileReader("scara.in")));
        PrintWriter out=new PrintWriter(
            new BufferedWriter(new FileWriter("scara.out")));

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

```

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

st.nextToken(); j=(int)st.nval;
for(i=1;i<=j;i++)
{
    st.nextToken(); treapta=(int)st.nval;
    st.nextToken(); cantitate=(int)st.nval;
    e[treapta]=cantitate;
}

p[1]=1;
for(k=2;k<=n;k++)
{
    pmin=k;
    for(i=1;i<=k-1;i++)
    {
        pmin=min(pmin,p[i]+(k-i));    // in cel mai rau caz !
        if(i+a[i]>=k) pmin=min(pmin,p[i]+1);
        if(i+2*e[i]>=k) pmin=min(pmin,p[i]+1);
    }
    p[k]=pmin;
}
out.println(p[n]);
out.close();

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

static int min(int a, int b)
{
    if(a<b) return a; else return b;
}
} // class

```

Varianta 1b:

```
import java.io.*;    // nr pasi si cost ... cu programare dinamica !
class Scara1b       // traseul = !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
{
    static int n;
    static int[] a;    // apa
    static int[] e;    // energizant
    static int[] p;    // nr pasi
    static int[] c;    // cost

    public static void main(String[] args) throws IOException
    {
        int k,j;
        int i,treapta, cantitate;
        int pmin,cmin;
        StreamTokenizer st=new StreamTokenizer(
            new BufferedReader(new FileReader("scara.in")));
        PrintWriter out=new PrintWriter(
            new BufferedWriter(new FileWriter("scara.out")));

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

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

        st.nextToken(); j=(int)st.nval;
        for(i=1;i<=j;i++)
        {
            st.nextToken(); treapta=(int)st.nval;
            st.nextToken(); cantitate=(int)st.nval;
            e[treapta]=cantitate;
        }

        p[1]=1;
        c[1]=0;
        for(k=2;k<=n;k++)
```

```

{
    pmin=p[k-1]+1;    // in cel mai rau caz !
    cmin=c[k-1];     // in cel mai rau caz !

    for(i=1;i<=k-1;i++)
    {
        pmin=min(pmin,p[i]+(k-i));    // in cel mai rau caz !

        if(i+a[i]>=k)
            if(p[i]+1<pmin) {pmin=p[i]+1; cmin=c[i];}    // apa=free !
            else if((p[i]+1==pmin)&&(a[i]>0)) cmin=min(cmin,c[i]);

        if(i+2*e[i]>=k)
            if(p[i]+1<pmin) { pmin=p[i]+1; cmin=c[i]+(k-i+1)/2;}
            else if((p[i]+1==pmin)&&(e[i]>0)) cmin=min(cmin,c[i]+(k-i+1)/2);
        }
        p[k]=pmin;
        c[k]=cmin;
    }

    out.println(p[n]+" "+c[n]);
    out.close();
} // main

static int min(int a, int b)
{
    if(a<b) return a; else return b;
}
} // class

```

Varianta 2a:

```

import java.io.*;           // cu "propagare" numai nr pasi
class Scara2a
{
    static int n;
    static int[] a;         // apa
    static int[] e;         // energizant
    static int[] p;         // nr pasi
    static int[] c;         // cost

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

```

```

t1=System.currentTimeMillis();

int k,j,i,treapta, cantitate,pmin,cmin;
StreamTokenizer st=new StreamTokenizer(
    new BufferedReader(new FileReader("scara.in")));
PrintWriter out=new PrintWriter(
    new BufferedWriter(new FileWriter("scara.out")));

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

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

st.nextToken(); j=(int)st.nval;
for(i=1;i<=j;i++)
{
    st.nextToken(); treapta=(int)st.nval;
    st.nextToken(); cantitate=(int)st.nval;
    e[treapta]=cantitate;
}

for(k=1;k<=n;k++) p[k]=k+1;    // initializare pentru min !!

p[1]=1; c[0]=0;
for(k=1;k<=n-1;k++)    // propag informatia de pe treapta k in sus !
{
    if(a[k]>0)
        for(i=k+1;i<=min(k+a[k],n);i++)    // propag apa in sus !
            if(p[k]+1<p[i]) p[i]=p[k]+1;

    for(i=k+a[k]+1;i<=n;i++)    // pas cu pas !
        if(p[i-1]+1<p[i]) p[i]=p[i-1]+1;

    if(e[k]>0)
        for(j=k+1;j<=min(k+2*e[k],n);j++)    // propag energizant in sus

```



```

        if(p[k]+1<p[j]) p[j]=p[k]+1;

        for(j=k+2*e[k]+1;j<=n;j++)          // pas cu pas !
            if(p[j-1]+1<p[j]) p[j]=p[j-1]+1;
    }

    out.println(p[n]+" "+c[n]); out.close();
    t2=System.currentTimeMillis(); System.out.println("Timp = "+(t2-t1));
}// main

static int min(int a, int b)
{
    if(a<b) return a; else return b;
}
}// class

```

Varianta 2b:

```

import java.io.*;    // cu "propagare" pasi si cost
class Scara2b       // traseul = !!!!!!!!!!!!!!!!!!!!!
{
    static int n;
    static int[] a;    // apa
    static int[] e;    // energizant
    static int[] p;    // nr pasi
    static int[] c;    // cost

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

        int k,j,i,treapta,cantitate,pmin,cmin;
        StreamTokenizer st=new StreamTokenizer(
            new BufferedReader(new FileReader("scara.in")));
        PrintWriter out=new PrintWriter(
            new BufferedWriter(new FileWriter("scara.out")));

        st.nextToken(); n=(int)st.nval;
        a=new int[n+1];
        e=new int[n+1];
        p=new int[n+1];
        c=new int[n+1];
    }
}

```

```

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

st.nextToken(); j=(int)st.nval;

for(i=1;i<=j;i++)
{
    st.nextToken(); treapta=(int)st.nval;
    st.nextToken(); cantitate=(int)st.nval;
    e[treapta]=cantitate;
}

for(k=1;k<=n;k++) p[k]=k+1; // initializare pentru min !!

p[1]=1; c[0]=0;
for(k=1;k<=n-1;k++) // propag informatia de pe treapta k in sus !
{
    if(a[k]>0)
    for(i=k+1;i<=min(k+a[k],n);i++) // propag apa in sus
        if(p[k]+1<p[i]) { p[i]=p[k]+1; c[i]=c[k]; } //apa=free
        else
            if(p[k]+1==p[i]) c[i]=min(c[i],c[k]);

    for(i=k+a[k]+1;i<=n;i++) // pas cu pas !
        if(p[i-1]+1<p[i]) { p[i]=p[i-1]+1; c[i]=c[i-1];}
        else
            if(p[i-1]+1==p[i]) c[i]=min(c[i],c[i-1]); // ???

    if(e[k]>0)
    for(j=k+1;j<=min(k+2*e[k],n);j++) // propag energizant in sus
        if(p[k]+1<p[j]) { p[j]=p[k]+1; c[j]=c[k]+(j-k+1)/2;}
        else
            if(p[k]+1==p[j]) c[j]=min(c[j],c[k]+(j-k+1)/2);

    for(j=k+2*e[k]+1;j<=n;j++) // pas cu pas !
        if(p[j-1]+1<p[j]) {p[j]=p[j-1]+1; c[j]=c[j-1];}
        else
            if(p[j-1]+1==p[j]) c[j]=min(c[j],c[j-1]);
}

```

```
        out.println(p[n]+" "+c[n]); out.close();
        t2=System.currentTimeMillis(); System.out.println("Timp = "+(t2-t1));
    }// main

    static int min(int a, int b)
    {
        if(a<b) return a; else return b;
    }
}// class
```



# Capitolul 5

## OJI 2006 clasa a XI-a

### 5.1 Graf - OJI 2006

*Victor Manz*

Se știe că într-un graf neorientat conex, între oricare două vârfuri există cel puțin un lanț iar lungimea unui lanț este egală cu numărul muchiilor care-l compun. Definim noțiunea *lanț optim între două vârfuri X și Y* ca fiind un lanț de lungime minimă care are ca extremități vrfurile X și Y. Este evident că între oricare două vârfuri ale unui graf conex vom avea unul sau mai multe lanțuri optime, depinzând de configurația grafului.

#### **Cerință:**

Fiind dat un graf neorientat conex cu N vârfuri etichetate cu numerele de ordine 1,2, ...,N și două vârfuri ale sale notate X și Y ( $1 \leq X, Y \leq N, X \neq Y$ ), se cere să scrieți un program care determină vârfurile care aparțin tuturor lanțurilor optime dintre X și Y.

#### **Date de intrare:**

Fișierul **graf.in** conține

- pe prima linie patru numere naturale reprezentând: N (numărul de vârfuri ale grafului), M (numărul de muchii), X și Y (cu semnificația din enunț).

- pe următoarele M linii câte două numere naturale nenule  $A_i, B_i$  ( $1 \leq A_i, B_i \leq N, A_i \neq B_i$ , pentru  $1 \leq i \leq M$ ) fiecare dintre aceste perechi de numere reprezentând câte o muchie din graf.

#### **Date de ieșire**

Fișierul **graf.out** va conține

- pe prima linie, numărul de vârfuri comune tuturor lanțurilor optime dintre X și Y;

- pe a doua linie, numerele corespunzătoare etichetelor acestor vârfuri, dispuse în ordine crescătoare; între două numere consecutive de pe această linie se va afla

câte un spațiu.

### Restricții

- $2 \leq N \leq 7500; 1 \leq M \leq 14000$
- pentru 50% din teste  $N \leq 200$

### Exemple

graf.in	graf.out	graf.in	graf.out
6 7 1 4	3	3 2 1 3	2
1 2	1 4 5	1 2	1 3
1 3		3 1	
1 6			
2 5			
3 5			
5 6			
5 4			

**Timp maxim de execuție/test:** 1 secundă.

#### 5.1.1 Indicații de rezolvare - descriere soluție \*

##### *Soluția oficială*

I) Dacă definim distanța între două vârfuri ale unui graf neorientat ca fiind lungimea celui mai scurt lanț dintre lanțurile care au drept capete vârfurile, atunci putem să observăm că un vârf oarecare  $Z$  se află pe un lanț de lungime minimă dintre  $X$  și  $Y$  dacă și numai dacă  $d(X, Z) + d(Z, Y) = d(X, Y)$ , pentru cazul în care considerăm lungimea lanțului ca fiind numărul muchiilor și  $d(X, Z) + d(Z, Y) = d(X, Y) + 1$ , pentru cazul în care considerăm lungimea ca fiind numărul vârfurilor.

Stabilim prin câte o parcurgere în lățime distanțele tuturor vârfurilor față de  $X$  și respectiv  $Y$  (capetele lanțului, citite din fișier). Vedem care dintre vârfurile ce aparțin cel puțin unui lanț de lungime minimă între  $X$  și  $Y$  au proprietatea că sunt singurele aflate la o anumită distanță de  $X$ . Acestea sunt vârfurile care aparțin tuturor lanțurilor de lungime minimă dintre  $X$  și  $Y$ .

Algoritmul are complexitate  $O(n + m)$ .

II) Facem o parcurgere în lățime din  $X$  și o parcurgere în lățime pornind din  $Y$ , în urma căreia determinăm pentru fiecare vârf  $z$  distanța dintre  $X$  și  $z$ , și  $Y$  și  $z$  - notate  $d(X, z)$  și respectiv  $d(Y, z)$  și numărul de drumuri optime dintre  $X$  și  $z$ , notat  $nr(X, z)$  și dintre  $Y$  și  $z$  -  $nr(Y, z)$ ). Un vârf  $z$  are proprietatea de a aparține tuturor drumurilor optime dintre  $X$  și  $Y$  dacă și numai dacă:

$$d(X, z) + d(Y, z) = d(X, Y) \quad \text{și} \quad nr(X, z) * nr(Y, z) = nr(X, Y).$$

Această soluție este însă dificil de implementat pentru grafuri cu un număr mare de noduri - se poate ajunge la operații cu numere mari;

III) Calculăm lungimea minimă a unui lanț de la  $X$  la  $Y$ . Eliminăm apoi succesiv câte un nod din arbore (cu excepția nodurilor  $X$  și  $Y$ ) și recalculăm lungimea minimă a unui lanț de la  $X$  la  $Y$ . Dacă această lungime diferă față de cea inițială, rezultă că toate lanțurile de lungime minimă trec prin nodul eliminat. Soluție de complexitate  $O(n(n+m))$  care rezolvă corect 50% din teste.

IV) Se pot da și soluții bazate pe backtracking, de exemplu:

- se determină (printr-o parcurgere în lățime a grafului) numărul de vârfuri aflate pe lanțul de lungime minimă dintre  $X$  și  $Y$ ; fie  $K$  numărul de vârfuri "intermediare" (fără  $X$  și  $Y$ );

- generăm toate lanțurile de lungime minimă dintre  $X$  și  $Y$  (inclusiv capetele) (algoritm de generare a aranjamentelor de  $n$  luate câte  $K$  cu verificările corespunzătoare de adiacență) și conținem pentru fiecare vârf numărul de lanțuri în care apare;

- vârfurile care formează soluția vor avea conținutul egal cu conținutul vârfurilor  $X$  și  $Y$ .

În funcție de implementare și de optimizările aduse algoritmilor de backtracking, programele bazate pe astfel de soluții pot primi maxim 40 puncte.

### 5.1.2 Rezolvare detaliată

### 5.1.3 Codul sursă \*

Varianta 1:

```
import java.io.*;
class graf // cu vectori de muchii, sortati (heapsort)
{
    static StreamTokenizer st;
    static PrintWriter out;

    static int n,m,x,y,nv; // nv = nr varfuri (solutie)

    static int[] v1; // v1[k] = v1 muchia k
    static int[] v2; // v2[k] = v2 muchia k
    static int[] q; // coada

    static int[] fdx; // fdx[k] = frecventa distantei k fata de x
    static int[] vdm; // vdm[i] = 1 daca i se afla pe un drum minim x...y

    static int[] dx; // dx[i] = dist(x,i)
    static int[] dy; // dy[i] = dist(i,y)
```

```
static int[] ai; // ai[k] = adresa inceput lista adiacenta pentru k
static int[] ne; // ne[k] = nr elemente din lista de adiacenta a lui k

static void afisv(int[] v,int i1, int i2)
{
    int i;
    for(i=i1;i<=i2;i++) System.out.print(v[i]+" ");
    System.out.println();
}

static void citire() throws IOException
{
    int i,j,k;

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

    v1=new int[2*m+1];
    v2=new int[2*m+1];

    q=new int[n+1];

    fdx=new int[n+1];
    vdm=new int[n+1];

    dx=new int[n+1];
    dy=new int[n+1];

    ai=new int[2*m+1];
    ne=new int[n+1];

    for(k=1;k<=m;k++)
    {
        st.nextToken(); i=(int)st.nval;
        st.nextToken(); j=(int)st.nval;

        v1[2*k-1]=i; v2[2*k-1]=j;
        v1[2*k]=j; v2[2*k]=i;
    }
}
```



```
static void calculd(int[] d, int nods) // nodstart
{
    int ic, sc, k, z;

    for(k=1;k<=n;k++) d[k]=-1;

    q[0]=nods;
    d[nods]=0;

    ic=0;
    sc=1;

    while(ic != sc)
    {
        z=q[ic];
        q[ic]=0;
        ic++;

        for(k=ai[z]; k<=ai[z]+ne[z]-1; k++)
        {
            if(d[v2[k]] == -1)
            {
                q[sc]=v2[k];
                sc++;
                d[v2[k]]=1+d[z];
            } // if
        } // for
    } // while
} // calculd()
```

```
static void hs(int[] x, int[] y, int n)
{
    int t,f,fs,fd,aux,k;

    // construiesc heap

    for(k=2;k<=n;k++)
    {
        // urc ...
        f=k;
        t=f/2;
        while(t>0)
        {
```

```

    if(x[t]<x[f])
    {
        aux=x[t]; x[t]=x[f]; x[f]=aux;
        aux=y[t]; y[t]=y[f]; y[f]=aux;
        f=t;
        t=f/2;
    }
    else break;
} //while
}

// sortez: 1) permut varaf<-->ultimul; 2) cobor ...

for(k=n-1;k>=1;k--) // k = dim heap dupa x[1]<-->x[k+1]
{
    // max <--> la coada ...
    aux=x[1]; x[1]=x[k+1]; x[k+1]=aux;
    aux=y[1]; y[1]=y[k+1]; y[k+1]=aux;

    // cobor ... f=max(fs,fd) ... daca exista fd ...
    t=1;
    fs=2; fd=3;

    f=fs;
    if(fd<=k) if(x[fs]<x[fd]) f=fd;

    while(f<=k)
    {
        if(x[t]<x[f])
        {
            aux=x[t]; x[t]=x[f]; x[f]=aux;
            aux=y[t]; y[t]=y[f]; y[f]=aux;

            t=f;

            if(t>0x7ffff/2) break; // fs > MAXINT ...

            fs=2*t; fd=fs+1;
            f=fs;

            if(fs<0x7ffff) // ca sa existe fd ...
            if(fd<=k) if(x[fs]<x[fd]) f=fd;
        }
    }
    else break;
}

```

```
    }// while
  }// for k
}

static void calcaine()
{
  int i,j,k;

  ai[v1[1]]=1;
  ne[v1[1]]=1;

  for(k=2;k<=2*m;k++)
    if(v1[k]==v1[k-1]) ne[v1[k]]++;
    else
    {
      ai[v1[k]]=k;
      ne[v1[k]]=1;
    }
}

static void solutia()
{
  int i;

  fdx=new int[n]; // distante = 0,1,...,n-1 !!!
  vdm=new int[n+1]; // varf pe drum minim x...y

  for(i=0;i<=n-1;i++) fdx[i]=0;
  for(i=1;i<=n;i++) vdm[i]=0;

  for(i=1;i<=n;i++)
    if(dx[i]+dy[i] == dx[y])
    {
      fdx[dx[i]]++;
      vdm[i]=1;
    }

  //afisv(fdx,0,n-1);
  //afisv(vdm,1,n);

  nv=0;
  for(i=0;i<=n-1;i++) if(fdx[i]==1) nv++;

  out.println(nv);
}
```

```
for(i=1;i<=n;i++)
    if((vdm[i]==1)&&(fdx[dx[i]]==1))
    {
        out.print(i+" ");
    }

out.println();
}

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

    citire();

    //afisv(v1,1,2*m);
    //afisv(v2,1,2*m);

    hs(v1,v2,2*m);

    //afisv(v1,1,2*m);
    //afisv(v2,1,2*m);

    calcaine();

    //afisv(ai,1,n);
    //afisv(ne,1,n);

    q=new int[n];

    calculd(dx,x);
    calculd(dy,y);

    //afisv(dx,1,n);
    //afisv(dy,1,n);

    solutia();

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

Varianta 2:

```

import java.io.*;
class graf // lista de adiacenta - o singura citire din fisier !!!
{
    static StreamTokenizer st;
    static PrintWriter out;

    static final int nmax=7500;
    static final int mmax=14000;

    static int n,m,x,y,nv; // nv = nr varfuri (solutie)

    static int[] v1; // v1[k] = v1 muchia k
    static int[] v2; // v2[k] = v2 muchia k
    static int[] q; // coada

    static int[] fdx; // fdx[k] = frecventa distantei k fata de x
    static int[] vdm; // vdm[i] = 1 daca i se afla pe un drum minim x...y

    static int[] dx=new int[nmax+1]; // dx[i] = dist(x,i)
    static int[] dy=new int[nmax+1]; // dy[i] = dist(i,y)

    static int[] gi; // gi[k] = grad interior nod k
    static int[] ge; // ge[k] = grad exterior nod k

    static int[][] a; // liste de adiacenta ...

    static void afisv(int[] v,int i1, int i2)
    {
        int i;
        for(i=i1;i<=i2;i++) System.out.print(v[i]+" ");
        System.out.println();
    }

    static void citire() throws IOException
    {
        int i,j,k;

        st.nextToken(); n=(int)st.nval;
        st.nextToken(); m=(int)st.nval;
        st.nextToken(); x=(int)st.nval;
        st.nextToken(); y=(int)st.nval;
    }
}

```

```
v1=new int[2*m+1];
v2=new int[2*m+1];

gi=new int[n+1];
ge=new int[n+1];

for(i=1;i<=n;i++) gi[i]=ge[i]=0;

for(k=1;k<=m;k++)
{
    st.nextToken(); i=(int)st.nval;
    st.nextToken(); j=(int)st.nval;
    v1[k]=i; v2[k]=j;
    gi[j]++; ge[i]++;
}

a=new int[n+1][1]; // apoi modific 1 ...

for(i=1;i<=n;i++)
{
    a[i]=new int[gi[i]+ge[i]+1]; // am modificat ...
    a[i][0]=0;
}

for(k=1;k<=m;k++)
{
    i=v1[k]; j=v2[k];
    a[i][0]++; a[j][0]++;

    a[i][a[i][0]]=j;
    a[j][a[j][0]]=i;
}
} // citire(...)

static void calculd(int[] d, int nods) // nodstart
{
    int ic, sc, k, z;

    for(k=1;k<=n;k++) d[k]=-1;

    q[0]=nods;
    d[nods]=0;

    ic=0;
```

```

sc=1;

while(ic != sc)
{
    z=q[ic];
    q[ic]=0;
    ic++;

    for(k=1; k<=a[z][0]; k++)
    {
        if(d[a[z][k]] == -1)
        {
            q[sc]=a[z][k];
            sc++;
            d[a[z][k]]=1+d[z];
        }
    }
}

static void solutia()
{
    int i;

    fdx=new int[n];          // distante = 0,1,...,n-1 !!!
    vdm=new int[n+1];        // varf pe drum minim x...y

    for(i=0;i<=n-1;i++) fdx[i]=0;
    for(i=1;i<=n;i++) vdm[i]=0;

    for(i=1;i<=n;i++)
        if(dx[i]+dy[i] == dx[y])
        {
            //System.out.println("i = "+i+" dx[i] = "+dx[i]);
            fdx[dx[i]]++;
            vdm[i]=1;
        }

    //afisv(fdx,0,n-1);
    //afisv(vdm,1,n);

    nv=0;
    for(i=0;i<=n-1;i++) if(fdx[i]==1) nv++;
}

```

```

    out.println(nv);

    for(i=1;i<=n;i++)
        if((vdm[i]==1)&&(fdx[dx[i]]==1))
            {
                out.print(i+" ");
            }

    out.println();
}

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

    citire();

    //afisv(v1,1,2*m);
    //afisv(v2,1,2*m);

    q=new int[n];

    calculd(dx,x);
    calculd(dy,y);

    //afisv(dx,1,n);
    //afisv(dy,1,n);

    solutia();

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

```

Varianta 3:

```

import java.io.*;
class graf // lista de adiacenta - doua citiri din fisier !!! !!!
{
    static StreamTokenizer st;
    static PrintWriter out;

```



```

static final int nmax=7500;
static final int mmax=14000;

static int n,m,x,y,nv;    // nv = nr varfuri (solutie)

static int[] q;          // coada

static int[] fdx;        // fdx[k] = frecventa distantei k fata de x
static int[] vdm;        // vdm[i] = 1 daca i se afla pe un drum minim x...y

static int[] dx=new int[nmax+1];    // dx[i] = dist(x,i)
static int[] dy=new int[nmax+1];    // dy[i] = dist(i,y)

static int[] gi;          // gi[k] = grad interior nod k
static int[] ge;          // ge[k] = grad exterior nod k

static int[][] a;        // liste de adiacenta ...

static void afisv(int[] v,int i1, int i2)
{
    int i;
    for(i=i1;i<=i2;i++) System.out.print(v[i]+" ");
    System.out.println();
}

static void citire() throws IOException
{
    int i,j,k;

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

    gi=new int[n+1];
    ge=new int[n+1];

    for(i=1;i<=n;i++) gi[i]=ge[i]=0;

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

```

```
}

a=new int[n+1][1];          // apoi modific 1 ...

for(i=1;i<=n;i++)
{
    a[i]=new int[gi[i]+ge[i]+1];
    a[i][0]=0;
}

// "reset" pentru a doua citire din fisier ... !!!
st=new StreamTokenizer(new BufferedReader(new FileReader("graf.in")));

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

for(k=1;k<=m;k++)
{
    st.nextToken(); i=(int)st.nval;
    st.nextToken(); j=(int)st.nval;
    a[i][0]++; a[j][0]++;

    a[i][a[i][0]]=j;
    a[j][a[j][0]]=i;
}
} // citire(...)

static void calculd(int[] d, int nods)    // nodstart
{
    int ic, sc, k, z;

    for(k=1;k<=n;k++) d[k]=-1;

    q[0]=nods;
    d[nods]=0;

    ic=0;
    sc=1;

    while(ic != sc)
    {
        z=q[ic];
```

```

    q[ic]=0;
    ic++;

    for(k=1; k<=a[z][0]; k++)
    {
        if(d[a[z][k]] == -1)
        {
            q[sc]=a[z][k];
            sc++;
            d[a[z][k]]=1+d[z];
        }// if
    }// for
} // while
} // calculd()

static void solutia()
{
    int i;

    fdx=new int[n];          // distante = 0,1,...,n-1 !!!
    vdm=new int[n+1];        // varf pe drum minim x...y

    for(i=0;i<=n-1;i++) fdx[i]=0;
    for(i=1;i<=n;i++) vdm[i]=0;

    for(i=1;i<=n;i++)
        if(dx[i]+dy[i] == dx[y])
        {
            //System.out.println("i = "+i+" dx[i] = "+dx[i]);
            fdx[dx[i]]++;
            vdm[i]=1;
        }

    //afisv(fdx,0,n-1);
    //afisv(vdm,1,n);

    nv=0;
    for(i=0;i<=n-1;i++) if(fdx[i]==1) nv++;

    out.println(nv);

    for(i=1;i<=n;i++)
        if((vdm[i]==1)&&(fdx[dx[i]]==1))
        {

```

```
        out.print(i+" ");
    }

    out.println();
}

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

    citire();

    //afisv(v1,1,2*m);
    //afisv(v2,1,2*m);

    q=new int[n];

    calculd(dx,x);
    calculd(dy,y);

    //afisv(dx,1,n);
    //afisv(dy,1,n);

    solutia();

    out.close();
}
}
```

## 5.2 Cifru - OJI 2006

*Stelian Ciurea*

Un criptolog amator își propune să construiască o mașină de cifrat care să cripteze un text alcătuit din exact  $N$  simboluri distincte. Cifrarea se realizează prin permutarea simbolurilor ce formează textul.

Criptologul nostru dorește ca reconstituirea textului inițial să poată fi realizată trecând textul cifrat încă de  $K - 1$  ori prin procedura de cifrare. Cu alte cuvinte, dacă textul rezultat din prima cifrare este cifrat încă o dată, rezultatul este cifrat din nou și așa mai departe, plecând de la textul inițial și aplicând în total  $K$  operații de cifrare successive, trebuie să obțină textul inițial.

Criptologul nostru ar vrea să afle, cunoscând  $N$  și  $K$ , numărul de moduri distincte în care poate fi realizată mașina de cifrat. Două moduri de realizare a

mașinii diferă dacă, există cel puțin un text în urma cifrării căruia, în cele două texte obținute există cel puțin o poziție în care se află simboluri diferite.

### Cerință

Scrieți un program care determină restul împărțirii numărului de moduri distincte în care poate fi realizată mașina de cifrat la 1997.

### Date de intrare

Fișierul **cifru.in** conține pe prima (și singura) linie, două valori numerice naturale separate printr-un spațiu,  $N$  și  $K$  (cu semnificația din enunț).

### Date de ieșire

Fișierul **cifru.out** va conține pe prima linie, numărul de moduri distincte de realizare a mașinii de cifrat modulo 1997.

### Restricții

- $1 \leq N \leq 2000$ ;  $2 \leq K \leq 1000000000$
- pentru 30% din teste  $N, K < 13$
- pentru 50% din teste  $N, K \leq 100$

### Exemple

cifru.in	cifru.out	cifru.in	cifru.out	cifru.in	cifru.out
3 3	3	9 6	11560	100 200	13767

**Tim maxim de execuție/test:** 2 secunde.

## 5.2.1 Indicații de rezolvare - descriere soluție \*

### Soluția oficială

1) Problema se poate reformula astfel: fie mulțimea  $\{1, 2, \dots, n\}$  și fie  $\sigma$  o permutare a acestei mulțimi care îndeplinește condiția:

$$\underbrace{\sigma(\sigma(\dots\sigma(x)\dots))}_{k \text{ ori}} = x \quad \text{pentru orice } x \in \{1, 2, \dots, n\}$$

- Pentru început, vom da soluția pentru un caz particular al problemei și anume pentru  $K = 3$ : să notăm cu  $f(n)$  numărul acestor permutări, și calculăm în câte moduri 1 (primul element al mulțimii) poate să revină pe prima poziție după  $K$  permutări. Există două posibilități și anume:

1)  $\sigma(1) = 1$  și atunci evident că și după  $K$  permutări successive 1 va rămâne pe prima poziție; în acest caz, pentru celelalte  $n - 1$  elemente ale mulțimii vor exista  $f(n - 1)$  permutări care îndeplinesc condiția cerută;

2) elementul 1 formează un ciclu de lungime trei (evident cu încă alte două elemente din mulțime, fie acestea  $a$  și  $b$ . Astfel  $\sigma(1) = a, \sigma(a) = b, \sigma(b) = 1$  și atunci aceste trei elemente revin pe pozițiile lor după trei permutări successive. În acest caz, cele două elemente  $a$  și  $b$  putem să le alegem în  $(n-1)(n-2) = A_n^2$  moduri, iar pentru restul de  $n-3$  elemente ale mulțimii vom avea  $f(n-3)$  permutări care corespund cerinței din enunț.

Având în vedere aceste două posibilități putem scrie următoarea relație de recurență:

$$f(n) = f(n-1) + (n-1)(n-2)f(n-3) \quad \text{și} \quad f(n) = 1 \quad \text{pentru} \quad n \leq 1$$

- Dacă  $K$  este număr prim, un raționament asemănător ne conduce la următoarea relație de recurență:

$$f(n) = f(n-1) + (n-1)(n-2)(n-k+1)f(n-k) \quad \text{și} \quad f(n) = 1 \quad \text{pentru} \quad n \leq 1$$

- Dacă  $K$  nu este număr prim, atunci raționamentul anterior trebuie efectuat pentru fiecare divizor a lui  $K$ .

De exemplu, dacă  $K = 6$ , atunci elementul 1 poate să revină pe prima poziție după  $K$  permutări succesive dacă

-  $\sigma(1) = 1$ ;

- elementul 1 formează un ciclu de lungime 2 (atunci revine pe prima poziție după două permutări, după patru permutări și în final după șase permutări); celălalt element al acestui ciclu poate fi ales în  $A_n^1$  iar pentru mulțimea formată din celelalte  $n-2$  elemente avem  $f(n-2)$  permutări;

- elementul 1 formează un ciclu de lungime 3 (atunci revine pe prima poziție după trei permutări și după șase permutări);

- elementul 1 formează un ciclu de lungime 6 (atunci revine pe prima poziție după șase permutări);

Obținem astfel pentru cazul  $n = 6$ , relația:

$$f(n) = f(n-1) + (n-1)f(n-2) + (n-1)(n-2)f(n-3) + (n-1)(n-2)\dots(n-5)f(n-6)$$

sau

$$f(n) = f(n-1) + A_n^1 f(n-2) + A_n^2 f(n-3) + A_n^5 f(n-6)$$

și

$$f(n) = 1 \quad \text{pentru} \quad n \leq 1$$

Generalizând, putem scrie recurența

$$f(n) = f(n-1) + \sum_{d=2}^k A_{n-1}^{d-1} f(n-d)$$

pentru toții divizorii  $d$  ai lui  $K$  care sunt mai mici sau egali cu  $n$ .

Programul constă în implementarea acestei formule. Pentru a obține punctaj maxim, trebuie făcute câteva optimizări cum ar fi:

- calculul  $A_{n-1}^{d2}$  unde  $d2$  este un divizor a lui  $K$  trebuie făcut plecând de la valoarea  $A_{n-1}^{d1}$  calculată în prealabil, unde  $d1 < d2$  și  $d1$  divizor al lui  $K$ ;
- reținerea valorilor calculate pentru funcția  $f$  într-un vector, pentru a evita calcularea în mod repetat a acestora.

Fără aceste optimizări, un program care implementează soluția descrisă primește 50 puncte.

*II)* O soluție bazată pe backtracking, care generează toate permutările și verifică pentru fiecare dacă îndeplinește condiția din enunț este corectă, dar se încadrează în timpul maxim de rulare/test doar pentru valori mici ale lui  $n$  și  $k$ . Pentru datele de test propuse, o astfel de soluție primește 20 puncte;

*III)* Se pot lua 30 puncte cu o astfel de soluție dacă se rulează programul în timpul concursului pentru toate combinațiile posibile sugerate de observația că

- pentru 30% din teste  $N, K < 13$

și se rețin rezultatele într-o matrice de constante.

## 5.2.2 Rezolvare detaliată

### 5.2.3 Codul sursă \*

Varianta 1:

```
import java.io.*;
class cifru
{
    static StreamTokenizer st;
    static PrintWriter out;

    static int[] x=new int[2001];
    static int n, k;

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

        citire();
        solutia();
    }
}
```

```
        out.println(x[n]);
        out.close();
    }

    static void citire() throws IOException
    {
        st.nextToken(); n=(int)st.nval;
        st.nextToken(); k=(int)st.nval;
    }

    static void solutia()
    {
        int i,j,a;

        x[0]=1;
        for(i=1;i<=n;i++)
        {
            j=1;
            a=1;
            while((j<=i)&&(j<=k))
            {
                if(k%j==0) x[i]=(x[i]+a*x[i-j])%19997;
                a=(a*(i-j))%19997;
                j++;
            } // while
        } //for
    } // solutia
} // class
```

Varianta 2:

```
import java.io.*;
class cifru
{
    static StreamTokenizer st;
    static PrintWriter out;

    static int[] x=new int[5005];
    static int n, k, rezsol, prim=19997;

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



```

    citire();
    solutia();

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

static void citire() throws IOException
{
    st.nextToken(); n=(int)st.nval;
    st.nextToken(); k=(int)st.nval;
}

static int f(int n)
{
    int d,i,rez=0,t;

    if (n<k)
    {
        if(n<=1) return 1;
        if(x[n]!=0) return x[n];

        int da=1, p=1;
        for(d=2;d<=n;d++)
            if(k%d==0)
            {
                for(i=n-da;i>=n-d+1;i--) p=(p*i)%prim;
                t=p*f(n-d);
                rez=(rez+t)%prim;
                da = d;
            }
        return x[n]=(f(n-1)+rez)%prim;
    }// if

    if(x[n]!=0) return x[n];

    int da=1, p=1;
    for(d=2;d<=k;d++)
        if (k%d==0)
        {
            for(i=n-da;i>=n-d+1;i--) p=(p*i)%prim;
            t=(p*f(n-d))%prim;
            rez=(rez+t)%prim;
        }
}

```

```
        da=d;
    }
    return x[n]=(f(n-1)+rez)%prim;
} // f(...)

static void solutia()
{
    rezsol=f(n);
} // solutia
} // class
```

# Capitolul 6

## OJI 2007 clasa a XI-a

### 6.1 Numere - OJI 2007

Fie  $a$  și  $b$  două numere naturale nenule.

#### Cerință

Scrieți un program care citește din fișierul de intrare două valori  $a$  și  $b$ , determină numărul de numere naturale formate din exact  $a$  cifre care au fiecare produsul cifrelor egal cu  $b$  și afișează în fișierul de ieșire restul împărțirii valorii determinate la numărul 9973.

#### Date de intrare

Fișierul de intrare **numere.in** conține pe prima linie numerele  $a$  și  $b$  despărțite printr-un spațiu.

#### Date de ieșire

Fișierul de ieșire **numere.out** va conține pe prima linie o singură valoare care reprezintă restul împărțirii numărului de numere naturale formate din exact  $a$  cifre care au produsul cifrelor egal cu  $b$  la 9973.

#### Restricții și precizări

- Pentru 10% din teste  $1 \leq a \leq 6, 1 \leq b \leq 1000$
- Pentru 20% din teste  $7 \leq a \leq 150, 1 \leq b \leq 100$
- Pentru 30% din teste  $151 \leq a \leq 1000, 1 \leq b \leq 100$
- Pentru 40% din teste  $1001 \leq a \leq 9000, 100 \leq b \leq 9000$

#### Exemple:

numere.în	numere.out	Explicații - numerele sunt:
3 9	6	119 133 191 313 331 911
4 15	12	1135 1153 1315 1351 1513 1531 3115 3151 3511 5113 5131 5311
1000 210	833	...

**Timp maxim de execuție/test:** 1 secundă

### 6.1.1 Indicații de rezolvare - descriere soluție \*

**Soluția 1 -Programare dinamică. (Ilie Vieru) -100 puncte**

- Dacă  $a$  are divizori primi mai mari decât 9 atunci scrie 0

- Altfel:

$$nr[1][d] = \begin{cases} 0, & \text{dacă } b \% d \neq 0 \\ 1, & \text{în rest} \end{cases}$$

$$nr[i][j] = \sum_{\substack{1 \leq p \leq 9 \\ p|b, p|j}} nr[i-1][j/p], \text{ unde } 2 \leq i \leq a, 1 \leq j \leq b$$

Se afișează  $nr[a][b](mod)$ .

Problema se rafinează ajungând la complexitatea  $O(a * nrdiv(b))$  înlocuind calculul matriceal cu cel cu vectori.

**Soluția 2 - backtracking (Stelian Ciurea) -100 puncte**

Problema se poate rezolva și prin metoda backtracking!

Pentru aceasta:

- se face descompunerea numărului  $b$  în factori primi (se observă că puterea maximă la care poate să apară un factor prim în această descompunere este 13 în cazul factorului prim 2);

- în funcție de această descompunere se reține pentru fiecare cifră care este numărul maxim de câte ori poate să apară în numărul  $a$  (în vectorul  $f$ );

- se determină prin backtracking, succesiv, câte o configurație posibilă a numărului  $a$  astfel încât produsul cifrelor lui să fie  $b$ : astfel  $s[2]$  va reține câte cifre de 2 apar în  $a$ ,  $s[3]$  va reține câte cifre de 3 apar în  $a$  etc;

- se determină numărul de cifre de 1 care apare într-o astfel de configurație;

- pentru o configurație posibilă se calculează toate numerele care au respectiva configurație a cifrelor, folosind formula aranjamentelor cu repetiție:

$$A_b^{s[1], s[2], \dots, s[9]} = \frac{b}{s[1]! s[2]! \dots s[9]!}$$

unde  $s[1]$  este numărul de cifre de 1 care apar în  $a$ ,  $s[2]$  este numărul de cifre de 2 care apar în  $a$  etc.

- plecând de la observația evidentă că numărul de cifre de 1 este mult mai mare în comparație cu numărul celorlalte cifre (mai ales pentru valori mari ale lui

b), se simplifică înainte de a începe calculul propriu-zis în formula anterioară  $b!$  cu  $s[1]!$ .

- pentru a evita calcule repetate, înainte de apelarea subprogramului care implementează backtracking-ul descris anterior, se precalculează în vectorul *fact* factorialele numerelor de la 2 la 13 (13 fiind factorialul maxim care apare la numitorul aranjamentelor cu repetiții după simplificarea descrisă mai sus), precum și puterile cifrelor de la 1 la 13 în matricea *pow*.

### 6.1.2 Rezolvare detaliată

### 6.1.3 Codul sursă \*

Varianta 1:

```
import java.io.*;
class numere
{
    static StreamTokenizer st;
    static PrintWriter out;

    static final int MAXAB=9000;

    static int nc,pc,ndiv=0,rez=-1; // nc=nr cifre, pc=produs cifre

    static int[] nrpos1=new int[MAXAB+1];
    static int[] nrpos2=new int[MAXAB+1];
    static int[] div =new int[MAXAB+1];

    static void citire() throws IOException
    {
        st.nextToken(); nc=(int)st.nval;
        st.nextToken(); pc=(int)st.nval;
    } // citire(...)

    static boolean prim(int nr)
    {
        int d;
        if((nr%2)==0) return false;
        for(d=3;d*d<=nr;d+=2) if((nr%d)==0) return false;
        return true;
    }
}
```

```
static void solutia()
{
    int i,j,p;
    boolean ok=true;

    for(i=11;i<=pc;++i)
        if((pc%i)==0 && prim(i))
        {
            ok=false;
            break;
        }

    for(i=1;i<=pc;++i) if((pc%i)==0) div[++ndiv]=i;

    if(ok)
    {
        for(i=1; i<=ndiv && div[i]<=9 ; ++i) nrpos1[div[i]]=1;

        for(i=2;i<=nc;++i)
        {
            for(j=1;j<=ndiv;++j) nrpos2[div[j]]=0;

            for(j=1;j<=ndiv;++j)
            {
                for(p=1;p<=ndiv && div[p]<=9;++p)
                    if(div[j]%div[p]==0)
                        nrpos2[div[j]]=(nrpos2[div[j]]+nrpos1[div[j]/div[p]])%9973;
            }

            for(j=1;j<=ndiv;++j) nrpos1[div[j]]=nrpos2[div[j]]%9973;
        }

        rez=nrpos1[pc];
    }
    else rez=0;
}

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

    citire();
    solutia();
}
```

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

Varianta 2:

```
import java.io.*;
class numere
{
    static StreamTokenizer st;
    static PrintWriter out;

    static int nc,pc; // nc=nr cifre, pc=produs cifre

    static int[] s=new int[10];
    static int[] f=new int[10];
    static int[] fact=new int[21];
    static int[][] pow=new int[10][14];
    static int k2,k3,k5,k7;

    static int j,b1,a,b,i;
    static int x,prod1,rez1,rez,prod;

    static void citire() throws IOException
    {
        st.nextToken(); a=(int)st.nval;
        st.nextToken(); b=(int)st.nval;
    }// citire(...)

    static int cmmdc(int a, int b)
    {
        while(a!=b) if(a>b) a-=b; else b-=a;
        return a;
    }

    static void print(int p)
    {
        int g13,j1,i,j,k;

        k=0; g13=0; prod1=1;
```

```
for(j=2;j<=p;j++)
{
    k=k+s[j];
    if(s[j]==13) g13=1;
    prod1=prod1*fact[s[j]];
}

s[1]=a-k;
rez1=1;
for(j=s[1]+1;j<=a;j++)
{
    j1=j;
    if(prod1!=1)
    {
        x=cmmdc(j,prod1);
        j1=j1/x;
        prod1=prod1/x;
        if(g13==1)
            if(j1%13==0) { g13=0; j1=j1/13; }
    }
    rez1=(rez1*j1)%9973;
}

rez=(rez+rez1)%9973;
}

static void back(int p)
{
    int i;
    for(i=0;i<=f[p];i++)
    {
        s[p]=i;
        prod=prod*pow[p][i];
        if(prod==b1) print(p);
        if(prod<b1) if(p<9) back(p+1);
        prod=prod/pow[p][i];
    }
}

static void solutia()
{
```



```

b1=b;
fact[0]=1;
for(i=1;i<=12;i++) fact[i]=fact[i-1]*i ;
fact[13]=fact[12];

for(i=2;i<=9;i++) { pow[i][1]=i; pow[i][0]=1;}
for(i=2;i<=9;i++)
    for(j=2;j<=13;j++)
        if(pow[i][j-1]*i<10000) pow[i][j]=pow[i][j-1]*i;

while(b%2==0) { b=b/2; k2++; }
while(b%3==0) { b=b/3; k3++; }
while(b%5==0) { b=b/5; k5++; }
while(b%7==0) { b=b/7; k7++; }

f[2]=k2; f[3]=k3; f[4]=k2/2; f[5]=k5;
f[7]=k7; f[8]=k2/3; f[9]=k3/2;
if(k2<k3) f[6]=k2; else f[6]=k3;

prod=1;
back(2);
}

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

    citire();
    solutia();

    System.out.println(rez);
    out.println(rez);
    out.close();
} // main
} // class

```

## 6.2 Cezar - OJI 2007

În Roma antică există  $n$  așezări senatoriale distincte, câte una pentru fiecare dintre cei  $n$  senatori ai Republicii. Așezările senatoriale sunt numerotate de la 1 la  $n$ , între oricare două așezări existând legături directe sau indirecte. O legătură

este directă dacă ea nu mai trece prin alte așezări senatoriale intermediare. Edilii au pavat unele dintre legăturile directe dintre două așezări (numind o astfel de legătură pavată "stradă"), astfel încât între oricare două așezări senatoriale să existe o singură succesiune de străzi prin care se poate ajunge de la o așezare senatorială la cealaltă.

Toți senatorii trebuie să participe la ședințele Senatului. În acest scop, ei se deplasează cu lectica. Orice senator care se deplasează pe o stradă plătește 1 ban pentru că a fost transportat cu lectica pe acea stradă.

La alegerea sa ca prim consul, Cezar a promis că va dota Roma cu o lectică gratuită care să circule pe un număr de  $k$  străzi ale Romei astfel încât orice senator care va circula pe străzile respective, să poată folosi lectica gratuită fără a plăti. Străzile pe care se deplasează lectica gratuită trebuie să fie legate între ele (zborul, metroul sau teleportarea nefiind posibile la acea vreme).

În plus, Cezar a promis să stabilească sediul sălii de ședințe a Senatului într-una dintre așezările senatoriale aflate pe traseul lecticii gratuite. Problema este de a alege cele  $k$  străzi și amplasarea sediului sălii de ședințe a Senatului astfel încât, prin folosirea transportului gratuit, senatorii, în drumul lor spre sala de ședințe, să facă economii cât mai însemnate. În calculul costului total de transport, pentru toți senatorii, Cezar a considerat că fiecare senator va călători exact o dată de la așezarea sa până la sala de ședințe a Senatului.

### Cerință

Scrieți un program care determină costul minim care se poate obține prin alegerea adecvată a celor  $k$  străzi pe care va circula lectica gratuită și a locului de amplasare a sălii de ședințe a Senatului.

### Date de intrare

Fișierul **cezar.in** conține

- pe prima linie două valori  $n$   $k$  separate printr-un spațiu reprezentând numărul total de senatori și numărul de străzi pe care circulă lectica gratuită
- pe următoarele  $n - 1$  linii se află câte două valori  $i$   $j$  separate printr-un spațiu, reprezentând numerele de ordine a două așezări senatoriale între care există stradă.

### Date de ieșire

Pe prima linie a fișierului **cezar.out** se va scrie costul total minim al transportării tuturor senatorilor pentru o alegere optimă a celor  $k$  străzi pe care va circula lectica gratuită și a locului unde va fi amplasată sala de ședințe a Senatului.

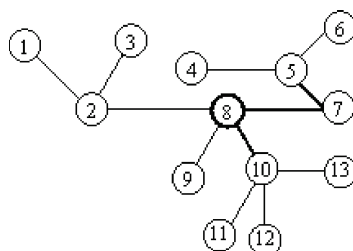
### Restricții și precizări

- $1 < n \leq 10000$ ,  $0 < k < n$
- $1 \leq i, j \leq n$ ,  $i \neq j$
- Oricare două perechi de valori de pe liniile 2, 3, ...,  $n$  din fișierul de intrare reprezintă două străzi distincte.

• Perechile din fișierul de intrare sunt date astfel încât respectă condițiile din problemă.

• Pentru 25% din teste  $n \leq 30$ , pentru 25% din teste  $30 < n \leq 1000$ , pentru 25% din teste  $1000 < n \leq 3000$ , pentru 10% din teste  $3000 < n \leq 5000$ , pentru 10% din teste  $5000 < n \leq 10000$ .

### Exemplu



cezar.in	cezar.out	Explicație
13 3	11	Costul minim se obține, de exemplu, pentru alegerea celor 3 străzi între așezările 5-7, 7-8, 8-10 și a sălii de ședințe a Senatului în așezarea 8 (după cum este evidențiat în desen).  Există și alte alegeri pentru care se obține soluția 11.
1 2		
2 3		
2 8		
7 8		
7 5		
5 4		
5 6		
8 9		
8 10		
10 11		
10 12		
10 13		

**Timp maxim de execuție/test:** 0.5 secunde

#### 6.2.1 Indicații de rezolvare - descriere soluție \*

O implementare posibilă utilizează metoda GREEDY,  $O(n * (n - k))$  sau  $O((n - k) * \log(n))$ .

Se elimină succesiv, dintre frunzele existente la un moment dat, frunza de cost minim. Toate nodurile au costul inițial 1. La eliminarea unei frunze, se incrementează cu 1 costul tatălui acesteia. Validitatea metodei rezultă din observația că, la eliminarea unei frunze oarecare, tatăl acesteia poate deveni frunză la rândul lui, dar cu un cost strict mai mare decât al frunzei eliminate.

Se poate reține arborele cu ajutorul listelor de adiacență (liniare sau organizate ca arbori de căutare), iar frunzele se pot memora într-un minheap de costuri, structură care se actualizează în timp logaritm.

### 6.2.2 Rezolvare detaliată

### 6.2.3 Codul sursă \*

Varianta 1a: Cu test incorect, dar care obține 17 rezultate corecte din 20.

```
import java.io.*;      // teste: 7(1070,1072), 17(17825,17853) si 20(17154,17191) ... ???
class cezar           // eliminat <==> g[i]=0 ... !!!
{
    static StreamTokenizer st;
    static PrintWriter out;

    static int n, ns;

    static int[] v1 = new int[10001];
    static int[] v2 = new int[10001];
    static int[] g = new int[10001];           // gradele
    static int[] ca = new int[10001];        // ca[k]=costul acumulat in nodul k !!!
    static int[] nde = new int[10001];      // nde[k]=nr "descendenti" eliminati pana in k

    static void afisv(int[] v,int i1, int i2)
    {
        int i;
        for(i=i1;i<=i2;i++)
        {
            System.out.print(v[i]+" ");
            if(i%50==0) System.out.println();
        }
        System.out.println();
    }

    static void citire() throws IOException
    {
        int i,j,k;

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

```

for(i=1;i<=n;i++) // curatenie ...
{
    g[i]=0;
    ca[i]=0;
    nde[i]=0;
}

for(k=1;k<=n-1;k++)
{
    st.nextToken(); i=(int)st.nval;
    st.nextToken(); j=(int)st.nval;

    v1[k]=i;
    v2[k]=j;
    g[i]++; g[j]++;
}

//afisv(v1,1,n-1);
//afisv(v2,1,n-1);
//afisv(g,1,n);
} // citire(...)

static int tata(int i) // mai bine cu lista de adiacenta ...
{
    int k,t;
    t=-1; // initializarea aiurea ...
    for(k=1;k<=n-1;k++) // n-1 muchii
    {
        if( (v1[k]==i) && (g[v2[k]] > 0)) {t=v2[k]; break;}
        else
            if( (v2[k]==i) && (g[v1[k]] > 0)) {t=v1[k]; break;}
    }
    return t;
}

static void eliminm() // frunze(g=1) cu cost=min
{
    int i, imin, timin;
    int min;

    min=Integer.MAX_VALUE;
    imin=-1; // initializare aiurea ...
    for(i=1;i<=n;i++) // mai bine cu heapMIN ...
        if(g[i]==1) // i=frunza

```

```
    if(ca[i]+nde[i]<min) // aici este testul INCORECT ...
    {
        min=ca[i]+nde[i];
        imin=i;
    }

    timin=tata(imin);

    g[imin]--; g[timin]--;

    ca[timin]=ca[timin]+ca[imin]+nde[imin]+1;
    nde[timin]=nde[timin]+nde[imin]+1;

    //System.out.println(" Elimin nodul "+imin+" timin = "+timin+
    // " ca = "+ca[timin]+" nde = "+nde[timin]);
} // elimin()

public static void main(String[] args) throws IOException
{
    int k,senat=0,cost=0;
    st=new StreamTokenizer(new BufferedReader(new FileReader("cezar20.in")));
    out=new PrintWriter(new BufferedWriter(new FileWriter("cezar.out")));

    citire();

    for(k=1;k<=n-1-ns;k++)
    {
        //System.out.print(k+" : ");
        eliminm();
    }

    //afisv(c,1,n);

    for(k=1;k<=n;k++)
        if(g[k]>0)
        {
            cost+=ca[k];
            senat=k;
        }

    System.out.println("\n"+cost+" "+senat);

    out.println(cost+" "+senat);
    out.close();
}
```

```

    }// main
}// class

```

Varianta 1b: Cu testul corect!

```

import java.io.*; // OK: "f" cu cat incarca "in plus" = nde[i]+1 (fara ca[i] !!!)
class cezar      // eliminat <==> g[i]=0 ... !!!
{
    static StreamTokenizer st;
    static PrintWriter out;

    static int n, ns;

    static int[] v1 = new int[10001];
    static int[] v2 = new int[10001];
    static int[] g = new int[10001];    // gradele
    static int[] ca = new int[10001];   // ca[k]=costul acumulat in nodul k !!!
    static int[] nde = new int[10001];  // nde[k]=nr "descendenti" eliminati pana in k

    static void citire() throws IOException
    {
        int i,j,k;

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

        for(i=1;i<=n;i++) g[i]=ca[i]=nde[i]=0; // curatenie ...

        for(k=1;k<=n-1;k++)
        {
            st.nextToken(); i=(int)st.nval;
            st.nextToken(); j=(int)st.nval;

            v1[k]=i;
            v2[k]=j;
            g[i]++; g[j]++;
        }
    }
}// citire(...)

static int tata(int i) // mai bine cu lista de adiacenta ...
{
    int k,t;
    t=-1;                // initializarea aiurea ...

```

```

for(k=1;k<=n-1;k++) // n-1 muchii
{
    if( (v1[k]==i) && (g[v2[k]] > 0) ) {t=v2[k]; break;}
    else
        if( (v2[k]==i) && (g[v1[k]] > 0) ) {t=v1[k]; break;}
}
return t;
}

static void eliminm() // frunze(g=1) cu cost=min
{
    int i, imin, timin;
    int min;

    min=Integer.MAX_VALUE;
    imin=-1; // initializare aiurea ...
    for(i=1;i<=n;i++) // mai bine cu heapMIN ...
        if(g[i]==1) // i=frunza
            if(nde[i]+1<min) // ... aici este testul CORECT ... !!!
                {
                    min=nde[i]+1;
                    imin=i;
                }

    timin=tata(imin);
    g[imin]--; g[timin]--;
    ca[timin]=ca[timin]+ca[imin]+nde[imin]+1;
    nde[timin]=nde[timin]+nde[imin]+1; // nr descendenti eliminati
} // elimin()

public static void main(String[] args) throws IOException
{
    int k, senat=0, cost=0;
    st=new StreamTokenizer(new BufferedReader(new FileReader("cezar20.in")));
    out=new PrintWriter(new BufferedWriter(new FileWriter("cezar.out")));

    citire();

    for(k=1;k<=n-1-ns;k++) eliminm();

    for(k=1;k<=n;k++)
        if(g[k]>0)
            {
                cost+=ca[k];
            }
}

```



```

        senat=k;
    }

    System.out.println("\n"+cost+" "+senat);
    out.println(cost+" "+senat);
    out.close();
    //afisv(g,1,n);
} // main
} // class

```

Varianta 2:

```

import java.io.*;          // ok toate testele
class cezar                // cost initial = 1 pentru toate nodurile ...
{
    static StreamTokenizer st;
    static PrintWriter out;

    static int n, ns,s=0;

    static int[] v1 = new int[10001];
    static int[] v2 = new int[10001];
    static int[] g = new int[10001];          // gradele
    static int[] ca = new int[10001];        // ca[k]=costul acumulat in nodul k !!!

    static void afisv(int[] v,int i1, int i2)
    {
        int i;
        for(i=i1;i<=i2;i++)
        {
            System.out.print(v[i]+" ");
            if(i%50==0) System.out.println();
        }
        System.out.println();
    }

    static void citire() throws IOException
    {
        int i,j,k;

        st.nextToken(); n=(int)st.nval;
        st.nextToken(); ns=(int)st.nval;
    }
}

```

```

for(i=1;i<=n;i++)          // curatenie ...
{
    g[i]=0;
    ca[i]=1;                // cost initial in nodul i
}

for(k=1;k<=n-1;k++)
{
    st.nextToken(); i=(int)st.nval;
    st.nextToken(); j=(int)st.nval;

    v1[k]=i;
    v2[k]=j;
    g[i]++; g[j]++;
}

//afisv(v1,1,n-1);
//afisv(v2,1,n-1);
//afisv(g,1,n);
} // citire(...)

static int tata(int i)     // mai bine cu liste de adiacenta ...
{
    int k,t;
    t=-1;                  // initializarea aiurea ...
    for(k=1;k<=n-1;k++)    // este mai bine cu lista de adiacenta ?
    {
        if( (v1[k]==i) && (g[v2[k]]>0)) {t=v2[k]; break;}
        else
            if( (v2[k]==i) && (g[v1[k]]>)) {t=v1[k]; break;}
    }
    return t;
}

static void eliminm()      // frunze(g=1) cu cost=min
{
    int i, imin, timin;
    int min;

    min=Integer.MAX_VALUE;
    imin=-1;               // initializare aiurea ...
    for(i=1;i<=n;i++)      // mai bine cu heapMIN ...
        if(g[i]==1)        // i=frunza
            if(ca[i]<min)    // cost acumulat

```

```
        {
            min=ca[i];
            imin=i;
        }

    timin=tata(imin);

    g[imin]--; g[timin]--;

    ca[timin]=ca[timin]+min;
    s+=min;

    //System.out.println(" Elimin nodul "+imin+
    //                    " timin = "+timin+" ca = "+ca[timin]);
} // elimin()

public static void main(String[] args) throws IOException
{
    int k, senat=0;
    st=new StreamTokenizer(new BufferedReader(new FileReader("cezar20.in")));
    out=new PrintWriter(new BufferedWriter(new FileWriter("cezar.out")));

    citire();

    for(k=1;k<=n-1-ns;k++)
    {
        eliminm();
    }

    //afisv(c,1,n);

    for(k=1;k<=n;k++)
        if(g[k]>0)
        {
            senat=k;
            break;
        }

    System.out.println("\n"+s+" "+senat);

    out.println(s+" "+senat);
    out.close();
} // main
} // class
```



## Capitolul 7

# ONI 2000 clasa a XI-a



### 7.1 Arbore - ONI 2000

*Marius Vlad, student, Universitatea Politehnica, București*

Se dă un arbore format din  $n$  noduri în care fiecare nod are asociat un număr natural nenul.

**Cerință:**

Să se selecteze din arborele inițial un subgraf conex pentru care suma valorilor asociate nodurilor este egală cu un număr natural  $k$  dat. Un subgraf este graful din care se elimină noduri împreună cu muchiile aferente.

**Date de intrare:**

Nodurile arborelui sunt numerotate de la 1 la  $n$ , rădăcina fiind nodul numerotat cu 1.

Fișierul de intrare ARBORE.IN, are următoarea structură:

pe prima linie sunt scrise numerele  $n$  și  $k$ ;

pe cea de-a doua linie este scrisă valoarea asociată nodului 1 (rădăcina arborelui);

pe cea de-a treia linie sunt scrise două numere, primul reprezentând nodul părinte al nodului 2, iar al doilea reprezentând valoarea asociată nodului 2;

pe cea de-a patra linie sunt scrise două numere, primul reprezentând nodul părinte al nodului 3, iar al doilea reprezentând valoarea asociată nodului 3;

...

pe linia  $n + 1$  sunt scrise două numere, primul reprezentând nodul părinte al nodului  $n$ , iar al doilea reprezentând valoarea asociată nodului  $n$ .

**Date de ieșire:**

Ieșirea se va face în fișierul ARBORE.OUT:

dacă nu există soluție se va afișa doar numărul  $-1$  pe prima linie;

dacă există soluție se vor afișa nodurile ce formează un subgraf conex care respectă cerința problemei (câte un singur nod pe fiecare linie).

**Restricții:**

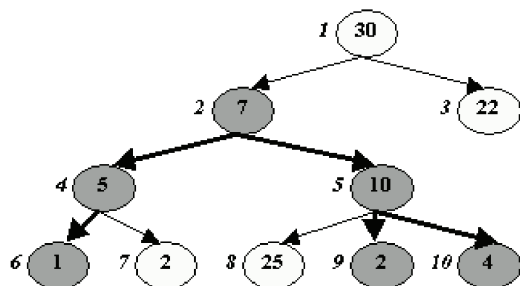
$1 \leq n \leq 100$

$1 \leq k \leq 1000$

$1 \leq \text{valoarea asociată unui nod} \leq 1000$

**Exemplu:**

ARBORE.IN	ARBORE.OUT poate conține:
10 29	2
30	4
1 7	5
1 22	6
2 5	9
2 10	10
4 1	
4 2	
5 25	
5 2	
5 4	



Tim maxim de execuție pe test: 3 secunde

### 7.1.1 Indicații de rezolvare - descriere soluție

### 7.1.2 Rezolvare detaliată

### 7.1.3 Codul sursă

## 7.2 Moara - ONI 2000

*prof. Ion Maxim, Inspectoratul Școlar Județean, Suceava*

La poarta unei mori se află  $n$  saci, etichetați cu numere de la 1 la  $n$ , așezați în linie, într-o ordine dată. Sacul cu eticheta  $i$  are greutatea  $g_i$ .

Morarul, cere ucenicului său să reșeze sacii în ordinea crescătoare a etichetelor. Cum nu este spațiu de manevră pentru a schimba sacii între ei, ucenicul ia un scaun și-l așează lângă poziția  $k$  din șir, alege un sac și-l pune pe scaun, ia alt sac și-l mută în locul gol, apoi aduce alt sac în locul eliberat de acesta etc. sau aduce sacul de pe scaun în locul rămas gol. Repetă procedeul până când sacii ajung în ordinea cerută de morar. Scaunul nu se mută. Efortul depus de ucenic pentru mutatul unui sac, indiferent dacă acesta a fost pus sau nu pe scaun, este egal cu produsul dintre greutatea sacului și distanța pe care a fost transportat sacul. Distanța dintre doi saci alăturați este egală cu unitatea.

#### Cerință:

Se cere să se stabilească poziția amplasării scaunului și ordinea schimbării sacilor între ei, astfel încât ucenicul să facă un număr minim de mutări și efortul depus de acesta să fie minim.

#### Date de intrare:

Fișierul de intrare MOARA.IN conține pe prima linie numărul de saci  $n$  și pe liniile următoare, ordinea așezării sacilor, urmată de șirul greutăților sacilor  $g_1, g_2, \dots, g_n$ , unde  $g_i$  este greutatea sacului cu eticheta  $i$ .

#### Date de ieșire:

Ieșirea se va face în fișierul MOARA.OUT, care va conține pe prima linie  $p k e$  - unde  $p$  este poziția scaunului,  $k$  numărul de mutări și  $e$  valoarea efortului depus, și pe fiecare din liniile următoare câte o mutare de forma:

$d s$  - unde  $d$  este poziția destinație și  $s$  este poziția sursă, poziția scaunului se va nota cu 0 (zero!).

#### Restricții:

$$2 \leq n \leq 10000$$

$$1 \leq k \leq n$$

$$1 \leq g_i \leq 255, \text{ pentru } 1 \leq i \leq n$$

#### Exemplu:

MOARA.IN	MOARA.OUT poate conține:
5	3 5 25
2 4 3	0 2
5 1	2 1
3 5 1 2 4	1 5
	5 4
	4 0

**Timp maxim de execuție pe test:** 1 secundă

### 7.2.1 Indicații de rezolvare - descriere soluție

### 7.2.2 Rezolvare detaliată

### 7.2.3 Codul sursă

## 7.3 Puncte - ONI 2000

*prof. Doru Anastasiu Popescu, Liceul "Radu Greceanu", Slatina*

Se dau  $n$  puncte în plan  $P_1, P_2, \dots, P_n$  și  $k$  un număr natural. Se cere să se verifice dacă se pot construi  $k$  segmente cu ambele capete în mulțimea formată din punctele  $P_1, P_2, \dots, P_n$ , astfel încât să nu se formeze nici un triunghi cu vârfurile în aceste puncte. Dacă există mai multe soluții se cere una dintre ele.

**Date de intrare:**

Fișierul de intrare PUNCTE.IN conține pe prima linie numărul  $n$ , iar pe a doua linie numărul  $k$ .

**Date de ieșire:**

Ieșirea se va face în fișierul PUNCTE.OUT ce va conține:

- o linie pe care se află caracterul 0, dacă nu există soluție.
- $k + 1$  linii, pe prima linie se află caracterul 1 (cu semnificația că există soluție), iar pe următoarele  $k$  linii se află câte două numere separate printr-un spațiu (de forma  $i j$  cu semnificația că  $P_i P_j$  reprezintă un segment), în cazul când există soluție.

**Restricții:**

$$1 \leq n \leq 300$$

$$1 \leq k \leq 300$$

**Exemple:**

PUNCTE.IN	PUNCTE.OUT
4	0
5	

PUNCTE.IN	PUNCTE.OUT poate conține:
4	1
2	1 4
	1 2

**Timp maxim de execuție pe test:** 1 secundă



### 7.3.1 Indicații de rezolvare - descriere soluție \*

#### *Soluția oficială*

Pentru a determina numărul maxim de segmente ce se pot construi fără să existe triunghiuri se partiționează mulțimea punctelor în două submulțimi A și B, prima cu  $\lfloor n/2 \rfloor$  elemente iar cea de-a doua cu  $n - \lfloor n/2 \rfloor$  elemente.

Se unesc prin segmente toate punctele din A cu toate punctele din B (ca la grafuri bipartite complete).

Din faptul că nu se unesc puncte din cadrul aceleiași mulțimi A sau B rezultă că nu se formează triunghiuri. Numărul maxim de segmente căutat este  $\lfloor n^2/4 \rfloor$ .

O soluție poate fi formată din orice  $k$  segmente construite în modul anterior.

### 7.3.2 Rezolvare detaliată

### 7.3.3 Codul sursă

## 7.4 SICN - ONI 2000

*prof. Emanuela Cerchez, Liceul de Informatică, Iași*

Serviciul de Informații al Comisiei Naționale (SICN) este constituit din  $n$  agenți oculti ( $1 \leq n \leq 150$ ), cu numere de cod distincte de la 1 la  $n$  și un agent șef codificat 0.

Din motive de securitate, nu oricare doi agenți au contacte (informaționale!) directe, dar prin contactele directe existente, oricare doi agenți își pot comunica informații.

Un serviciu este considerat *forte* dacă și numai dacă nu conține nici un agent prin suprimarea căruia se compromite comunicarea (și ca urmare, să existe agenți care să nu își mai poată transmite informații).

Scrieți un program, care verifică dacă SICN este *forte*. În plus, dacă SICN nu este *forte*, programul să determine:

- verigile slabe ale SICN (agenții prin a căror suprimare se compromite comunicarea);

- toate grupurile de agenți care sunt *forte* în cadrul SICN, maximale cu această proprietate.

#### **Date de intrare:**

Fișierul de intrare se numește SICN.IN și conține pe prima linie  $n$ , numărul de agenți din SICN, (exclusiv șeful), iar pe fiecare din următoarele linii câte o pereche de numere

$x y$

unde  $x, y \in \{0, 1, 2, \dots, n\}$ , sunt separate prin spații și au semnificația "x și y au contact direct".

**Date de ieșire:**

Fișierul de ieșire, numit SICN.OUT, va conține mesajul "SICN este forte" sau:

- pe prima linie, mesajul "SICN nu este forte".
- pe a doua linie, numerele de cod ale agenților care constituie verigile slabe ale serviciului, în ordine crescătoare, separate prin spații;
- pe a treia linie,  $m$  - numărul de grupuri *forte* în cadrul SICN;
- pe fiecare din următoarele  $m$  linii, numerele de cod ale agenților fiecărui grup *forte*, în ordine crescătoare, separate prin spații.

**Exemple:**

SICN.IN	SICN.OUT
2	SICN este forte
0 2	
0 1	
2 1	

SICN.IN	SICN.OUT
3	SICN nu este forte
0 1	0 1
0 3	3
2 1	0 3
	1 2
	0 1

**Timp de execuție:** maxim 1 secundă per test.

#### 7.4.1 Indicații de rezolvare - descriere soluție

#### 7.4.2 Rezolvare detaliată

#### 7.4.3 Codul sursă

### 7.5 Spioni - ONI 2000

*prof. Rodica Pinteș, Liceul "Grigore Moisil", București*

O rețea formată din  $n$  spioni, numerotați de la 1 la  $n$ , aflați în diferite localități, au o convenție de convocare. Ei vor fi convocați telefonic într-o localitate

fixă numită bază. Convenția de transmitere a convocării cuprinde următoarele reguli:

cu excepția unora, fiecare spion știe ce persoane trebuie să-l contacteze telefonic; pentru a considera convocarea validă și a-i da curs, trebuie să primească semnalul de convocare de la toate persoanele de contact;

cei câțiva spioni care nu așteaptă nici o convocare știu că trebuie să pornească toți deodată, la o oră considerată ora 0, spre bază;

fiecare persoană, imediat ce a ajuns la bază, primește lista persoanelor pe care trebuie să le sune, transmițându-le convocarea;

durata unei convorbiri telefonice este neglijabilă;

abia în momentul în care sunt prezente toate cele  $n$  persoane la bază poate să înceapă ședința.

**Cerința:**

Cunoscându-se, pentru fiecare spion, durata deplasării până la bază (exprimată în ore) și lista persoanelor care trebuie să îl contacteze, scrieți un program care stabilește dacă se poate realiza convocarea tuturor persoanelor. Dacă este posibil, să se determine:

a) numărul minim de ore care trec până când poate să înceapă ședința;

b) numărul minim de persoane care trebuie transportate în regim de urgență astfel încât ședința să poată începe cu cel puțin o oră mai devreme; numim transport în regim de urgență un transport în care, în locul timpului dat, transportul durează cu cel puțin o oră mai puțin;

c) numerele de ordine ale persoanelor care vor fi transportate în regim de urgență.

**Date de intrare:**

Fișierul de intrare SPIONI.IN, conține:

pe prima linie, numărul  $n$  de spioni;

pe a doua linie,  $n$  numere naturale  $t_1, t_2, \dots, t_n$ , despărțite prin câte un spațiu, numere reprezentând durata deplasării fiecărui spion până la bază (în regim normal), durată exprimată în ore;

pe fiecare dintre următoarele  $n$  linii, câte o succesiune de minim 0 și maxim  $n$  numere naturale despărțite prin câte un spațiu, numerele de pe linia  $i + 2$  a fișierului reprezentând lista persoanelor care trebuie să convoace persoana având numărul  $i$ , pentru ca aceasta să pornească spre bază.

**Datele de ieșire:**

Pe prima linie a fișierului SPIONI.OUT se află numărul  $-1$  dacă nu se poate realiza convocarea, sau numărul de ore după care poate să înceapă ședința, în caz contrar.

În cazul în care convocarea este posibilă, fișierul mai conține:

- pe a doua linie, un număr  $p$ , reprezentând numărul de persoane care, transportate în regim de urgență, pot devansa ora de începere a ședinței cu cel puțin o oră;

- pe a treia linie, numerele de ordine ale persoanelor care trebuie să fie transportate în regim de urgență, pentru a putea începe ședința mai devreme, numere

despărțite prin câte un spațiu.

**Restricții:**

$$1 \leq n \leq 100$$

$$2 \leq t_i \leq 500$$

**Exemplu:**

SPIONI.IN	SPIONI.OUT poate conține:
5	18
9 8 3 9 1	1
	4
1 3 5	
2 3	

**Timp maxim de execuție pe test:** 3 secunde

### 7.5.1 Indicații de rezolvare - descriere soluție

### 7.5.2 Rezolvare detaliată

### 7.5.3 Codul sursă

## 7.6 Tezaur - ONI 2000

*prof. Doru Anastasiu Popescu, Liceul "Radu Greceanu", Slatina*

Într-un document arheologic recent descoperit se face referire la un mare tezaur. Datorită faptului că documentul poate fi interpretat în mai multe moduri se face apel la  $n$  arheologi care vor studia independent documentul.

La terminarea studiului, fiecare arheolog întocmește o hartă pe care marchează o zonă poligonală închisă și convexă despre care se presupune că este locul unde se află tezaurul.

Deoarece fondurile alocate pentru descoperirea tezaurului sunt reduse, se ia hotărârea să se înceapă cercetările pe teren doar în zona de pe hartă, precizată de toți arheologii.

**Cerință:**

Cunoscând valoarea  $n$  și coordonatele vârfurilor zonelor determinate de arheologi, să se determine aria suprafeței de pe hartă precizată de toți arheologii (intersecția celor  $n$  zone).

**Date de intrare:**

Fișierul de intrare TEZAUR.IN conține:

$n$  - numărul de zone

$m[1]$  - numărul de vârfuri pentru zona primului arheolog

$x_{11}y_{11} \dots x_{1m[1]}y_{1m[1]}$  - coordonatele vârfurilor zonei primului arheolog (date în sens invers acelor de ceasornic)

...

$m[n]$  - numărul de vârfuri pentru zona ultimului arheolog

$x_{n1}y_{n1} \dots x_{nm[n]}y_{nm[n]}$  - coordonatele vârfurilor zonei celui de-al  $n$ -lea arheolog (date în sens invers acelor de ceasornic)

**Date de ieșire:**

Pe prima linie a fișierului TEZAUR.OUT se va scrie aria zonei de pe hartă, precizată de toți arheologii, cu două zecimale. Dacă nu există suprafață comună zonelor celor  $n$  arheologi, în fișierul de ieșire se va scrie numărul 0.

**Exemplu:**

TEZAUR.IN	TEZAUR.OUT
3	400.00
4	
-20 30 -20 -20 40 -20 40 30	
5	
10 80 10 -30 60 -50 100 60 70 80	
4	
20 10 70 10 70 60 20 60	

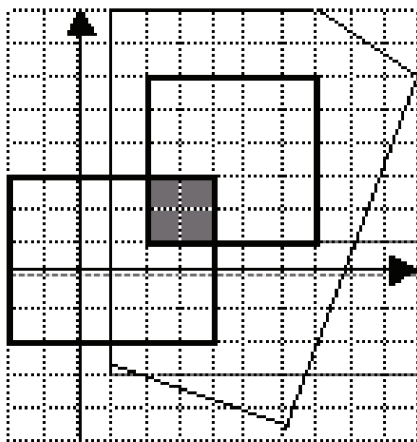
**Restricții:**

$1 \leq n \leq 30$

$3 \leq m[i] \leq 20, i \in \{1, 2, \dots, n\}$

Coordonatele vârfurilor zonelor sunt numere întregi din intervalul  $[-1000, 1000]$

Numărul din fișierul de ieșire se va scrie cu 2 zecimale.



**Timp maxim de execuție pe test:** 1 secundă

**7.6.1 Indicații de rezolvare - descriere soluție****7.6.2 Rezolvare detaliată****7.6.3 Codul sursă**

## Capitolul 8

# ONI 2001 clasa a XI-a



### 8.1 Comparări - ONI 2001

**Bogdan Dumitru, București**

Fie un șir de  $N$  numere întregi. Numim **al doilea maxim** al șirului cel mai mare număr din șir, cu excepția maximumului. Pentru a afla maximumul și al doilea maxim, veți efectua comparări între elementele șirului.

Având la dispoziție numai  $N + \lceil \log_2 N \rceil - 2$  comparări, aflați pozițiile pe care le ocupă maximumul și al doilea maxim. Prin  $\lceil \log_2 N \rceil$  s-a notat partea întreagă superioară a lui "logaritm în baza 2 din  $N$ ".

#### **Cerință**

Aflați cele două numere, efectuând cel mult  $N + \lceil \log_2 N \rceil - 2$  comparări. Pentru a putea efectua comparările, comisia vă pune la dispoziție funcția:

**int Compara(int i, int j)** (pentru cei care programează în C sau C++), respectiv funcția

**Compara(i,j:Integer):Integer;** (pentru programatorii în Pascal), care com-

pară numărul de pe poziția  $i$  cu cel de pe poziția  $j$  și vă returnează 1, dacă elementul aflat pe poziția  $i$  este mai mare sau egal cu cel de pe poziția  $j$ , respectiv -1, în caz contrar.

Programul trebuie să apeleze la început funcția **GetN** care returnează numărul de elemente din șir. Apelul va fi de forma **N=GetN()**, pentru programatorii în C (și C++), respectiv **N:=GetN**, pentru programatorii în Pascal, unde am presupus că  $N$  este variabilă de tip **int**, respectiv **Integer**, în care este memorată lungimea șirului. Este esențial ca apelul funcției **GetN** să preceadă apelurile funcției **Compara** și în plus, să existe un singur apel al funcției **GetN** (în caz contrar programul va primi 0 puncte pe acel test).

În directorul **C:\OLIMP** programatorii în C vor găsi fișierul **Compar.h**, iar cei care lucrează în Pascal vor găsi fișierul **Compar.pas** reprezentând sursele unit-ului comisiei.

În fișierul **Compar.out** veți scrie pozițiile pe care se află maximum șirului și al doilea maximum, separate printr-un blank.

Pentru a folosi unitul, trebuie să creați fișierul **Compar.in**, în care veți scrie pe prima linie numărul  $N$ , iar pe a doua linie elementele șirului, separate prin câte un spațiu.

Versiunea Pascal a unit-ului este:

```

Unit compar;

interface

const my_nmax=1000;
var my_a:array[1..my_nmax+20] of Integer;
    my_apelat, my_n, my_napel, my_apel : Integer;
function GetN:Integer;
function compara(i,j:Integer):Integer;

implementation

function GetN:Integer;
var f:Text;
    i,log,tmp:Integer;
begin
    my_apelat:=1;
    Assign(f,'COMPAR.IN');
    Reset(f);
    Readln(f,my_n);
    for i:=1 to my_n do
        Read(f,my_a[i]);
        Close(f);
    my_apel:=0;
    tmp:=1;
    log:=0;

```



```

    while tmp;my_n do
    begin
        tmp:=tmp*2;
        log:=log+1
    end;
    my_napel:=my_n+log-2;
    getn:=my_n
end;

function compara(i,j:Integer):Integer;
var f:Text;
begin
    if my_apelat=0 then
    begin
        Writeln('Nu ati apelat functia GetN!');
        Halt
    end;
    my_apelat:=my_apelat+1;
    if my_apelat>my_napel then
    begin
        Writeln('Prea multe apeluri!');
        Halt
    end;
    if (i<1) or (i>my_n) or (j<1) or (j>my_n) then
    begin
        Writeln('Apel ilegal!');
        Halt
    end;
    if my_a[i]>=my_a[j] then
        compara:=1
    else compara:=-1
    end;
end;

End.

```

Implementarea C/C++ a aceluiași funcții este următoarea:

```

#include<stdio.h>
#include<math.h>
#include<process.h>
#define my_nmax 1000

int my_a[my_nmax+20];
int my_apelat, my_n, my_napel, my_apel;

```

```
extern int GetN(void)
{
    FILE *f;
    int i,log,tmp;
    my_apelat=1;
    f=fopen("COMPAR.IN","r");
    fscanf(f,"%d",&my_n);
    for (i=1;i<=my_n;i++)
        fscanf(f,"%d",&my_a[i]);
    fclose(f);
    my_apel=0;
    tmp=1;
    log=0;
    while(tmp<my_n)
    {
        tmp=tmp*2;
        log++;
    }
    my_napel=my_n+log-2;
    return my_n;
}

extern int Compara(int i, int j)
{
    FILE *f;
    if (!my_apelat)
    {
        puts("Nu ati apelat functia GetN(!");
        exit(0);
    }
    my_apel=my_apel+1;
    if(my_apel<my_napel)
    {
        puts("Prea multe apeluri!");
        exit(0);
    }
    if(i<1 || i>my_n || j<1 || j>my_n)
    {
        puts("Apel ilegal!");
        exit(0);
    };
    if(my_a[i]>=my_a[j])
        return 1;
    else return -1;
}
```

}

**Restricții** $2 \leq N \leq 1000$ **Exemplul 1**

Dacă șirul considerat este (5, 3, 4), atunci o execuție corectă a programului este următoarea:

...;

N = GetN(), obțineți  $N = 3$ ;

Compara(1, 2), se returnează 1;

Compara(1, 3), se returnează 1;

Compara(2, 3), se returnează -1;

...;

În acest moment ați epuizat numărul de comparații permise ( $3 + 2 - 2 = 3$  comparații) și în fișierul **Compar.out** veți scrie numerele 1 și 3, separate printr-un spațiu, reprezentând pozițiile pe care se află maximul șirului (5), respectiv al doilea maxim (4).

**Exemplul 2**

Dacă șirul considerat este (5, 7, 7), o execuție corectă a programului este următoarea:

...;

N = GetN(), obțineți  $N = 3$ ;

Compara(1, 2), se returnează -1;

Compara(2, 3), returnează 1;

Compara(1, 3), se returnează -1;

...;

În acest moment ați epuizat numărul de comparații permise ( $3 + 2 - 2 = 3$  comparații) și în fișierul **Compar.out** veți scrie numerele 2 și 3, separate printr-un spațiu, reprezentând pozițiile pe care se află maximul șirului (7), respectiv al doilea maxim (7).

**Observație:**

Pentru acest exemplu, mai există o soluție corectă, cea în care considerați maximul (7) pe poziția 3 și al doilea maxim (7) pe poziția 2.

**Timp maxim de execuție/test:** 1 secundă.

Se garantează că 1008 apeluri ale funcției **Compara**, împreună cu un apel al funcției **GetN** nu durează mai mult de 0.1 secunde.

În timpul concursului, pentru a vă putea testa programele, veți avea la dispoziție module *echivalente* cu cele pe care le va utiliza comisia în timpul evaluării programelor voastre.

**8.1.1 Indicații de rezolvare - descriere soluție**

### 8.1.2 Rezolvare detaliată

### 8.1.3 Codul sursă

## 8.2 Relee - ONI 2001

*șef lucrări Stelian Ciurea, Sibiu*

Fie date altitudinile a  $N$  puncte, situate în linie dreaptă de-a lungul axei  $Ox$ , astfel încât ele corespund unor abscise naturale consecutive. Primul punct are abscisa 1. Din acest punct trebuie trimisă o rază laser în ultimul punct (cel de abscisă  $N$ ). Raza se propagă doar în linie dreaptă. Pentru a "ocoli" punctele având altitudini care împiedică trecerea razei, în anumite puncte se montează relee care schimbă unghiul sub care se propagă raza, cu scopul ca ea să poată trece de vârfurile care se află în drumul ei. Releele se vor monta în oricare dintre punctele date, mai puțin în primul punct, de unde raza poate porni sub orice unghi și în ultimul punct unde raza poate fi recepționată sub orice unghi.

S-a observat că dacă în anumite puncte releul se montează pe un pilon, numărul releelor necesare se poate micșora. Toți pilonii care se vor monta au aceeași înălțime dată  $H$ .

#### Cerință

Determinați numărul minim de relee pentru ca raza să ajungă din punctul inițial în cel final, precum și punctele în care acestea se vor monta. În cazul în care există mai multe soluții cu același număr minim de relee, se va alege cea cu număr minim de piloni.

#### Date de intrare

Fișier de intrare: RELEE.IN

Linia 1:  $N H$  două numere naturale nenule, reprezentând numărul punctelor ( $N$ ), respectiv înălțimea pilonilor ( $H$ );

Linia 2:  $A_1 A_2 \dots A_N$   $N$  numere întregi, separate prin câte un spațiu, reprezentând altitudinile (înălțimile) punctelor.

#### Date de ieșire

Fișier de ieșire: RELEE.OUT

Linia 1:  $NR_{relee}$  număr natural nenul, reprezentând numărul releelor care se vor monta, fără să fie înălțate pe piloni;

Linia 2:  $NR_{piloni}$  număr natural nenul, reprezentând numărul pilonilor care se vor monta;

Linia 3:  $C_1 C_2 \dots C_{NR_{relee}}$  numere naturale nenule, reprezentând numărul de ordine al punctelor unde se vor monta relee, fără să fie înălțate pe piloni;

Linia 4:  $D_1 D_2 \dots D_{NR_{piloni}}$  numere naturale nenule, reprezentând numărul de ordine al punctelor unde releele se vor monta pe piloni.

**Restricții**

$$1 \leq N \leq 200$$

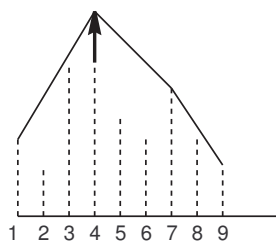
$$1 \leq H \leq 500$$

$$1 \leq A_i \leq 2500, i = 1, 2, \dots, N$$

dacă trei vârfuri sunt coliniare, atunci pe cel din mijloc nu trebuie amplasat un releu.

**Exemplu**

RELEE.IN	RELEE.OUT
9 2	1
3 2 6 6 4 3 5 3 2	1
	7
	4



Liniile punctate reprezintă altitudinea punctelor date. Releu fără pilon se va monta în punctul 7, iar pe pilon în punctul 4.

**Timp maxim de execuție/test:** 1 secundă

**8.2.1 Indicații de rezolvare - descriere soluție****8.2.2 Rezolvare detaliată****8.2.3 Codul sursă****8.3 Telecomanda - ONI 2001**

*prof. Emanuela Cerchez și prof. Marinel Șerban, Iași*

Cu ocazia olimpiadei, televiziunea locală organizează un nou joc în direct. Organizatorii utilizează un calculator, care generează și afișează pe un monitor două numere de maxim 100 de cifre fiecare ( $N1$  și  $N2$ ).

Fiecare concurent dispune de o telecomandă prevăzută cu un afișaj de o cifră și cu anumite taste, ca în figura alăturată. Telecomanda are și o memorie, în care sunt reținute în ordine cifrele obținute de concurenți.

afisaj				
0	1	2	3	4
5	6	7	8	9
+	-	*	/	# =

Cifrele primului număr ( $N1$ ) sunt afișate succesiv pe afișajul telecomenzii fiecărui concurent, în ordine de la stânga la dreapta. Concurenții trebuie să transforme primul număr, obținând în memoria telecomenzii proprii pe cel de al doilea, utilizând tastele pe care le au la dispoziție pe telecomandă. După efectuarea unei operații asupra cifrei curente (cea de pe afișaj), pe afișaj apare automat următoarea cifră din  $N1$  (dacă mai există).

Efectele apăsării tastelor sunt următoarele:

Taste acționate	Efect
+ urmat de o cifră	Se generează suma dintre cifra de pe afișaj și cifra tastată (operație posibilă doar dacă suma este tot o cifră). Cifra sumă este reținută în memorie.
- urmat de o cifră	Se generează diferența dintre cifra de pe afișaj și cifra tastată (operație posibilă doar dacă se obține tot o cifră). Cifra obținută este reținută în memorie.
* urmat de o cifră	Se reține în memorie valoarea tastei care se acționează după tasta *. Deoarece asupra cifrei curente din $N1$ nu se efectuează nici o operație, aceasta nu dispăre de pe afișaj.
/	Se șterge cifra curentă din $N1$
#	Se șterg din $N1$ cifra curentă și toate cifrele care urmează, până la sfârșit.
=	Se copiază în memorie cifra curentă.

Acțiunea se încheie atunci când toate cifrele lui  $N1$  au fost prelucrate. Am obținut o soluție când în memoria telecomenzii se află cifrele numărului  $N2$ . O soluție este optimă dacă numărul de taste acționate este minim. Câștigătorii jocului sunt acei concurenți care descoperă o soluție optimă.

### Cerință

Date fiind  $N1$  și  $N2$ , scrieți un program care să determine o soluție optimă de transformare a numărului  $N1$  în numărul  $N2$ .

### Date de intrare

Fișierul de intrare TELE.IN conține două linii:

$N1$

$N2$

Date de ieșire

Fișierul de ieșire TELE.OUT conține două linii:

$min$

$t_1 t_2 \dots t_{min}$

unde:

$min$  este un număr natural nenul, reprezentând numărul minim de taste acționate pentru transformarea lui  $N1$  în  $N2$ .

$t_1 t_2 \dots t_{min}$  este o succesiune de  $min$  caractere, care reprezintă tastele acționate; între caractere nu se vor pune separatori.

**Exemplu**

TELE.IN	TELE.OUT
372	4
78	/=+6

**Timp maxim de execuție/test:** 1 secundă

### 8.3.1 Indicații de rezolvare - descriere soluție

### 8.3.2 Rezolvare detaliată

### 8.3.3 Codul sursă

## 8.4 Entries - ONI 2001

*Dumitru Bogdan, București*

Se consideră un graf care inițial este format din  $P$  noduri izolate, etichetate de la 1 la  $P$ . Se mai consideră  $N$  intrări, unde intrare poate însemna:

**comandă** - o comandă are forma '**I+J**', cu semnificația că în graf se adaugă muchia care unește nodurile  $I$  și  $J$  (dacă  $I$  și  $J$  erau deja unite în acel moment, nu se întreprinde nici o acțiune);

**întrebare** - o întrebare este de forma '**I?J**', adică se întreabă dacă în acel moment  $I$  și  $J$  sunt în aceeași componentă conexă.

Se pleacă deci de la un graf inițial format din noduri izolate, care pe parcurs se "unifică". Tot pe parcurs sunteți întrebat dacă anumite perechi de noduri sunt sau nu în aceeași componentă conexă.

Din fișierul ENTRIES.IN veți citi de pe prima linie numărul  $N$  de intrări. Pe următoarele  $N$  linii se găsesc intrările, câte una pe linie. O intrare este codificată

prin trei numere separate prin câte un blank. Primele două numere reprezintă nodurile  $I$  și  $J$  (numere întregi, cuprinse între 1 și  $P$ ), iar al treilea este 1 dacă intrarea este o comandă, respectiv 2 dacă intrarea este o întrebare.

La fiecare întrebare, veți scrie pe o linie separată în fișierul ENTRIES.OUT numărul 1 dacă nodurile despre care ați fost întrebat sunt în acel moment în aceeași componentă conexă, respectiv numărul 0 în caz contrar.

**Restricții**

$$1 \leq N \leq 5000$$

$$1 \leq P \leq 10000000$$

**Exemplu**

ENTRIES.IN	ENTRIES.OUT
9	0
1 2 2	0
1 2 1	1
3 7 2	0
2 3 1	1
1 3 2	0
2 4 2	
1 4 1	
3 4 2	
1 7 2	

Timp maxim de executare/test: 1 secundă

#### 8.4.1 Indicații de rezolvare - descriere soluție

#### 8.4.2 Rezolvare detaliată

#### 8.4.3 Codul sursă

### 8.5 Robot - ONI 2001

*prof. Emanuela Cerchez, Iași*

Un robot punctiform se poate deplasa, în plan, în linie dreaptă în orice direcție. Robotul se găsește într-o poziție inițială  $S$  și trebuie să ajungă într-o poziție finală  $F$ , evitând coliziunile cu obstacolele existente în teren. Obstacolele sunt suprafețe poligonale convexe, cu interioarele și frontierele disjuncte. Spunem că robotul a intrat în coliziune cu un obstacol dacă poziția sa devine interioară



obstacolului. Prin urmare, dacă robotul se deplasează de-a lungul unui obstacol, nu intră în coliziune cu acesta.

#### **Cerință**

Scrieți un program care să determine cel mai scurt traseu pe care robotul îl poate urma de la poziția sa inițială  $S$  la poziția sa finală  $F$ , fără a intra în coliziune cu nici un obstacol.

Traseul va fi precizat prin succesiunea punctelor critice (punctul inițial, punctele în care robotul își schimbă direcția și punctul final). Lungimea traseului este egală cu suma lungimilor segmentelor care îl compun.

#### **Date de intrare:**

Fișierul de intrare ROBOT.IN conține:

$x_S y_S$  - coordonatele poziției inițiale a robotului

$x_F y_F$  - coordonatele poziției finale a robotului

$n$  - numărul de obstacole

$k_1$  - numărul de vârfuri ale primului obstacol

$x_1 y_1$

$x_2 y_2$

...

$x_{k_1} y_{k_1}$  - coordonatele vârfurilor primului obstacol

$k_2$  - numărul de vârfuri ale celui de-al doilea obstacol

$x_1 y_1$

$x_2 y_2$

...

$x_{k_2} y_{k_2}$  - coordonatele vârfurilor celui de-al doilea obstacol

...

$k_n$  - numărul de vârfuri ale celui de-al  $n$ -lea obstacol

$x_1 y_1$

$x_2 y_2$

...

$x_{k_n} y_{k_n}$  - coordonatele vârfurilor celui de-al  $n$ -lea obstacol

#### **Date de ieșire:**

Fișierul de ieșire ROBOT.OUT conține traseul robotului codificat ca o succesiune de puncte între care robotul se mișcă în linie dreaptă:

$nr$  - numărul de puncte de pe traseu

$x_S y_S$  - coordonatele punctului inițial

$x_1 y_1$  - coordonatele primului punct critic de pe traseu

$x_2 y_2$  - coordonatele celui de-al doilea punct critic de pe traseu

...

$x_F y_F$  - coordonatele punctului final

#### **Restricții:**

$n$  număr natural,  $0 \leq n \leq 50$

$k_1 + k_2 + \dots + k_n \leq 200$

$x_i, y_i$  sunt numere reale,  $|x_i|, |y_i| \leq 100000$

punctele  $S$  și  $F$  nu se află în interiorul nici unui obstacol

coordonatele se vor afișa în fișierul de ieșire cu trei zecimale semnificative.

**Exemplu:**

ROBOT.IN	ROBOT.OUT
10 5	3
-10 -10	10 5
1	10.5 0
3	-10 -10
0 0	
0 10	
10.5 0	

**Observație:** Dacă există mai multe trasee de lungime minimă, în fișierul de ieșire se va obține o singură soluție.

**Timp maxim de execuție:** 1 secundă/test.

### 8.5.1 Indicații de rezolvare - descriere soluție

### 8.5.2 Rezolvare detaliată

### 8.5.3 Codul sursă

## 8.6 Text mare - ONI 2001

*Radu Ștefan, Brașov*

Se dă un șir de caractere, ce reprezintă o frază; între cuvinte nu apar spații de separare. De asemenea se dă un vocabular având cel mult 32000 de cuvinte; fiecare cuvânt este format din cel mult 16 caractere. Fraza poate avea cel mult 32000 caractere, ce sunt litere mici din alfabetul latin. Cuvintele din vocabular nu sunt neapărat diferite și nu apar într-o ordine prestabilită.

**Cerință**

Despărțiți fraza într-un număr minim de cuvinte; toate aceste cuvinte trebuie să existe în vocabularul dat.

**Date de intrare**

Fișier de intrare: TEXTMARE.IN

pe prima linie apare fraza care trebuie despărțită în cuvinte, terminată cu un punct;

următoarele linii conțin câte un cuvânt din vocabular;

fișierul se termină cu o linie liberă.

**Date de ieșire**

Fișier de ieșire: TEXTMARE.OUT

Fișierul este format dintr-o singură linie, pe care apare fraza despărțită în cuvinte, urmată imediat de un punct. Între oricare două cuvinte consecutive va apărea exact câte un blank.

**Restricții și precizări:**

cel puțin jumătate din teste vor avea mai puțin de 1000 de caractere în frază și cel mult 1000 de cuvinte în vocabular;

dacă există mai multe soluții, la ieșire va fi produsă una singură;

dacă nu există soluție, în fișierul de ieșire se va scrie doar cifra 0 (zero);

într-o frază un cuvânt poate să apară de mai multe ori, fiecare apariție a sa fiind numărată.

**Exemplu**

TEXTMARE.IN	TEXTMARE.OUT
acestaesteuntext.	acesta este un text.
text	
acesta	
acest	
a	
care	
este	
un	
simplu	

Timp maxim de executare/test: 2 secunde.

**8.6.1 Indicații de rezolvare - descriere soluție****8.6.2 Rezolvare detaliată****8.6.3 Codul sursă**



## Capitolul 9

# ONI 2002 clasa a XI-a



### 9.1 Arbore - ONI 2002

Să considerăm un arbore cu  $N$  vârfuri, numerotate de la 1 la  $N$ .

#### Cerință

Scrieți un program care să adauge, dacă este posibil, un număr minim de muchii astfel încât fiecare vârf să aparțină exact unui singur ciclu.

#### Date de intrare

Fișierul de intrare ARBORE.IN conține:

ARBORE.IN	Semnificație
$N$	numărul de vârfuri din arbore
$x_1 y_1$	
$x_2 y_2$	
...	
$x_{N-1} y_{N-1}$	$x_i$ și $y_i$ sunt extremitățile muchiei $i$

#### Date de ieșire

Fișierul de ieșire ARBORE.OUT va conține pe prima linie valoarea  $-1$  dacă problema nu admite soluție, respectiv numărul de muchii adăugate, dacă problema

admite soluție. Dacă problema admite soluție, pe fiecare dintre următoarele linii se vor scrie extremitățile unei muchii adăugate, separate printr-un spațiu, sub forma:

ARBORE.OUT	Semnificație
$Nr$	numărul de muchii adăugate
$a_1 b_1$	
$a_2 b_2$	
...	
$a_{Nr} b_{Nr}$	$a_i$ și $b_i$ sunt extremitățile unei muchii adăugate

### Restricții

$$3 \leq N \leq 100$$

$x_i, y_i$  sunt numere întregi din intervalul  $[1, N]$ .

### Exemple

ARBORE.IN	ARBORE.OUT	ARBORE.IN	ARBORE.OUT
4	-1	7	2
1 2		1 2	6 7
2 3		1 3	4 2
2 4		3 5	
		3 4	
		5 6	
		5 7	

**Timp maxim de executare:** 1 secundă/test

**Observație.** Punctajul pentru testele care nu admit soluție se va acorda dacă și numai dacă la un test care admite soluție s-a răspuns corect.

## 9.1.1 Indicații de rezolvare - descriere soluție \*

*Emanuela Cerchez*

1. Vom selecta un vârf neterminal și vom considera acest vârf rădăcina arborelui.

2. Un vârf este considerat potențial "rezolvabil" dacă și numai dacă are ca subarbori numai vârfuri terminale sau "fire" (subarbore care este lanț).

3. Cât timp nu am rezolvat toate vârfurile (există vârfuri care nu aparțin unui ciclu) și nici nu am depistat o situație nerezolvabilă execut:

- identific vârfurile potențial rezolvabile; fie  $x$  un astfel de vârf

- analizez cazurile următoare (ceva mai rafinat decât explic)

I.  $x$  are mai mult de 2 fii terminali  $\Rightarrow$  nu există soluție

II.  $x$  are exact 2 fii și aceștia sunt terminali  $\Rightarrow$  unesc cei doi fii printr-o muchie; am obținut un ciclu cu 3 vârfuri care îl "rezolvă" pe  $x$  și cei doi fii ai săi;

III.  $x$  are 2 fii terminali dar și alte "fire": unesc cei doi fii terminali (rezultă un ciclu de lungime 3 în care intră  $x$  și cei doi terminali), apoi rezolv firele (unesc printr-o muchie primul și ultimul vârf de pe "fir"; evident acest lucru este posibil dacă toate firele au lungimea mai mare decât 2);

IV.  $x$  are 1 fiu terminal și evident alte "fire": unesc vârful terminal cu extremitatea finală a unuia dintre fire (cel de lungime 2 dacă există, oricare altul dacă nu există), apoi rezolv firele (acestea trebuie să aibă lungime mai mare decât 2)

V.  $x$  nu are fii terminali, numai fire; rezolv firele (dacă există mai mult de 2 fire de lungime 2 nu există soluție), unind între ele extremitățile finale a două fire (cele de lungime 2 dacă există, sau oricare altele), iar restul le rezolv independent (unind extremitatea lor inițială cu extremitatea lor finală).

### 9.1.2 Rezolvare detaliată

### 9.1.3 Codul sursă

## 9.2 Decod - ONI 2002

Serviciul Român de Informații (SRI) a dat de urma unei binecunoscute și periculoase organizații teroriste, care își are sediul pe teritoriul țării noastre. Folosind cei mai pricepuți și mai bine antrenați spioni și ofițeri, SRI a reușit să identifice computerul principal al organizației teroriste. Dacă va reuși să acceseze informațiile din acest computer, SRI îi va putea aresta pe toți membrii organizației și va asigura (în continuare) pacea mondială. Singura problemă este spargerea codului de acces. Tot ceea ce se știe despre acest cod este că el este reprezentat de către o permutare de lungime  $N$ . Specialiștii SRI au încercat diverse metode de a descoperi codul, însă tot ceea ce au reușit să obțină este un program care, transmițându-i-se ca parametru o permutare de lungime  $N$ , specifică în câte poziții coincid această permutare și codul de acces. Din păcate, șefii nu cred în utilitatea acestui program.

#### Cerință

Scrieți un program care, folosind programul-ajutător (reprezentat sub forma unui modul), să determine codul de acces în computerul teroriștilor.

#### Date de intrare

Programul dumneavoastră nu va citi date din vreun fișier de intrare. El va apela întâi funcția **GetN** a modulului **PROG**, care îi va returna valoarea  $N$  - numărul de elemente ale permutării ce trebuie descoperită.

Apoi va apela numai funcția **Check**, căreia îi va fi transmisă ca parametru, de fiecare dată, o permutare de lungime  $N$ . Această funcție va returna numărul de poziții în care permutarea transmisă ca parametru coincide cu permutarea ce trebuie descoperită. Programul dumneavoastră trebuie ca, după un număr finit de apelări ale funcției **Check**, să descopere permutarea căutată.

#### Date de ieșire

Programul dumneavoastră va trebui să tipărească în fișierul DECOD.OUT o permutare de lungime  $N$ . Toate cele  $N$  elemente vor fi tipărite pe prima linie a fișierului, fiind separate de câte un spațiu.

**Restricții:**

$$5 \leq N \leq 256$$

**Instrucțiuni pentru programatorii în C/C++**

Veți avea la dispoziție header-ul **prog.h**, pe care va trebui să-l includeți în sursa dumneavoastră. Acest fișier declară următoarele funcții:

```
int GetN ( void )
```

```
int Check ( int p[256] )
```

Funcția **Check** are ca parametru un vector cu elemente întregi, care reprezintă o permutare. Ea va evalua această permutare și va returna una dintre următoarele valori:

–1      dacă primele  $N$  elemente întregi (începând cu poziția 0) ale vectorului transmis ca parametru nu constituie o permutare de lungime  $N$

0 –  $N$       numărul de poziții în care coincide permutarea de lungime  $N$  transmisă ca parametru, cu permutarea ce trebuie descoperită

**Instrucțiuni pentru programatorii în PASCAL**

Modulul **PROG** este implementat sub forma unit-ului **PROG**. În acest unit sunt definite următoarele tipuri și funcții, care vor fi folosite de către programul dumneavoastră:

tipuri:

```
perm = array [ 1..256 ] of integer;
```

funcții:

```
function GetN : integer;
```

```
function Check ( var permut : perm ) : integer;
```

Funcția **Check** are ca parametru un vector de tipul **perm**. Ea va evalua această permutare și va returna una din următoarele valori:

–1      dacă primele  $N$  elemente întregi ale vectorului transmis ca parametru nu constituie o permutare de lungime  $N$

0 –  $N$       numărul de poziții în care coincide permutarea de lungime  $N$  transmisă ca parametru, cu permutarea ce trebuie descoperită

**Exemplu de apel**

<i>/* C/C++ */</i>	{ Pascal }
<code>int p[256];</code>	<code>var p: perm;</code>
<code>...</code>	<code>...</code>
<code>tmp = Check(p);</code>	<code>tmp := Check(p);</code>

Pentru a vă testa programul, creați un fișier numit DECOD.IN. Pe prima linie a acestui fișier scrieți valoarea  $N$ . Pe a doua linie scrieți o permutare de lungime  $N$ , având elementele separate prin spațiu. Modulul **PROG** va citi această permutare din fișierul DECOD.IN (din directorul curent) și o va considera permutarea ce trebuie descoperită?

**Exemplu**



DECOD.IN
5
2 1 3 4 5

O serie posibilă de apeluri ale funcțiilor **GetN** și **Check** este următoarea:

- Se apelează funcția **GetN** și se returnează valoarea 5
- Se apelează funcția **Check** cu permutarea (1 2 3 4 5) și se returnează valoarea 3
- Se apelează funcția **Check** cu permutarea (3 2 1 4 5) și se returnează valoarea 2
- Se apelează funcția **Check** cu permutarea (2 1 3 4 5) și se returnează valoarea 5
- Se tipărește în fișierul **DECOD.OUT** permutarea 2 1 3 4 5.

Fișierul **DECOD.OUT** va arăta astfel:

2 1 3 4 5 - pe prima linie a fișierului

**Timp maxim de executare:** 1 secundă/test

**Notă:** Pe calculatorul dumneavoastră va fi instalat o versiune a modului **PROG** în directorul `c:\decod\`, pe care o puteți folosi pentru testare. Înainte de evaluare, programul dumneavoastră va fi compilat folosind o altă versiune a modului **PROG**. Această versiune nu va necesita un timp mai mare pentru executarea funcțiilor puse la dispoziție decât varianta de pe calculatoarele dumneavoastră.

### 9.2.1 Indicații de rezolvare - descriere soluție \*

*Mugurel Ionuț Andreica*

1. Se determină poziția corectă a unui element din permutare în  $O(N^2)$  întrebări.

2. Utilizând elementul fixat, putem determina elementul care apare pe orice altă poziție în  $O(N)$  întrebări.

Complexitate:  $O(N^2)$  apeluri ale funcției **CHECK**  $\Rightarrow O(N^3)$ .

### 9.2.2 Rezolvare detaliată

### 9.2.3 Codul sursă

### 9.3 Seti - ONI 2002

Se pare că în sfârșit căutătorii vieții extraterestre au descoperit ceva! În cursul proiectului SETI@home a fost izolată o secvență care ar putea reprezenta un semnal de la alte forme de viață inteligentă. Ca urmare, proiectul SETI@ONI își propune să verifice dacă acel semnal provine într-adevăr de la extraterestri sau doar de la niște puști care beau Fanta.

#### Cerință

Pentru comoditate, porțiunea de semnal ce trebuie analizată vi se pune la dispoziție sub forma unei succesiuni de litere ale alfabetului latin. Vi se mai pune la dispoziție și un dicționar de cuvinte extraterestre, codificate în același mod. Scopul dumneavoastră este să numărați de câte ori apare fiecare dintre aceste cuvinte în posibilul mesaj extraterestru. Pornind de la aceste date, lingviștii pot să înceapă lucrul la traducerea mesajului.

#### Date de intrare

Pe prima linie a fișierului de intrare SETI.IN este scris numărul  $N$  de linii ale mesajului. Urmează  $N$  linii, fiecare conținând exact 64 de litere ale alfabetului latin urmate de marcajul de sfârșit de linie. Prin alipirea acestor bucăți se obține mesajul de analizat, format din  $64 * N$  litere.

Pe prima linie a celui de-al doilea fișier de intrare DIC.IN este scris numărul  $M$  de cuvinte din dicționar. Urmează apoi  $M$  linii, fiecare conținând un cuvânt din dicționar, reprezentat ca o secvență de cel puțin una și cel mult 16 litere. Cuvintele nu sunt neapărat distincte.

#### Date de ieșire

Fișierul de ieșire SETI.OUT va conține exact  $M$  linii. Pe linia cu numărul  $i$  va fi scris numărul de apariții în mesajul extraterestru ale cuvântului cu numărul  $i$  din dicționar. Numărul de apariții nu va depăși niciodată 65535. Orice apariție a unui cuvânt trebuie numărată, chiar dacă se suprapune peste alte apariții. Se va face diferență între litere mari și litere mici.

#### Restricții

$$0 \leq N < 2048$$

$$0 \leq M \leq 32000$$

#### Exemplu

SETI.IN
2 aaBaba babaaBaB

DIC.IN
2 3 b bab b

SETI.OUT
3
2
3

**Timpi maxim de executare:** 1 secundă/test

### 9.3.1 Indicații de rezolvare - descriere soluție \*

*Mihai Pătrașcu*

O metodă de a rezolva această problemă este să se ordoneze toate substringurile de lungime 16 din mesajul care trebuie analizat. Apoi, pentru fiecare cuvânt din dicționar se fac două căutări binare. Sunt necesare însă următoarele două rafinări ale ideii:

- sortarea tuturor substringurilor este consumatoare de timp. Dacă se folosesc metode precum **quicksort** sau **heapsort**, este probabil ca programul să nu treacă toate testele. Cea mai eficientă metodă de sortare în cazul de față, care se încadrează fără probleme în timp, este **radixsort**.

- nu se poate ține un tabel cu substringurile după ce au fost sortate. În cel mai bun caz trebuie să reținem o referință către fiecare substring (de exemplu, un pointer sau un indice), dar fiindcă pot fi peste 100,000 de substringuri, referința trebuie să fie de minim 24 de biți, ceea ce face tabloul de dimensiune inacceptabilă. Soluția constă în a împărți arbitrar mesajul în bucăți de 32,768 de caractere, și de a aplica algoritmul menționat pentru fiecare bucată. Bineînțeles, trebuie să ne asigurăm că algoritmul ia în calcul și secvențele conținute în două bucăți alăturate.

### 9.3.2 Rezolvare detaliată

### 9.3.3 Codul sursă

## 9.4 Suma divizorilor - ONI 2002

Se consideră două numere naturale  $A$  și  $B$ . Fie  $S$  suma tuturor divizorilor naturali ai lui  $A^B$  ( $A$  la puterea  $B$ ).

#### **Cerintă**

Sa se afișeze restul împărțirii lui  $S$  la 9901.

#### **Date de intrare**

Pe prima linie a fișierului de intrare SUMDIV.IN sunt scrise cele două numere  $A$  și  $B$ , separate prin cel puțin un spațiu.

#### **Date de ieșire**

Prima linie a fișierului SUMDIV.OUT va conține restul împărțirii lui  $S$  la 9901.

**Restricții**

$0 \leq A, B \leq 50000000$  (cincizeci de milioane)

**Exemplu**

SUMDIV.IN	SUMDIV.OUT
2 3	15

**Explicație**

$2^3 = 8$ .

Divizorii naturali ai lui 8 sunt: 1, 2, 4, 8. Suma lor este 15.

Restul împărțirii lui 15 la 9901 este 15 (care trebuie să apară în fișierul de ieșire).

**Timp maxim de executare:** 0.5 secunde (500 de milisecunde)/test

**9.4.1 Indicații de rezolvare - descriere soluție \***

*Mihai Pătrașcu*

Să presupunem că avem  $A=12$  și  $B=3$ .

Evident,  $12^3 = (2^2 \cdot 3)^3 = 2^6 \cdot 3^3$ .

Un divizor al acestui număr are forma:  $2^x \cdot 3^y$ ,  $0 \leq x \leq 6$ ,  $0 \leq y \leq 3$ .

Deci suma divizorilor este:

$$\sum_{x=0,6;y=0,3} 2^x \cdot 3^y$$

Această sumă dublă se poate rescrie ca:

$$(1 + 2 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6) \cdot (1 + 3 + 3^2 + 3^3)$$

Există mai multe metode de a evalua în timp logaritmic progresiile geometrice care apar. Una dintre ele folosește în mod esențial faptul ca 9901 este prim.

**Testarea**

Test	A	B	S % 9901
0	1	45,000,963	1
1	34,002,980	0	1
2	9,901	999	1
3	1,576	1	2970
4	$2^{25} = 33,554,432$	$2^{25} = 33,554,432$	3612
5	$24,999,983 \cdot 2 = 49,999,966$	$2^{25} - 1 = 33,554,431$	1739
6	$2 \cdot 3 \cdot 5 \cdot 7 \cdot 59,407 = 12,475,470$	90,000	9879
7	$22 \cdot 3 \cdot 3,980,203 = 47,762,436$	137	3575
8	$33 \cdot 52 \cdot 73 \cdot 11 \cdot 17 = 43,295,175$	25,983,137	3816
9	$33 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 = 43,648,605$	49,999,801	6407

**Lămuriri:**

- fie  $p$  cel mai mare factor prim al lui  $A$ ; factorizarea trebuie să fie  $O(\sqrt{p})$ ; dacă este  $O(p)$ , testele 5, 7 și poate 1, nu vor fi trecute;
- dacă folosiți o metodă în timp liniar pentru a evalua progresia geometrică, testele 4, 5, 8, 9 și poate 6 nu vor fi trecute;
- un caz particular important pentru progresia geometrică este  $q = 1$ ; dacă ați omis acest caz, nu veți obține punctaj pentru testele 6 și 7 (59407 % 9901 == 1).

### 9.4.2 Rezolvare detaliată

### 9.4.3 Codul sursă

## 9.5 Sistem - ONI 2002

Județul în care are loc Olimpiada Națională de Informatică de anul acesta este puțin ciudat. În județ există  $N$  orașe, numerotate de la 1 la  $N$ . Fiecare dintre cele  $N$  orașe ale județului este legat de EXACT alte 2 orașe, prin străzi bidirecționale. Și mai ciudat este faptul că, în cadrul acestui sistem stradal, nu este întotdeauna posibil să ajungi din orice oraș în oricare alt oraș mergând pe străzi. Oricum, locuitorii județului sunt mândri de acest sistem al lor și sunt de părere că nu mai există altul la fel. Dumneavoastră vreți să le demonstrați contrariul și pentru aceasta vreți să calculați câte sisteme stradale distincte cu proprietatea de mai sus există. Două sisteme sunt considerate distincte dacă există cel puțin o stradă între o pereche de orașe  $i$  și  $j$  în cadrul primului sistem, care nu există în cadrul celui de-al doilea.

#### Cerință

Scrieți un program care să calculeze câte sisteme stradale distincte există.

#### Date de intrare

Din fișierul SISTEM.IN veți citi valoarea întreagă  $N$ , reprezentând numărul de orașe ale județului.

#### Date de ieșire

În fișierul SISTEM.OUT veți afișa o valoare întreagă, reprezentând numărul de sisteme stradale distincte, cu proprietatea că orice oraș este legat prin străzi directe de exact alte 2 orașe.

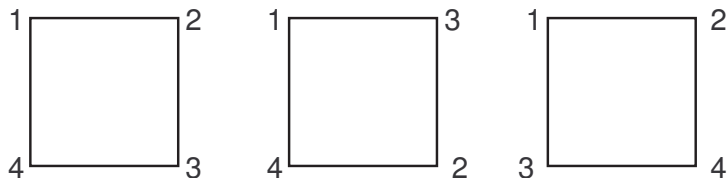
#### Restricții

$$3 \leq N \leq 100$$

#### Exemplu

SISTEM.IN	SISTEM.OUT
4	3

Cele 3 soluții sunt următoarele:



SISTEM.IN	SISTEM.OUT
6	70

**Timp maxim de executare:** 1 secundă/test

### 9.5.1 Indicații de rezolvare - descriere soluție \*

*Mugurel Ionuț Andreica*

Notăm cu  $G_i$  numărul de grafuri 2-regulate (în care fiecare nod are gradul 2). Avem:

$$G_0 = 1; G_1 = 0; G_2 = 0; G_3 = 1$$

Pentru  $i > 3$ , formula de calcul este următoarea:

$$G_i = \sum G_{i-k} * C(i-1, k-1) * (k-1)!/2, \quad k = 3, \dots, i$$

unde  $C(i, j)$  reprezintă combinații de  $i$  luate câte  $j$ .

### 9.5.2 Rezolvare detaliată

### 9.5.3 Codul sursă

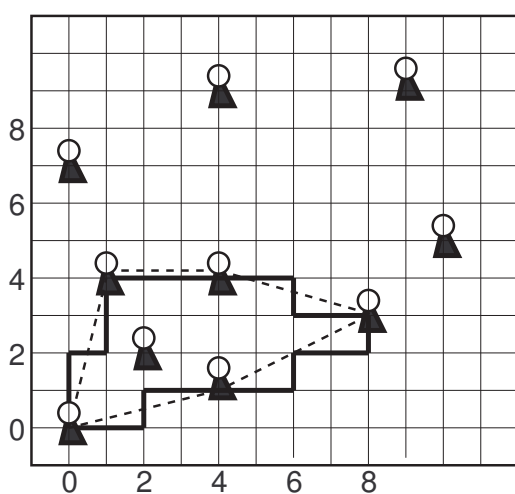
## 9.6 Comitat - ONI 2002

Toate semințiile conviețuitoare pe Terra au hotărât ca hobbitii, păstrătorii Inelului Puterii, să fie izolați într-o zonă a pământului numită Comitat. Hotarele Comitatului trebuie să fie reprezentate de un poligon convex cu câte un turn de pază în fiecare vârf.

Se cunosc pozițiile tuturor turnurilor din regiune (două numere naturale raportate la un sistem de axe rectangulare). Un paznic pe cal alb veghează hotarele comitatului parcugând, pe rând, toate distanțele dintre două turnuri succesive mergând pe drum minim, numai pe cărări paralele cu axele sistemului de axe.

Se cunoaște lungimea maximă a drumului pe care-l poate parcurge paznicul la un tur complet al hotarelor comitatului și se cere să se determine un poligon cu un număr maxim de turnuri pe contur, poligon ce poate constitui hotarul comitatului. În plus, hotarul trebuie să conțină turnul din Mordor (de coordonate 0 și 0) într-un vârf, în fiecare dintre celelalte vârfuri aflându-se obligatoriu unul din turnurile existente.

De exemplu, pentru amplasamentul turnurilor ilustrat alăturat și pentru limita de 25 Km a unui tur efectuat de paznic, hotarul comitatului poate fi format, în această ordine, din turnurile de coordonate  $(0, 0)$ ,  $(4, 1)$ ,  $(8, 3)$ ,  $(4, 4)$ ,  $(1, 4)$ ,  $(0, 0)$ . Se observă că poligonul determinat de aceste turnuri este un poligon convex cu 5 turnuri pe contur.



Poligonul cu vârfurile  $(0, 0)$ ,  $(4, 1)$ ,  $(4, 12)$ ,  $(0, 7)$ ,  $(0, 0)$  are tot 5 turnuri pe contur, dar un tur complet al acestui poligon depășește 25 Km.

#### Date de intrare

Din fișierul COMITAT.IN se citesc, în ordine:

COMITAT.IN	Semnificație
$n$	numărul de turnuri din ținut
$x_1 y_1$	(cu excepția turnului implicit - Mordor)
$x_2 y_2$	
...	
$x_n y_n$	coordonatele (abscisă ordonată)
$L$	ale fiecăruia dintre cele $n$ turnuri
	numărul reprezentând lungimea maximă
	a unui tur complet al poligonului

#### Date de ieșire

În fișierul COMITAT.OUT se scriu:

COMITAT.IN	Semnificație
$v$ $t_1 t_2 \dots t_{v-1}$	numărul de turnuri de pe conturul poligonului numerele de ordine (în ordinea din fișierul de intrare) ale turnurilor de pe contur, pornind de la turnul Mordor (care este implicit) și respectând succesiunea în sens trigonometric sau în sensul acelor de ceasornic a turnurilor de pe contur

**Observații**

- Pot exista turnuri strict în interiorul poligonului, dar acestea nu sunt luate în considerare pentru criteriul de maxim.

- Se consideră soluții și poligonul degenerat format dintr-un singur vârf (Mordor) sau din două vârfuri (Mordor și un alt turn) sau din mai multe vârfuri coliniare.

- Pot exista turnuri coliniare pe conturul poligonului determinat.

- Dacă există mai multe soluții ce respectă condițiile din enunț, se va furniza doar una dintre acestea.

- În fișierul de intrare nu există două turnuri ale căror poziții să coincidă și nu există un turn în poziția (0, 0).

**Restricții**

$$0 < n \leq 50$$

$$0 \leq x_i, y_i \leq 200$$

$$0 < L < 1000$$

**Exemplu**

COMITAT.IN (conform figurii)	COMITAT.OUT (o soluție posibilă)
9	5
0 7	4 7 5 2
1 4	
2 2	
4 1	
4 4	
4 9	
8 3	
9 9	
10 5	
25	

**Timp maxim de executare:** 1 secundă/test.

**9.6.1 Indicații de rezolvare - descriere soluție \***

*Mugurel Andreica și Rodica Pinte*

O rezolvare de programare dinamică presupune construirea unui tablou alocat dinamic ce reține pentru fiecare triplet  $i, j, k$ :

- lungimea liniei convexe cu  $k$  turnuri pe contur, linie ce porneste din turnul de ordin 0 (Mordor) și are ultima latură determinată de turnurile de ordin  $i$  și  $j$



$lmin[i,j,k]=\min(lmin(j,p,k-1)+distM(i,j))$  astfel încât  $(p,j,i)=colț$  convex

### 9.6.2 Rezolvare detaliată

### 9.6.3 Codul sursă



## Capitolul 10

# ONI 2003 clasa a XI-a



### 10.1 Asmin - ONI 2003

Se consideră un arbore (graf conex aciclic) cu  $N$  vârfuri, fără rădăcină fixată. Drept rădăcină, poate fi ales oricare dintre vârfuri. Să presupunem că a fost ales vârful cu numărul  $T$ . Între oricare vârf și  $T$  există un drum unic care conține fiecare vârf al arborelui cel mult o singură dată (un drum între vârfurile  $i$  și  $j$  este o secvență de vârfuri, care începe cu  $i$ , se termină cu  $j$ , iar între oricare două vârfuri consecutive există o muchie în arbore). Fiecărui vârf  $i$  (inclusiv  $T$ ) trebuie să i se asocieze o valoare  $V_i$ , mai mare sau egală cu 0, astfel încât suma valorilor vârfurilor de pe drumul dintre  $i$  și rădăcina  $T$ , împărțită la  $K$ , să dea restul  $R_i$ . Se definește costul arborelui cu rădăcina fixată în  $T$ ,  $C_T$ , ca fiind suma valorilor asociate fiecărui nod. Dintre toate posibilitățile de alegere a valorilor  $V_i$  care respectă condiția precizată anterior, se va alege aceea pentru care  $C_T$  este minim.

Se constată ușor că alegând alt vârf drept rădăcină, de exemplu, vârful  $S$  (diferit de  $T$ ),  $C_S$  nu este neapărat egal cu  $C_T$ .

#### **Cerință**

Dându-se un arbore cu  $N$  vârfuri, un număr întreg  $K$  și valorile  $R_i$ ,  $i = 1, 2, \dots, N$ , corespunzătoare fiecărui vârf, determinați acele vârfuri  $T$  care pot fi

alese drept rădăcină, pentru care costul  $C_T$  este minim (adică  $C_T \leq CS$ , oricare ar fi  $S$  diferit de  $T$ ), precum și costul respectiv.

#### Date de intrare

Pe prima linie a fișierului de intrare **asmin.in** se află 2 valori întregi:  $N$  și  $K$ . Pe următoarele  $N - 1$  linii se află câte două numere întregi  $a$  și  $b$ , separate printr-un spațiu, având semnificația că există muchie între vârfurile  $a$  și  $b$ . Vârfurile sunt numerotate de la 1 la  $N$ . Pe următoarea linie se află  $N$  numere întregi, reprezentând valorile  $R_i$ ,  $i = 1, 2, \dots, N$ .

#### Date de ieșire

Pe prima linie a fișierului de ieșire **asmin.out** se vor afișa două valori întregi:  $C$  și  $M$ .  $C$  reprezintă costul minim posibil al arborelui.  $M$  reprezintă numărul de vârfuri care pot fi alese drept rădăcină și pentru care se obține costul  $C$ . Pe a doua linie se află  $M$  numere întregi separate prin câte un spațiu, scrise în ordine crescătoare, reprezentând numerele vârfurilor ce pot fi alese ca rădăcină astfel încât să se obțină costul  $C$ .

#### Restricții și precizări

$$2 \leq N \leq 16000$$

$$2 \leq K \leq 1000$$

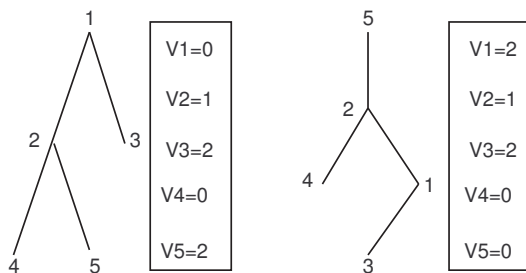
$$0 \leq R_i \leq K - 1$$

Cel puțin 40% din testele folosite la evaluare vor avea  $N \leq 1000$

#### Exemplu

asmin.in	asmin.out
5 3	5 2
1 2	1 5
1 3	
2 4	
2 5	
0 1 2 1 0	

Cei doi arbori obținuți (împreună cu valorile asociate vârfurilor) sunt următorii:



**Tim maxim de execuție:** 0.2 secunde/test

### 10.1.1 Indicații de rezolvare - descriere soluție \*

#### Soluția oficială

Există o soluție simplă, cu complexitatea  $O(N^2)$ . Această soluție încearcă să aleagă fiecare nod drept rădăcină și pentru rădăcina astfel fixată, calculează în  $O(N)$  valorile asociate nodurilor. Dacă rădăcina este nodul  $i$ , atunci valoarea rădăcinii este  $R_i$ . Pentru fiecare alt nod, valoarea asociată lui este cea mai mică valoare astfel încât suma valorilor până la tatăl lui în arborele cu rădăcina fixată plus această valoare să dea restul cerut, la împărțirea cu numărul  $K$ . Valorile sunt atribuite nodurilor de la rădăcina către frunze. Astfel, o simplă parcurgere **DF** este de ajuns, o dată ce rădăcina a fost fixată.

Soluția optimă are complexitatea  $O(N)$ . Se fixează întâi ca rădăcina nodul 1 și se calculează valorile asociate nodurilor arborelui având rădăcina în nodul 1, conform algoritmului descris mai sus. Astfel se obține  $C[1]$  = suma valorilor nodurilor arborelui când nodul 1 este rădăcina. Pentru a calcula  $C[i]$  ( $i$  diferit de 1) nu este nevoie să repetăm parcurgerea **DF** pentru nodul  $i$  considerat rădăcina. În schimb, încă o parcurgere **DF** din nodul 1 este suficientă pentru a calcula costurile fiecărui nod. Să presupunem că vrem să calculăm  $C[i]$  și am calculat deja  $C[j]$ , unde  $j$  este tatăl lui  $i$ , în arborele având rădăcina fixată în nodul 1.

$$\text{Atunci, } C[i] = C[j] - R[j] - \min(R[j], R[i]) + R[i] + \min(R[i], R[j]),$$

unde  $\min(a, b)$  (cu  $0 <= a, b <= K-1$ ) calculează valoarea  $x$  minimă ( $x >= 0$ ), astfel încât  $a + x = b \pmod{K}$ .

Se alege minimul din vectorul  $C$  și se tipăresc nodurile  $i$  având valoarea  $C[i]$  egală cu minimul.

#### Mihai Stroe, Ginfo nr 13/7 - noiembrie 2003

Pentru această problemă există o soluție a cărei ordin de complexitate este  $O(N^2)$ . În continuare prezentăm această soluție.

Se consideră fiecare nod (pe rând) ca fiind rădăcină. Pentru rădăcina astfel fixată, valorile asociate nodurilor se pot calcula în timp liniar. Dacă rădăcină este nodul  $i$ , atunci valoarea rădăcinii este  $R_i$ . Pentru fiecare alt nod, valoarea asociată lui este cea mai mică valoare, astfel încât suma valorilor până la tatăl său în arborele cu rădăcina fixată plus această valoare să dea restul cerut la împărțirea cu numărul  $K$ . Valorile sunt atribuite nodurilor de la rădăcină către frunze. Astfel, o simplă parcurgere *DF* (Depth First -parcurgere în adâncime) este suficientă, odată ce rădăcina arborelui a fost fixată. Evident, această soluție nu se va încadra în timp pe testele mari.

Soluția optimă are ordinul de complexitate  $O(N)$ . Se fixează mai întâi ca rădăcină nodul 1 și se calculează valorile asociate nodurilor arborelui având rădăcina în nodul 1, conform algoritmului descris mai sus. Astfel se obține o valoare  $C_1$

egală cu suma valorilor nodurilor arborelui când nodul 1 este rădăcina. Vom numi această valoare costul asociat nodului 1.

Pentru a calcula  $C_i$  ( $i \neq 1$ ) nu este nevoie să repetăm parcurgerea  $DF$  pentru nodul  $i$  considerat rădăcină. În schimb, încă o parcurgere  $DF$  din nodul 1 este suficientă pentru a calcula costurile asociate fiecărui nod. Să presupunem că vrem să calculăm  $C_i$  și am calculat deja  $C_j$ , unde  $j$  este tatăl lui  $i$ , în arborele având rădăcina fixată în nodul 1.

Atunci,

$$C_i = C_j - R_j - \min(R_j, R_i) + R_i + \min(R_i, R_j),$$

unde  $\min(a, b)$  (cu  $0 \leq a, b \leq K - 1$ ) reprezintă valoarea  $x$  minimă ( $0 \leq x$ ), astfel încât  $a + x = b \pmod{K}$ . Se alege minimul din vectorul  $C$  și se tipăresc nodurile  $i$  având valoarea  $C_i$  egală cu minimul.

### Analiza complexității

Operația de citire a datelor de intrare are ordinul de complexitate  $O(N)$ , iar cea de scriere a rezultatului are ordinul de complexitate  $O(1)$ .

Prima rezolvare constă într-o parcurgere  $DF$  pentru fie-care nod. Deoarece ordinul de complexitate al unei parcurgeri este  $O(N)$ , ordinul de complexitate al acestei soluții este  $O(N^2)$ .

Rezolvarea optimă constă în două parcurgeri  $DF$ , în care se execută calcule elementare. Ordinul de complexitate al unei astfel de parcurgeri este  $O(N)$ .

În concluzie, ordinul de complexitate al soluției optime pentru această problemă este  $O(N) + O(1) + O(N) = O(N)$ .

## 10.1.2 Rezolvare detaliată

### 10.1.3 Codul sursă

## 10.2 Căutare - ONI 2003

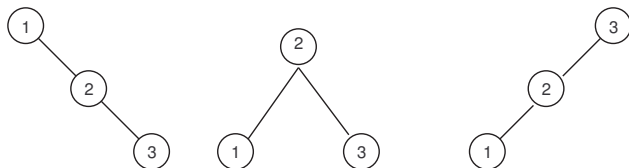
Știm cu toții ce este un arbore binar de căutare. Este acel arbore binar în care informația din orice nod este mai mare decât informațiile nodurilor din subarborele stâng al nodului respectiv și mai mică decât cele din subarborele drept.

Când se caută o informație într-un arbore binar de căutare, începem din rădăcina arborelui și comparăm cu informația din rădăcină. Dacă informația căutată e mai mică decât informația din rădăcină, se continuă căutarea în subarborele stâng, iar dacă e mai mare în subarborele drept. Dacă cele două informații sunt egale căutarea se termină cu succes. Dacă în direcția în care continuăm căutarea (stânga sau dreapta) subarborele nu mai are noduri înseamnă că informația căutată

nu se găsește în arbore. Evident, numărul de comparații efectuate la o căutare depinde de distanța dintre rădăcină și nodul în care se găsește informația, respectiv cel la care putem decide că informația nu se află în arbore.

Ce s-ar întâmpla dacă înainte de a construi arborele binar de căutare am ști ce informații urmează să fie căutate în el? Nu cumva am putea construi arborele în așa fel încât să minimizăm numărul de comparații efectuate?

De exemplu cu informațiile 1, 2 și 3 putem construi un arbore binar de căutare în următoarele 3 moduri (din totalul de 5 posibile):



Dacă vom căuta informațiile (1, 1, 3, 1, 2), atunci timpul total de căutare este pentru arborele din stânga  $1 + 1 + 3 + 1 + 2 = 8$ , pentru arborele din centru  $2 + 2 + 2 + 2 + 1 = 9$ , iar pentru arborele din dreapta  $3 + 3 + 1 + 3 + 2 = 12$ . Deci arborele din stânga este cel adecvat pentru căutările noastre, iar timpul total de căutare minim este 8.

De remarcat că dacă se caută o informație care nu există în arbore atunci numărul de comparații efectuate la căutare este egal cu nivelul ultimului nod interogată. De exemplu, dacă se caută informația 4 pe cei trei arbori atunci timpurile vor fi: 3, 2, respectiv 1 (de la stânga la dreapta).

#### Cerință

Scrieți un program care, pentru anumite informații căutate, determină timpul total de căutare minim.

#### Date de intrare

Din fișierul **cautare.in** se citește de pe prima linie numărul  $T$  de teste. În fișier urmează cele  $T$  teste. Pentru fiecare test în fișier sunt scrise următoarele linii:

- prima linie conține  $N$ , numărul de noduri ale arborelui de căutare și  $M$  numărul de interogări, separate prin spațiu;

- pe următoarea linie urmează  $N$  numere întregi distincte, separate prin câte un spațiu, reprezentând informațiile din nodurile arborelui

- pe fiecare dintre următoarele  $M$  linii sunt câte 2 numere întregi separate printr-un spațiu, reprezentând un număr căutat și respectiv de câte ori a fost căutat (între 1 și 100).

#### Date de ieșire

În fișierul **cautare.out** se va scrie, pentru fiecare test câte o linie care conține timpul total minim de căutare pentru interogările din testul respectiv.

#### Restricții

-  $0 < T, N < 101; 0 < M < 1001$

- informațiile nodurilor arborelui sunt numere întregi din intervalul  $[-10000, 10000]$

- informațiile căutate sunt numere întregi din intervalul  $[-1000000, 1000000]$  și pot fi căutate de un număr de ori cuprins între 1 și 100.

#### Exemplu

cautare.in	cautare.out	Explicații
3	8	Primul test este cel discutat.
3 3	251	În al doilea test se interoghează 1 de
1 2 3	22	50 de ori, 2 de 49 de ori și 3 de 51 de ori.
1 3		Cel mai bun rezultat se obține în cazul
2 1		arborelui din centru, și anume 251
3 1		$(2*50+49+2*51)$ .
3 3		Al treilea test conține o interogare a lui 4
1 2 3		(care nu se află în arbore) de 20 de ori.
1 50		Evident că cel mai bun e arborele din
2 49		dreapta, pe care ne dăm seama repede că
3 51		4 nu se află în arbore. Timpul total este
3 2		$(1*20+2)$ .
1 2 3		
2 1		
4 20		

#### Observații:

Pentru un fișier se ia punctajul maxim dacă toate testele din fișier sunt corecte, altfel 0 puncte

Pentru 5 fișiere de test (din 10)  $T < 4$ .

**Timpul maxim de execuție** pentru un fișier de test: 1.5 secunde/Windows respectiv 0.3 secunde/Linux.

### 10.2.1 Indicații de rezolvare - descriere soluție \*

#### Soluția oficială

Se citesc nodurile, se sortează.

Se citesc interogările și (prin căutări binare) se adaugă la nodul respectiv sau la intervalul dintre nodurile respective. Astfel se obțin niște ponderi ale nodurilor și intervalelor dintre ele.

Dinamică: se folosește o matrice  $a[i,j] =$  costul minim pentru a forma un arbore de căutare cu nodurile de la  $i$  la  $j$ . Pentru aceasta se încearcă fixarea rădăcinii în punctul  $k$  (între  $i$  și  $j$ ), iar costul arborelui se determină pe baza lui  $a[i,k]$  și  $a[k+1,j]$ .

Se află matricea, și se afisează  $a[1,n]$ , dar  $O(n^3)$ .

Pentru a reduce complexitatea la  $O(n^2)$  trebuie să ne dăm seama că :

rădăcina arborelui de la  $i$  la  $j-1 \leq$  rădăcina arborelui de la  $i$  la  $j \leq$  rădăcina arborelui de la  $i+1$  la  $j$ .



**Mihai Stroe, Ginfo nr 13/7 - noiembrie 2003**

În prima fază a rezolvării, se citesc și se sortează valorile care vor fi plasate în nodurile arborelui.

Se citesc interogările și, folosind căutări binare, se determină de câte ori se caută fiecare din numerele date și de câte ori se caută numere din intervalul dintre două numere date, consecutive în șirul sortat (sau unul din intervalele exterioare arborelui). Se obține astfel un șir de ponderi.

În continuare, se folosește metoda programării dinamice. Se calculează o matrice  $A$ , unde  $a_{i,j}$  = costul minim pentru a forma un arbore de căutare cu nodurile de la  $i$  la  $j$  (unde  $i$  și  $j$  sunt poziții, deci indici pe șirul sortat de valori).

Pentru calculul unei astfel de valori se încearcă fixarea rădăcinii în fiecare poziție  $k$  (între  $i$  și  $j$  inclusiv). Costul arborelui se determină pe baza lui  $a_{i,k}$  și  $a_{k+1,j}$ ; determinarea relației exacte este lăsată ca exercițiu pentru cititor.

Valorile  $a_{i,j}$  sunt calculate în ordinea crescătoare a diferenței  $i - j$ ; matricea este deci completată pe diagonale. Rezultatul cerut se va afla în  $a_{1,N}$ .

Ordinul de complexitate al acestei rezolvări este  $O(N^3)$  și este dat de cele 3 cicluri for imbricate (pentru diferența între  $i$  și  $j$ , pentru  $i$  și pentru  $k$ ). Timpul de rezolvare pentru un set de date de intrare este rezonabil, dar fișierul de intrare poate conține un număr mare de astfel de seturi de date. În acest caz, rezolvarea nu se va încadra în timp.

Pentru a reduce complexitatea la  $O(N^2)$  putem face următoarea observație. Dacă notăm cu  $R_{i,j}$  rădăcina arborelui optim care ar conține pozițiile de la  $i$  la  $j$  din șirul sortat de valori, atunci:

$$R_{i,j-1} \leq R_{i,j} \leq R_{i+1,j}.$$

Folosind această observație, valorile de pe o anumită diagonală a matricei se pot calcula în timp liniar.

**Analiza complexității**

În continuare vom analiza complexitatea rezolvării pentru un singur set de date.

Operația de citire a datelor de intrare are ordinul de complexitate  $O(N + M)$ , iar scrierea rezultatului se realizează în timp constant.

Ordinul de complexitate al primei variante de construire a matricei este  $O(N^3)$ . Această variantă nu s-ar fi încadrat în timp pentru testele mari.

Folosind relația existentă între valorile din  $R$  și calculând elementele de pe fiecare diagonală a matricei în  $O(N)$ , ordinul de complexitate al construirii matricei este  $O(N^2)$ .

În concluzie, ordinul de complexitate al algoritmului de rezolvare pentru un singur set de date este

$$O(N + M) + O(1) + O(N^2) = O(N^2 + M).$$

Dacă luăm în considerare toate cele  $T$  teste, ordinul de complexitate devine  $O(T \cdot (N^2 + M))$ .

### 10.2.2 Rezolvare detaliată

### 10.2.3 Codul sursă

## 10.3 A007 - ONI 2003

Agentul **007** are de distrus o tabără de teroriști. Tabăra de teroriști este formată din mai multe obiective (depozite de muniție, pavilioane pentru teroriști, etc.), considerate punctiforme în plan. Agentul **007** primește de la serviciul de informații o hartă cu  $n$  obiective din tabăra teroriștilor, date prin coordonatele carteziene. Pe lângă hartă, agentul **007** mai primește și o armă specială (construită pentru această misiune). Arma primită are două țevi și permite tragerea simultană pe aceeași direcție (rectilie), dar în sens invers a două rachete cu aceeași viteză. După ce se trage cu arma, odată cu atingerea unei ținte explodează și cealaltă rachetă (chiar dacă aceasta din urmă nu și-a atins ținta).

#### Cerință

Agentul **007** vrea să distrugă tabăra cât mai repede și cu cât mai puține rachete, pentru acest lucru el studiază posibilitatea să se așeze într-un punct din tabără (diferit de obiective) care să permită trageri eficiente, adică la fiecare tragere să distrugă câte două obiective simultan.

Determinați dacă este posibil să se găsească un astfel de punct.

#### Date de intrare

În fișierul **a007.in** pe prima linie se află numărul de teste  $k$ , după care urmează date pentru fiecare test. Pentru fiecare test pe o linie se află  $n$ , iar pe următoarele  $n$  linii sunt coordonatele obiectivelor din tabăra teroriștilor (separate printr-un spațiu în ordinea abscisă ordonată).

#### Date de ieșire

În fișierul **a007.out** se vor scrie  $k$  linii, pe fiecare linie se va scrie 1, dacă există soluție și 0 dacă nu există soluție. În cazul în care există soluție se va scrie în continuare pe aceeași linie, separate printr-un spațiu, coordonatele punctului cerut (numere reale trunchiate la 4 zecimale, în ordinea abscisă ordonată).

#### Restricții

$$0 \leq n \leq 10000$$

$$1 \leq k \leq 3$$

coordonatele punctelor sunt întregi din intervalul  $[-10000, 10000]$

#### Observație

Un obiectiv este distrus dacă racheta explodează exact în punctul core-spunzător lui.

#### Exemplu

a007.in	a007.out
2	1 5.0000 5.0000
4	0
10 0	
10 10	
0 10	
0 0	
6	
0 0	
10 0	
2 10	
12 0	
5 0	
7 0	

**Timpan maxim de execuție/test:** 0.2 secunde (pentru Windows și Linux)

### 10.3.1 Indicații de rezolvare - descriere soluție \*

#### Soluția oficială

Se observă că problema se reduce la găsirea unui centru de simetrie într-o mulțime de  $n$  puncte, cu coordonate întregi.

Punctul de simetrie pentru mulțimea de puncte are proprietatea că simetricul fiecărui punct din mulțime față de centrul de simetrie este tot un punct din mulțimea de puncte. De aici obținem că pentru a exista centru de simetrie este necesar ca  $n$  să fie par.

Acum dacă  $n$  este par, observăm că centrul de simetrie este mijlocul a  $n/2$  segmente cu capete în aceste puncte și fiecare punct din mulțime este capăt pentru un singur segment.

Notăm cu  $x$  și  $y$  vectorii ce rețin coordonatele celor  $n$  puncte.

Ordonăm punctele crescător după abscise punctele, la abscise egale vom ordona crescător după ordonată.

Pentru un segment cu capetele de coordonate  $(x_1, y_1)$  și  $(x_2, y_2)$  mijlocul segmentului are coordonatele  $x=(x_1+x_2)/2$ ,  $y=(y_1+y_2)/2$ .

Dacă există centru de simetrie, atunci acesta este cu centrul în mijlocul segmentului cu capetele în primul punct și celălalt în ultimul punct (adică cu un capăt în cel mai din stânga punct și celălalt în cel mai din stânga). Notăm acest punct cu  $M$ .

Pentru ca să existe centru de simetrie trebuie ca segmentele cu coordonatele  $(x_i, y_i)$  și  $(x_{n-i+1}, y_{n-i+1})$  să aibă centrul în  $M$ ,  $i=1, 2, \dots, n \text{ div } 2$ .

Complexitate algoritm:  $O(n \log n)$ .

**Mihai Stroe, Ginfo nr 13/7 - noiembrie 2003**

Problema se reduce la găsirea unui centru de simetrie într-o mulțime de  $n$  puncte având coordonate întregi.

Punctul de simetrie pentru mulțimea de puncte are proprietatea că simetricul fiecărui punct din mulțime față de centrul de simetrie este tot un punct din mulțime. În concluzie, pentru a exista centru de simetrie trebuie ca  $n$  să fie par.

Dacă  $n$  este par, se observă că punctul căutat (centrul de simetrie) trebuie să fie mijlocul a  $n/2$  segmente cu capetele în punctele date. Fiecare punct din mulțime ar fi capăt pentru exact un segment. Dacă am putea identifica aceste segmente, problema s-ar reduce la determinarea mijlocului unuia dintre ele și la testarea dacă acesta este și mijlocul celorlalte.

Notăm cu  $x$  și  $y$  vectorii care rețin coordonatele celor  $n$  puncte.

Ordonăm punctele crescător în funcție de abscise. În cazul în care două abscise sunt egale vom ordona crescător în funcție de ordonată.

Pentru un segment cu capetele de coordonate  $(x_1, y_1)$  și  $(x_2, y_2)$  mijlocul segmentului are coordonatele  $x = (x_1 + x_2)/2$  și  $y = (y_1 + y_2)/2$ .

Dacă există centru de simetrie, atunci acesta este plasat în mijlocul segmentului având capetele în primul punct respectiv în ultimul punct (adică cu un capăt în cel mai din stânga punct și celălalt în cel mai din dreapta). Notăm acest punct cu  $M$ .

Pentru ca să existe centru de simetrie trebuie ca segmentele având coordonatele  $(x_i, y_i)$  și  $(x_{n-i+1}, y_{n-i+1})$  să aibă mijlocul în  $M$ , pentru orice  $i = 1, 2, \dots, n \text{ div } 2$ . Se verifică aceste condiții pentru punctul  $M$  determinat.

Fără a observa legătura dintre sortarea punctelor în ordinea precizată și relația de simetrie, putea fi încercată o abordare bazată pe determinarea repetată a înfășurătorii convexe. Această abordare are complexitatea  $O(N^2)$  pe cazul cel mai defavorabil, deci probabil că nu s-ar fi încadrat în timp.

**Analiza complexității**

Operația de citire a datelor de intrare are ordinul de complexitate  $O(N)$ .

Sortarea punctelor se va realiza cu unul din algoritmi clasici având ordinul de complexitate  $O(N \cdot \log N)$ .

Operațiile de determinare a mijlocului unuia din segmente și verificarea faptului că acest punct este și mijlocul celorlalte segmente au, în total, ordinul de complexitate  $O(N)$ .

Operația de scriere a rezultatelor are ordinul de complexitate  $O(1)$ .

În final, ordinul de complexitate al algoritmului de rezolvare a acestei probleme este  $O(N) + O(N \cdot \log N) + O(N) = O(N \cdot \log N)$ .

**10.3.2 Rezolvare detaliată**

### 10.3.3 Codul sursă

## 10.4 Inter - ONI 2003

În țara *Smar* sunt  $N$  autostrăzi, sub forma unor drepte în plan. Se știe că la intersecții de drumuri (care includ și autostrăzi) există un risc ridicat de accidente. De aceea polițiștii din această țară au hotărât stabilirea unei zone compacte care să includă toate intersecțiile și în care să se supravegheze atent circulația. Din motive financiare zona trebuie să fie de perimetru minim.

### Cerință

Scrieți un program care să determine aria zonei de supraveghere alese.

### Date de intrare

Din fișierul **inter.in** se va citi de pe prima linie numărul de autostrăzi, iar de pe fiecare dintre următoarele  $N$  linii câte patru numere reale, separate prin câte un spațiu, reprezentând coordonatele a două puncte distincte ce determină câte o dreaptă. Ele sunt date în ordinea  $X_1Y_1X_2Y_2$ , adică abscisa și ordonata punctului 1, apoi abscisa și ordonata punctului 2.

### Date de ieșire

În fișierul **inter.out** se va scrie pe prima linie un singur număr real, cu două zecimale exacte (cu trunchiere), reprezentând aria zonei alese pentru supraveghere.

### Restricții și precizări

Între oricare două autostrăzi există fix o intersecție.

Aria suprafeței de supraveghere este strict pozitivă pentru datele de test.

5 teste din 10 vor avea  $N < 501$ .

$2 < N < 5001$

### Exemplu

inter.in	inter.out
4	3.00
0 0 1 0	
0 0 0 2	
0 2 1 0	
-2 0 0 1	

**Timp maxim de execuție/test:** 0.2 secunde pentru Linux, 1.5 secunde pentru Windows.

### 10.4.1 Indicații de rezolvare - descriere soluție \*

#### Soluția oficială

Sunt  $N*(N-1)/2$  puncte de intersecție. Dar din acestea maxim  $2*N$  ne interesează (extremitățile de pe drepte), dar de fapt sunt maxim  $N$ . Pentru aflarea acestor puncte, care pot fi pe înfășurătoarea convexă a celor  $N*(N-1)/2$  puncte, se sortează

drepte după pantă. Punctele de interes sunt intersecția a două drepte consecutive în sortare, și în plus prima cu ultima.

Pe aceste  $N$  puncte se face înfășurătoarea convexă.

Se calculează aria zonei rezultate.

Complexitate :

1 - sortare după pantă -  $O(N \log N)$

2 - calcularea punctelor candidate de a fi pe înfășurătoare -  $O(N)$

3 - înfășurătoarea convexă -  $O(N \log N)$

4 - calcularea ariei -  $O(N)$

În total:  $O(N \log N)$

### Mihai Stroe, Ginfo nr 13/7 - noiembrie 2003

Sunt  $N \cdot (N - 1) / 2$  puncte de intersecție. Din aceste puncte ne interesează punctele situate în extremități pe fiecare dreaptă. Aceste puncte, cel mult  $N$ , sunt vârful înfășurătorii convexe a celor  $N \cdot (N - 1) / 2$  puncte.

Se sortează drepte după pantă. Se observă că punctele de interes sunt intersecțiile a câte două drepte consecutive în vectorul sortat, și în plus intersecția primei drepte cu ultima.

Se determină înfășurătoarea convexă a acestor  $N$  puncte, folosind un algoritm clasic, și se calculează aria zonei rezultate.

#### Analiza complexității

Operația de citire a datelor de intrare are ordinul de complexitate  $O(N)$ , iar cea de scriere a rezultatului are ordinul de complexitate  $O(1)$ .

Sortarea dreptelor în funcție de pantă are ordinul de complexitate  $O(N \cdot \log N)$ .

Determinarea punctelor de interes are ordinul de complexitate  $O(N)$ .

Determinarea înfășurătorii convexe a acestor puncte are ordinul de complexitate  $O(N \cdot \log N)$ .

Calcularea ariei poligonului obținut are ordinul de complexitate  $O(N)$ .

În final, ordinul de complexitate al rezolvării este  $O(N) + O(N \cdot \log N) + O(N) + O(N) + O(N) = O(N \cdot \log N)$ .

### 10.4.2 Rezolvare detaliată

### 10.4.3 Codul sursă

## 10.5 Număr - ONI 2003

Fie  $x$  un număr natural cu exact  $n$  cifre scris în baza 10.

#### Cerință

Scrieți un program care să determine cel mai mic număr natural strict mai mare decât  $x$ , care are aceleași cifre ca și numărul  $x$  și care este palindrom.

#### Date de intrare

Fișierul de intrare **nr.in** conține două linii. Pe prima linie este scris  $n$ , numărul de cifre ale numărului  $x$ . Pe cea de a doua linie sunt scrise cele  $n$  cifre ale lui  $x$ .

#### Date de ieșire

Fișierul de ieșire **nr.out** conține o singură linie pe care se află cel mai mic număr natural strict mai mare decât  $x$ , care are aceleași cifre ca și numărul  $x$  și care este palindrom. Dacă nu există soluție pe prima linie a fișierului de ieșire va fi scrisă valoarea 0.

#### Restricții

$$2 \leq n \leq 1000$$

Numim palindrom un număr care citit de la stânga la dreapta, cât și de la dreapta la stânga este același (de exemplu 1331, 12321, etc).

Prima cifra a unui număr trebuie să fie nenulă.

Prin aceleași cifre se înțelege că fiecare cifră de la 0 la 9 apare în rezultat de același număr de ori ca și în numărul  $x$ .

#### Exemple

nr.in	nr.out	nr.in	nr.out
5	0	5	20102
12022		12200	

**Timp maxim de execuție:** 0.1 secunde/test (atât sub Windows cât și sub Linux).

### 10.5.1 Indicații de rezolvare - descriere soluție \*

#### Soluția oficială

1. Determinăm numărul de apariții pentru fiecare cifră (0..9) în numărul dat ( $x$ ). Acest număr trebuie să fie par (cu excepția eventuală a unei singure cifre; în caz că o astfel de cifră există trebuie să fie plasată la mijlocul palindromului), altfel nu există soluție.

2. Determinăm numărul de apariții ale fiecărei cifre (0..9) în prima jumătate a numărului  $x$ , precum și numărul de apariții necesar (pentru numărul cerut). Dacă pentru o cifră numărul de apariții existent în prima jumătate a numărului dat este prea mare, eliminăm aparițiile suplimentare, începând de la dreapta spre stânga.

3. Încerc să distribuim cifrele care mai sunt necesare în prima jumătate a numărului astfel încât să obținem cel mai mic număr posibil  $\geq x$  (cu o eventuală rearanjare a cifrelor existente). Distribuim cifrele de face căutând de la stânga la dreapta să completăm pozițiile cu cifre egale cu cele din numărul dat. Când nu mai este posibil:

- am obținut o jumătate mai mică sau egală cu cea din numărul dat (cea mai mare cu această proprietate, determinăm anagrama imediat următoare din punct de vedere lexicografic lui  $x$  - dacă există);

- am gasit o cea mai mică cifră mai mare decât cea de pe poziția curentă, o plasez, ordonez crescător cifrele de după  $\Rightarrow$  cea mai mică jumătate  $> x$ .

### Mihai Stroe, Ginfo nr 13/7 - noiembrie 2003

Se determină numărul de apariții pentru fiecare cifră având valori în intervalul  $[0, 9]$  în numărul dat  $x$ . Acest număr trebuie să fie par (cu excepția eventuală a unei singure cifre; în caz că o astfel de cifră există, trebuie să fie plasată la mijlocul palindromului), altfel nu există soluție.

În continuare se va construi prima jumătate a palindromului; cealaltă jumătate va fi obținută prin oglindirea primei jumătăți.

Se determină numărul de apariții ale fiecărei cifre (0...9) în prima jumătate a numărului  $x$ , precum și numărul de apariții necesar (pentru numărul cerut). Dacă pentru o cifră numărul de apariții existent în prima jumătate a numărului dat este prea mare, se elimină aparițiile suplimentare, începând de la dreapta spre stânga.

Se încearcă să se distribuie cifrele care mai sunt necesare în prima jumătate a numărului astfel încât să se obțină cel mai mic număr posibil mai mare sau egal cu  $x$  (cu o eventuală rearanjare a cifrelor existente). Distribuirea cifrelor se face căutând de la stânga la dreapta pozițiile cu cifre egale și completarea numărului cu cele din numărul dat.

Când acest lucru nu mai este posibil, ne aflăm în unul din următoarele două cazuri:

- am obținut o jumătate mai mică sau egală cu cea din numărul dat (cea mai mare cu această proprietate). Determinăm anagrama imediat următoare din punct de vedere lexicografic a primei jumătăți a lui  $x$ ; dacă o astfel de anagramă există, ea (împreună cu cea de-a doua jumătate a palindromului) dă soluția problemei.

- am găsit-o pe cea mai mică cifră mai mare decât cea de pe poziția curentă. Plasăm cifra respectivă, ordonăm crescător cifrele rămase, construim cea de-a doua jumătate și obținem soluția problemei.

#### Analiza complexității

Ordinul de complexitate al operațiilor de citire a datelor de intrare și furnizare a rezultatelor este  $O(N)$ .

Determinarea numărului de apariții al fiecărei cifre în numărul  $x$  și în numărul dorit au ordinul de complexitate  $O(N)$ .

Construirea pas cu pas a primei jumătăți a palindromului prin adăugarea unei cifre cât timp este posibil are ordinul de complexitate  $O(N)$ .

Fiecare din cele două cazuri care încheie rezolvarea se tratează cu algoritmi de complexitate  $O(N)$ .

În concluzie, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este  $O(N) + O(N) + O(N) + O(N) + O(N) = O(N)$ .

### 10.5.2 Rezolvare detaliată



### 10.5.3 Codul sursă

## 10.6 Proc - ONI 2003

O aplicație ce trebuie executată pe un calculator multi-procesor constă din  $N$  fragmente de cod independente, ce pot fi rulate în paralel. Fiecare fragment trebuie executat în totalitate pe un singur procesor. Din dorința de a paraleliza cât mai mult aplicația, fragmentele de cod au dimensiuni mici și, în consecință, timpi de execuție mici. Mai precis, execuția fiecărui fragment durează UNUL sau DOUĂ cicluri de ceas pe un procesor de tipul Pentium IV.

Sistemul pe care urmează să fie executată aplicația constă din  $P$  procesoare. Spre deosebire de majoritatea sistemelor de acest fel, însă, cele  $P$  procesoare au viteze de execuție diferite. Primul procesor este un Pentium IV și este cel mai rapid. Al doilea procesor este de două ori mai lent decât primul, al treilea de trei ori mai lent ... al  $i$ -lea procesor este de  $i$  ori mai încet decât primul. În aceste condiții, timpul de execuție al fiecărui fragment de cod diferă, în funcție de procesorul pe care va fi executat. Să presupunem că un segment de cod are timpul de execuție  $T$  (unde  $T$  este 1 sau 2) pe primul procesor. Atunci pe un procesor  $i$ , timpul său de execuție va fi  $i * T$ .

#### Cerință

Știind că fragmentele de cod pot fi executate în orice ordine și pe orice procesor și că orice procesor poate executa, la un moment dat, un singur fragment de cod, determinați timpul minim după care se va termina execuția aplicației (adică a tuturor fragmentelor de cod). Timpul după care se termină aplicația este egal cu maximumul dintre timpii după care fiecare procesor redevine disponibil. Timpul după care un procesor redevine disponibil este egal cu suma timpilor de execuție a fragmentelor de cod rulate pe procesorul respectiv.

#### Date de intrare

Prima (și singura) linie a fișierului de intrare **proc.in** conține trei numere întregi, separate prin spații:  $N$  - numărul de fragmente de cod,  $K$  - numărul de fragmente de cod care au timpul de execuție pe un Pentium IV egal cu 1 (implicit,  $N-K$  au timpul de execuție egal cu 2) și  $P$  - numărul de procesoare ale sistemului.

#### Date de ieșire

Fișierul **proc.out** va conține o singură linie pe care se află timpul minim după care se termină de executat aplicația.

#### Restricții și precizări

$$0 \leq K \leq N \leq 1000000000$$

$$1 \leq P \leq 65535$$

Cel puțin 40% din teste vor avea  $N \leq 2000$  și  $P \leq 2000$ .

Cel puțin 70% din teste vor avea  $N \leq 65535$  și  $P \leq 16383$ .

#### Exemplu

proc.in	proc.out
4 3 2	4

**Explicație**

Pe primul procesor se execută un fragment de cod cu timpul de execuție (calculat pe un Pentium IV) egal cu 1 și un fragment de cod cu timpul de execuție egal cu 2  $\Rightarrow$  timpul după care acest procesor devine disponibil este  $1 * 1 + 1 * 2 = 3$ . Pe al doilea procesor se execută două fragmente de cod cu timpul de execuție (calculat pe un Pentium IV) egal cu 1  $\Rightarrow$  timpul după care acest procesor devine disponibil este 2 [numărul de fragmente] \* (2\*1) [timpul de execuție al fiecărui fragment pe procesorul 2] = 4.

**Timp maxim de execuție:** 0.2 secunde/test pentru Linux și 1.5 secunde/test pentru Windows.

**10.6.1 Indicații de rezolvare - descriere soluție \*****Soluția oficială**

Testele au fost structurate astfel încât diverse abordări ale problemei să obțină un număr diferit de puncte, astfel:

- 20% din punctaj, pentru o soluție cu complexitate exponențială
- 40% din punctaj pentru o soluție greedy cu complexitatea  $O(N * P)$
- 70% din punctaj pentru o soluție greedy cu complexitatea  $O(N * \log P)$
- 100% din punctaj pentru o soluție cu complexitatea  $O(P * \log N)$

**Soluția exponențială:**

Există mai multe posibilități de a implementa o soluție care folosește **backtracking**. Se poate varia pentru fiecare fragment de cod procesorul pe care va fi executat -  $O(NP)$  - sau se poate stabili pentru fiecare procesor câte fragmente de cod de tipul 1, respectiv 2 vor fi executate pe el. O optimizare a celei de-a doua variante ar fi că timpul după care un procesor  $i$  devine disponibil, notat prin  $T_i$ , trebuie să respecte relația  $T_i/i = T_j/j$ , unde  $j$  este numărul unui procesor mai rapid decât  $i$  (adică  $j > i$ ).

**Soluția greedy cu complexitate  $O(N * P)$** 

Se inițializează timpii după care fiecare procesor redevine disponibil cu 0. Se stabilește procesorul pe care va fi executat fiecare fragment de cod cu timpul 2. Apoi se stabilește același lucru (adică procesorul pe care va fi executat) pentru fiecare fragment de cod cu timpul 1. Pentru a stabili procesorul pe care va fi executat un fragment de cod, se încearcă executarea sa pe orice procesor de la 1 la  $P$  și se alege acel procesor pentru care se găsește minimumul dintre timpii după care procesoarele redevin disponibile. Altfel spus, să presupunem că până la adăugarea fragmentului de cod curent, timpul de revenire al fiecărui procesor  $i$  este  $T_i$  și să presupunem că fragmentul de cod actual are timpul de execuție  $Q$  ( $Q$  este 1 sau 2). Atunci se alege procesorul  $i$ , pentru care  $T_i + Q * i$  este minim.

**Soluția greedy cu complexitate  $O(N * \log P)$**

Ideea acestei solutii este la fel ca cea descrisă mai sus, numai că determinarea procesorului pentru care se obține timpul de revenire minim se face logaritmice, folosind o structura de **heap**.

**Soluția (comisiei) cu complexitatea  $O(P * \log N)$**

Se observă că timpul minim după care se va termina aplicația este între 0 și  $2 \cdot N$  (în cazul când toate fragmentele de cod au timpul 2 și sunt executate pe primul procesor). Se va folosi o căutare binară pentru determinarea acestui timp. La un moment dat, în cadrul căutării, se va testa dacă timpul minim este mai mic sau egal decât timpul  $T$ . Dacă da, atunci se încearcă un  $T$  mai mic. Altfel, se mărește  $T$ -ul. Funcția care decide dacă timpul minim este mai mic sau egal cu  $T$  are complexitatea  $O(P)$ . Se observă că dacă timpul minim este cel mult  $T$ , atunci suma timpilor elementari pe fiecare procesor  $k$  trebuie să fie cel mult  $\lceil T/k \rceil$ . Se calculează

$$N_2 = \sum_{k=1}^P \left\lceil \frac{\lceil T/k \rceil}{2} \right\rceil,$$

având semnificația: numărul maxim de fragmente de cod cu timpul 2 ce pot fi executate dacă timpul minim este cel mult  $T$ . Dacă  $N_2$  este mai mare sau egal cu numărul de fragmente de cod cu timpul 2 existente (să notăm acest număr cu DOI), se verifică dacă se pot executa destule fragmente de cod cu timpul 1. Pentru aceasta se calculează

$$N_1 = \sum_{k=1}^P \left( \left\lceil \frac{T}{k} \right\rceil \bmod 2 \right),$$

Dacă  $N_1 + 2 \cdot (N_2 - \text{DOI})$  este mai mare sau egal cu numărul de fragmente de cod cu timpul 1, atunci timpul minim după care se termină aplicația este mai mic sau egal cu  $T$ . Altfel, nu.

### Mihai Stroe, Ginfo nr 13/7 - noiembrie 2003

Se observă că timpul minim după care se va termina aplicația este între 0 și  $2 \cdot N$  (în cazul când toate fragmentele de cod au timpul 2 și sunt executate pe primul procesor).

De asemenea, se observă că, fixând o limită de timp  $T$ , se poate determina dacă putem executa toate fragmentele de cod în timpul limită acordat, folosind procesoarele existente.

Ne interesează limita minimă de timp pentru care această condiție este respectată.

Se va folosi o căutare binară pentru determinarea limitei de timp. La un moment dat, în cadrul căutării, se va testa dacă putem executa toate fragmentele de cod. Dacă da, atunci se încearcă un  $T$  mai mic. Altfel, se mărește  $T$ -ul.

Funcția care decide dacă putem executa toate fragmentele de cod în timpul  $T$  are complexitatea  $O(P)$ , unde  $P$  este numărul de procesoare disponibile. Se observă că fiecare procesor  $K$  asigură  $\lceil T/K \rceil$  unități de timp elementare.

Se calculează  $N_2 = \sum_{k=1}^P \lceil [T/k]/2 \rceil$ , având semnificația: numărul maxim de fragmente de cod cu timpul 2 care pot fi executate dacă timpul minim este cel mult  $T$ .

Dacă  $N_2$  este mai mare sau egal cu numărul de fragmente de cod cu timpul 2 existente (să notăm acest număr cu DOI), se verifică dacă se pot executa destule fragmente de cod cu timpul 1. Pentru aceasta se calculează  $N_1 = \sum_{k=1}^P \lceil [T/k] \bmod 2 \rceil$

Dacă  $N_1 + 2 \cdot (N_2 - DOI)$  este mai mare sau egal cu numărul de fragmente de cod cu timpul 1, atunci putem executa toate fragmentele de cod în timpul  $T$ .

#### **Analiza complexității**

Operațiile de citire a datelor și afișarea rezultatelor au ordinul de complexitate  $O(1)$ .

Sunt  $\log N$  pași de căutare binară, deoarece valoarea căutată este limitată superior de  $2 \cdot N$ .

În cadrul fiecărui pas se execută o funcție care are ordinul de complexitate  $O(P)$ .

În concluzie, ordinul de complexitate al algoritmului de rezolvare pentru această problemă este  $O(1) + O(\log N) \cdot O(P) = O(P \cdot \log N)$ .

### **10.6.2 Rezolvare detaliată**

#### **10.6.3 Codul sursă**

# Capitolul 11

## ONI 2004 clasa a XI-a



### 11.1 BASE3 - ONI 2004

Se dau 3 numere scrise în baza 3 (folosind cifrele 0, 1 și 2). Se dorește găsirea unui număr  $N$  în baza 3, care să aibă un număr impar de cifre, iar cifra de pe poziția din mijloc să aibă valoarea 1. Acest număr  $N$  trebuie obținut prin concatenarea celor trei numere date; în această concatenare, fiecare din cele 3 numere poate fi folosit de zero sau mai multe ori.

#### **Cerință**

Determinați numărul minim de cifre pe care îl poate avea un număr având proprietățile precizate mai sus.

#### **Date de intrare**

Fișierul de intrare **base3.in** conține 3 linii. Pe fiecare linie se află scris un număr în baza 3.

#### **Date de ieșire**

Fișierul de ieșire **base3.out** va conține numărul minim de cifre pe care îl poate avea un număr  $N$  cu proprietățile specificate. Dacă nu se poate obține nici un astfel de număr, afișați în fișier valoarea 0.

#### **Restricții și precizări**

Numărul de cifre al fiecăruia din cele 3 numere este un număr întreg între 1 și 16000.

Numerele date pot conține zerouri la început; acestea trebuie luate în considerare, dacă numărul respectiv este folosit în concatenare.

**Exemplu**

base3.in	base3.out
001	13
020	
2020	

**Explicație:**

Se poate obține numărul 2020001001001.

**Timp maxim de executare/test:** 0.4 secunde pentru Linux și 0.8 secunde pentru Windows

### 11.1.1 Indicații de rezolvare - descriere soluție \*

**Soluția oficială - Mugurel Ionuț Andreica**

Se calculează matricea  $MIN[i, x, j]$ , cu  $i$  între 1 și 3,  $j$  între 1 și 2, iar  $x$  între 0 și lungimea numărului  $i$ , având următoarea semnificație:

- dacă  $j = 1$ ,  $MIN[i, x, j]$  reprezintă lungimea celui mai scurt număr care are la mijloc primele  $x$  cifre din al  $i$ -lea număr
- dacă  $j = 2$ ,  $MIN[i, x, j]$  reprezintă lungimea celui mai scurt număr care are la mijloc ultimele  $x$  cifre din al  $i$ -lea număr

Calculul acestor valori corespunde unei determinări a numărului din exterior spre centru. Cei 3 indici ai matricei codifică o stare, iar trecerea de la o stare la alta se realizează prin concatenarea unuia din numere în partea stângă sau dreaptă. Astfel, se poate folosi un algoritm de drum minim (de exemplu, Dijkstra cu heap-uri). Conform acestei codificări, numărul cerut corespunde unui drum minim în graful stărilor, iar lungimea acestui drum este limitată superior de  $6 * 16000^2$  (dar, în practică, este mai mică).

Complexitatea algoritmului este  $O((L_1 + L_2 + L_3) * \log(L_1 + L_2 + L_3))$ .

### 11.1.2 Rezolvare detaliată

### 11.1.3 Codul sursă

## 11.2 Coach - ONI 2004

Sunteți antrenorul ciclistului Adirem Onamihs. În curând va avea loc un eveniment sportiv, iar pentru organizarea acestuia s-au amenajat  $N$  intersecții și  $M$  drumuri bidirecționale între aceste intersecții. Pentru fiecare drum se cunoaște numărul de minute necesare pentru parcurgerea lui. La fiecare intersecție ciclistul care trece pe acolo este obligat să servească o băutură energizantă și răcoritoare. Băutura diferă de la intersecție la intersecție și se cunoaște deja numărul de calorii ale fiecărei băuturi.

Ca mare antrenor, urmează să întocmiți un plan special pentru a-l antrena pe Adirem. Doriți ca durata traseului pe care îl alege Adirem să aibă exact  $T$  minute, însă nu vreți să-i plănuți întregul traseu (Adirem trebuie să își antreneze și mintea, nu numai corpul). Îi veți preciza lui Adirem intersecția de unde își începe traseul și intersecția unde îl termină. Adirem învață repede - el știe întotdeauna să aleagă traseul optim (drumul cel mai scurt dintre cele două intersecții). Pentru a-l face să meargă exact  $T$  minute îi veți interzice trecerea prin anumite intersecții, sub pretextul că valoarea calorică a băuturii servite în intersecția respectivă nu este benefică pentru antrenamentul lui. Astfel, îi veți preciza o limită inferioară și una superioară pentru numărul de calorii ale băuturilor pe care el are voie să le bea. Adirem nu va trece decât prin intersecțiile unde se servește o băutură cu valoare calorică între limitele date.

### Cerință

Scrieți un program care să calculeze cele patru variabile din antrenamentul lui Adirem: intersecția de start, intersecția de finish, valoarea calorică minimă pe care poate să o consume și valoarea calorică maximă, astfel încât drumul cel mai scurt dintre cele două intersecții (care să respecte restricțiile) să dureze exact  $T$  minute.

### Date de intrare

Prima linie a fișierului **coach.in** conține trei numere întregi  $N$ ,  $M$  și  $T$  - numărul de intersecții, numărul de drumuri, respectiv timpul dorit. Următoarele  $N$  linii conțin câte un număr - valorile calorice (întregi între 1 și 10000 inclusiv) ale băuturilor din intersecții, în ordine (de la 1 la  $N$ ). Următoarele  $M$  linii conțin câte un triplet de numere: două intersecții (numere distincte între 1 și  $N$ ) și durata drumului dintre ele (întreg între 1 și 10000 inclusiv).

### Date de ieșire

Fișierul **coach.out** va conține o linie pe care se vor afla cele patru valori găsite: nodul de start, nodul de finish, valoarea calorică minimă și valoarea calorică maximă. Nodurile vor fi întregi între 1 și  $N$ , iar valorile calorice vor fi întregi între 1 și 10000 (inclusiv).

### Restricții și precizări

$$1 \leq N \leq 100,$$

$$1 \leq M \leq 4950,$$

$$1 \leq T \leq 1000000$$

Intersecțiile găsite (de start și de finish) trebuie să respecte și ele restricțiile calorice

O băutură cu valoare calorică  $x$  poate fi băută dacă și numai dacă  $cm_{in} \leq x \leq cm_{ax}$ , unde  $cm_{in}$  și  $cm_{ax}$  sunt valorile calorice minime și maxime stabilite de antrenor

Între două intersecții există maximum un drum

Valorile calorice sunt distincte

Există întotdeauna soluție; dacă există mai multe soluții se cere oricare dintre ele

### Exemplu

coach.in	coach.out
6 9 11	3 6 20 55
40	
10	
20	
30	
60	
50	
1 2 2	
1 3 2	
1 4 4	
1 6 10	
2 3 3	
2 4 1	
4 5 1	
4 6 5	
5 6 2	

**Timp maxim de executare/test:** 0.5 secunde pentru Linux și 0.9 secunde pentru Windows

## 11.2.1 Indicații de rezolvare - descriere soluție \*

### Soluția oficială - Radu Berinde

Se sortează nodurile după valoarea asociată și se renumerează (nodul 1 are valoarea cea mai mică).

Se fixează un nod de start  $s$  (și deci un  $cm_{in}$ ) și se aplică algoritmul *Roy – Floyd*, pentru nodurile din graf care au valoarea asociată mai mare sau egală cu  $cm_{in}$  (noduri de la  $s$  la  $N$ ). Proprietatea algoritmului *Roy – Floyd* aplicat de la nodul  $s$  este următoarea: la pasul  $k$  (cu  $k$  de la  $s$  la  $N$ ),  $A[i][j]$  este drumul cel mai scurt de la  $i$  la  $j$  care poate trece prin noduri de la  $s$  la  $k$ .

Datorită acestei proprietăți, complexitatea totală este  $O(N^4)$ , nefiind necesară și fixarea lui  $cm_{ax}$  (care ar fi condus la o complexitate  $O(N^5)$  și obținerea a 40 puncte), înainte aplicării algoritmului.



### 11.2.2 Rezolvare detaliată

### 11.2.3 Codul sursă

## 11.3 Color - ONI 2004

Ion și Vasile joacă un joc. Ei au la dispoziție un arbore binar strict (adică fiecare nod are 0 sau 2 fii) cu  $N$  noduri, numerotate de la 1 la  $N$  (nodul numerotat cu 1 este rădăcina arborelui). Inițial, toate nodurile sunt colorate în alb. Jucătorii vor efectua mutări alternativ, iar jucătorul aflat la mutare va colora în negru un nod colorat în alb. Ion efectuează prima mutare și poate colora în negru un singur nod (oricare) al arborelui. Considerând că ultimul nod colorat de unul dintre jucători este  $P$ , jucătorul care urmează la mutare poate colora în negru unul din următoarele noduri (dacă nu au fost deja colorate în negru):

- unul din cei 2 fii ai lui  $P$  (dacă  $P$  nu este frunză în arbore)
- tatăl lui  $P$  (dacă  $P$  nu este rădăcina arborelui)

Jocul continuă până când unul dintre jucători nu mai poate efectua nici o mutare. Atunci, jucătorul care a efectuat ultima mutare este considerat câștigător. Nodul ales de Ion la prima mutare este numit nod câștigător dacă Ion poate câștiga jocul, indiferent de mutările lui Vasile (adică Ion are strategie sigură de câștig).

#### Cerință

Considerând că ambii jucători joacă optim, determinați toate nodurile din arbore pe care le poate colora Ion la prima mutare, astfel încât să fie sigur de victorie (nodurile câștigătoare).

#### Date de intrare

Prima linie a fișierului **color.in** conține numărul întreg  $N$ , reprezentând numărul de noduri din arbore. Următoarele  $N - 1$  linii conțin câte două numere întregi separate printr-un spațiu,  $a$  și  $b$ , având semnificația că  $a$  este tatăl lui  $b$ .

#### Date de ieșire

Pe prima linie a fișierului **color.out** veți afișa numărul întreg  $M$ , reprezentând numărul de noduri câștigătoare. Pe următoarea linie veți afișa numerele acestor noduri, în ordine crescătoare.

#### Restricții și precizări

$1 \leq N \leq 16000$ ,  $N$  impar

40% din teste vor avea  $N \leq 1000$

#### Exemplu

color.in	color.out
9	6
1 2	1 5 6 7 8 9
1 3	
2 4	
2 5	
4 6	
4 7	
3 8	
3 9	

**Timp maxim de executare/test:** 0.2 secunde pentru Linux și 0.3 secunde pentru Windows

### 11.3.1 Indicații de rezolvare - descriere soluție \*

#### Soluția oficială - Mugurel Ionuț Andreica

Pentru fiecare nod din arbore se calculează două valori :  $WINJOS[i] = 1$ , dacă jucătorul care începe are strategie sigură de câștig, în cazul în care el colorează întâi nodul  $i$ , iar al doilea jucător colorează, în continuare, unul din fiii lui  $i$  (și 0 în caz contrar) , respectiv  $WINSUS[i] = 1$ , dacă jucătorul care începe are strategie sigură de câștig, în cazul în care el colorează întâi nodul  $i$ , iar al doilea jucător colorează, în continuare, tatăl lui  $i$ .  $WINJOS[i]$  se calculează pe baza valorilor fiilor lui  $i$ , iar  $WINSUS[i]$ , pe baza lui  $WINSUS[tata[i]]$  și  $WINJOS[frate[i]]$ , unde  $frate[i]$  este nodul care are același tată ca și nodul  $i$ . Ambele valori se calculează în timp liniar.

### 11.3.2 Rezolvare detaliată

### 11.3.3 Codul sursă

## 11.4 Magic - ONI 2004

Misopan și Trofonaced sunt doi eroi care vor să-și unească forțele în lupta împotriva răului. Regatul este reprezentat printr-o matrice dreptunghiulară de  $N$  linii și  $M$  coloane. Fiecare element al matricei corespunde unei bucăți de teren uscat sau mlăștinos. Cei doi eroi nu se vor aventura în părțile mlăștinoase ale regatului - se vor deplasa numai pe uscat. Ei se pot muta dintr-o poziție a matricei în una din cele 4 poziții vecine pe orizontală sau pe verticală, dacă această poziție corespunde unei zone de uscat. Unele poziții de uscat pot fi transformate prin vrajă în mlăștină.

**Cerință**

Ajutați un vrăjitor malefic să aleagă un număr minim de poziții "transformabile", prin schimbarea cărora cei doi eroi să nu se poată întâlni (să nu existe un drum pe uscat între cei doi).

**Date de intrare**

Prima linie a fișierului **magic.in** conține două numere întregi  $N$  și  $M$  reprezentând numărul de linii, respectiv de coloane ale matricei. Următoarele  $N$  linii conțin câte  $M$  caractere cu următoarea semnificație:

- . pentru o poziție uscată
- x (mic) pentru una mlăștinoasă
- \* pentru una uscată "transformabilă" în una mlăștinoasă de către vrăjitor
- M pentru poziția eroului Misopan
- T pentru poziția eroului Trofonaced

**Date de ieșire**

Pe prima linie a fișierului **magic.out** se scrie numărul întreg  $R$ , reprezentând numărul minim de poziții care trebuie transformate. Pe următoarele  $R$  linii vor apărea câte 2 numere, reprezentând pozițiile alese. Primul număr va fi linia (între 1 și  $N$ ), iar al doilea va fi coloana (între 1 și  $M$ ).

**Restricții și precizări**

$1 \leq N, M \leq 50$

$R$  (rezultatul afișat) poate fi 0

Pe testele date va exista întotdeauna soluție

Se garantează că în toată matricea caracterele  $M$ , respectiv  $T$  vor apărea fiecare exact o dată

Pozițiile eroilor sunt implicit zone de uscat care nu pot fi transformate de vrăjitor

**Exemplu**

magic.in	magic.out
4 4	1
MxxT	3 3
. x * .	
. * * .	
* * x .	

**Timp maxim de executare/test:** 0.6 secunde pentru Linux și 1.8 secunde pentru Windows

**11.4.1 Indicații de rezolvare - descriere soluție \*****Soluția oficială - Radu Berinde**

Matricei date i se poate asocia un graf, construit astfel:

- pozițiilor uscate li se vor asocia câte un nod

- pozițiilor transformabile li se vor asocia câte două noduri (un nod dedublat) - unul pentru muchiile care "intră" în poziția respectivă, iar celălalt pentru cele care "ies" din poziție; se va pune o muchie de capacitate 1 între aceste două noduri.

- se vor pune muchii cu capacitate infinită între două noduri vecine în matrice (fiecare poziție are maxim 4 vecini)

Este destul de evident că problema inițială este echivalentă cu aflarea unei tăieturi minime în graful astfel creat. Tăietura minimă se determină folosind un algoritm de flux maxim.

Complexitate:  $O(N^4)$ . De observat că un algoritm care creează explicit graful folosește destul de multă memorie și ar putea să nu intre în timp pe toate testele. Este de preferat o implementare unde graful este implicit, fluxurile fiind ținute într-o matrice (câte 4 fluxuri pentru fiecare poziție + un flux intern pentru pozițiile dedublate).

### 11.4.2 Rezolvare detaliată

### 11.4.3 Codul sursă

## 11.5 Pătrate - ONI 2004

Ovi este un băiețel foarte isteț căruia îi place să scrie pe asfalt cu creta și să țopăie. El desenează cu cretă roșie un dreptunghi de lățime exact 2 metri și lungime  $N$  metri, pe care îl împarte în pătrate egale de latură 1 metru, unele laturi interioare fiind desenate cu cretă roșie, iar restul laturilor interioare cu cretă albă. Ovi pornește din pătratul aflat în colțul stânga sus al dreptunghiului, sărind dintr-un pătrat în altul vecin pe linie sau coloană, cu condiția ca latura care desparte cele două pătrate să nu fie colorată în roșu. El își dorește ca prin sărituri succesive să ajungă în toate pătratele dreptunghiului, dar a observat că numai pentru anumite variante de colorare a laturilor pătratelor reușește acest lucru.

În exemplele de mai jos (cu  $N = 2$ ) liniile interioare îngroșate sunt colorate cu roșu, iar cele punctate sunt colorate cu alb. La exemplul din fig. 1, pornind din colțul stânga sus se poate ajunge în oricare alt pătrat, dar în exemplul din fig. 2 nu se poate ajunge la pătratele din partea dreaptă.

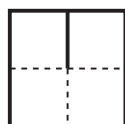


fig. 1

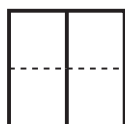


fig. 2

**Cerință**

Ajutați-l pe Ovi să numere câte posibilități de colorare în roșu a unor laturi interioare ale pătratelor sunt astfel încât plecând din colțul stânga sus să poată ajunge prin sărituri în oricare alt pătrat.

**Date de intrare**

În fișierul **patrate.in** se află un singur număr natural  $N$  ce reprezintă lungimea în metri a dreptunghiului.

**Date de ieșire**

În fișierul **patrate.out** veți scrie un singur număr natural (urmat de caracterul de sfârșit de linie) ce reprezintă numărul de posibilități cerut.

**Restricții și precizări**

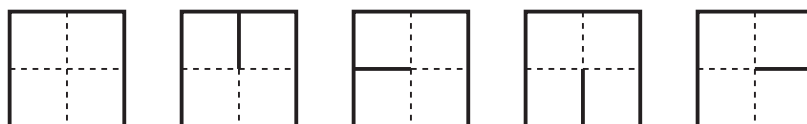
$$2 \leq N \leq 1000$$

**Exemplu**

patrate.in	patrate.out
2	5

**Explicație:**

Cele 5 posibilități sunt:



**Tim maxim de executare/test:** 0.2 secunde pentru Linux și 0.4 secunde pentru Windows

**11.5.1 Indicații de rezolvare - descriere soluție \*****Soluția oficială - Dan Pracsiu**

Să notăm numărul căutat cu  $X_N$  (numărul posibilităților pentru matricea  $2 \times N$ ). Relația de recurență este:

$$X_N = 5 * X_{N-1} - 2 * X_{N-2}$$

Este vorba de o relație de conexitate în matrice. La trecerea de la o matrice cu  $N$  linii la una cu  $N + 1$  linii trebuie avut grijă la faptul că dintr-o matrice de lungime  $N$  neconexa se poate obține o matrice conexă (dacă la ultima coloană nu așez nici o linie roșie).

Pentru  $n \leq 15$  se obține un rezultat care se încadrează în *long*. În rest se lucrează cu numere mari. Era suficientă, pentru încadrarea în timp, doar implementarea operației de adunare.

Aplicarea unui algoritm backtracking ar fi obținut 20 sau 30 puncte.

Complexitatea:  $O(n^2)$

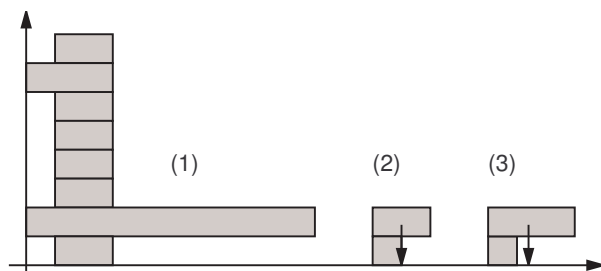
### 11.5.2 Rezolvare detaliată

### 11.5.3 Codul sursă

## 11.6 Turnuri - ONI 2004

Renumitul arhitect Prăbușilă dorește să construiască unul din cele mai interesante turnuri de pe planetă. Acest turn, în mod cu totul deosebit, va avea etaje de diverse lățimi, între 1 și 100, numere întregi.

Prăbușilă s-a hotărât deja ce dimensiune va avea fiecare din etajele turnului, dar nu și cum să le așeze pe orizontală. El ar dori mai întâi să știe câte variante are.



Etajele pot fi așezate la coordonate întregi și va trebui ca un astfel de turn să nu se dărâme.

Condiția pentru ca un turn să fie stabil este ca la fiecare etaj perpendiculara coborâtă din centrul de greutate al grupului etajelor superioare să cadă strict în interiorul aceluia etaj (nu are voie să fie pe margini sau în afară - de ex. turnurile 2 și 3 sunt instabile).

Centrul de greutate al unui etaj se află la mijlocul etajului respectiv.

Centrul de greutate al unui grup de etaje are drept coordonată  $x$  (orizontală) media coordonatelor  $x$  ale centrelor de greutate ale etajelor componente. (Etajele au mase egale, indiferent de cât de late sunt).

În exemplul 1, etajul din vârf are coordonata  $x$  a centrului de greutate 2, iar grupul celor 2 etaje din vârf are centrul de greutate la coordonata  $x = 1.75$  (media aritmetică între 2 și 1.5)

Se observă în figura 1 că, deși perpendiculara din centrul de greutate al etajului 2 cade în afara etajului 1, totuși turnul este stabil, deoarece perpendiculara din centrul de greutate al grupului format din etajele 2 – 8 cade strict în interiorul etajului 1.

**Cerință**

Să se afle câte turnuri stabile există.

#### Date de intrare

Fișierul de intrare **turnuri.in** conține pe o singură linie lista de numere naturale separate prin câte un spațiu, numere reprezentând lățimile etajelor turnului, începând cu cel mai de sus. Lista se termină cu un 0.

#### Date de ieșire

Fișierul de ieșire **turnuri.out** conține numărul de turnuri.

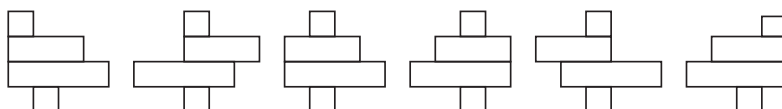
#### Restricții și precizări:

- numărul maxim de turnuri nu va depăși 2 miliarde
- numărul maxim de etaje ale unui turn este 200
- lățimea maximă a unui etaj este 100

#### Exemplu:

turnuri.in	turnuri.out
1 3 4 1 0	6

Cele 6 variante sunt:



**Timp maxim de executare/test:** 0.2 secunde pentru Linux și 0.2 secunde pentru Windows

### 11.6.1 Indicații de rezolvare - descriere soluție \*

#### Soluția oficială - Radu Andrei Ștefan

Programare dinamică. Construim soluția de sus în jos. Pentru fiecare nivel "n", se observă că partea fracționară a coordonatei centrului de greutate poate fi  $k/(2 \cdot \text{numărul de etaje din care facem media} = n)$   $k \in [0, 2n - 1]$ . Vom reține într-un vector câte etaje există pentru fiecare fracție. Astfel, pentru etajul cel mai de sus: numărul de turnuri formate numai din acest etaj cu centrul de greutate cu partea fracționară  $1/2$  este 1 iar numărul de turnuri cu centrul de greutate cu parte fracționară 0 este 0 (dacă etajul are lungime impară).

Construim posibilitățile de la nivelul  $n+1$  prin sumarea elementelor de la nivelul  $n$  (etajele sunt numărate de sus în jos), pentru toate pozițiile pe orizontală etajului  $n+1$ . Atenție, fracțiile se schimbă de la un etaj la altul din  $k/(2n)$  în  $k/(2n+2)$ .

Complexitate:  $O(\text{înălțime} \cdot \text{suma lungimilor})$ .

### 11.6.2 Rezolvare detaliată

**11.6.3 Codul sursă**



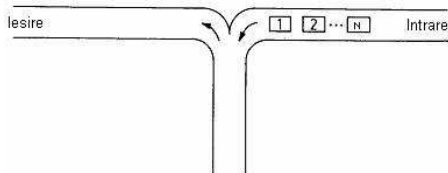
## Capitolul 12

# ONI 2005 clasa a XI-a



### 12.1 Mașina - ONI 2005

Se știe că Balaurul este un împătimit al volanului. Ieri a ajuns la o intersecție (dacă îi putem zice așa) foarte ciudată, ca în figura alăturată ...



La această intersecție au ajuns  $N$  mașini (numerotate de la 1 la  $N$ ). Balaurul se afla în mașina  $X$ . În momentul acela se întrecea cu mașina  $Y$  ( $X$  diferit de

Y). Cele 3 drumuri sunt foarte înguste, așa încât doar o mașină poate să încapă, deci depășirea este imposibilă. Totuși, datorită configurației drumurilor, mașinile își pot schimba poziția la ieșire.

De exemplu, pentru  $N = 3$ , la final avem 5 posibilități de ordonare a celor 3 mașini:

1) 1 2 3 : intră mașina 1 pe drumul din mijloc, și iese 1, intră 2 și iese 2, intră 3 și iese 3

2) 1 3 2 : intră 1 și iese 1, intră 2, intră 3, iese 3, iese 2

3) 2 1 3 : intră 1, intră 2, iese 2, iese 1, intră 3, iese 3

4) 2 3 1 : intră 1, intră 2, iese 2, intră 3, iese 3, iese 1

5) 3 2 1 : intră 1, intră 2, intră 3, iese 3, iese 2, iese 1

Oricare din cele  $M$  (în cazul acesta  $N = 3$ ,  $M = 5$ ) configurații posibile are șanse egale de a se întâmpla.

Balaurul vrea să știe care sunt șansele (în procente) ca la final să iasă în fața mașinii cu care se întrecea.

### Cerință

Ajutați-l pe Balaur să determine șansele de a câștiga, deci de a ieși în fața mașinii  $Y$ .

### Date de intrare

Pe prima linie a fișierului **masina.in** se află 3 numere naturale  $N$ ,  $X$  și  $Y$ , separate prin câte un spațiu, reprezentând numărul de mașini, mașina Balaurului și respectiv mașina concurentului.

### Date de ieșire

Fișierul **masina.out** va conține pe singura sa linie un singur număr real cu primele 2 zecimale exacte (obținute prin trunchiere), și anume șansele (în procente) ca Balaurul să iasă la final în fața concurentului.

### Restricții și precizări

$1 < N < 101$

$0 < X, Y < N + 1$

pentru 50% din teste  $X = 1$

trunchierea la două zecimale exacte a numărului real 60.5673 este 60.56

trunchierea la două zecimale exacte a numărului real 60.5628 este 60.56

trunchierea la două zecimale exacte a numărului real 60.5655 este 60.56

### Exemplu

masina.in	masina.out	Explicație
3 1 3	60.00	Din cele 5 configurații în total, în 3 dintre ele mașina 1 iese în fața mașinii 3, deci șansele sunt de 60%

**Timp maxim de execuție/test:** 0.1 secunde sub Linux și 0.1 secunde sub Windows

### 12.1.1 Indicații de rezolvare - descriere soluție \*

#### Soluția oficială

Problema este de programare dinamică. Există mai multe feluri de rezolvare, iar acesta este unul dintre ele:

Asemănăm acel drum din mijloc cu o stivă, iar operațiile sunt paranteze deschise (atunci când vine o mașină în stivă) și paranteze închise (când pleacă o mașină din stivă).

Construim:

$A[i][j]$  = numărul de posibilități de a ajunge în poziția cu  $i$  paranteze deschise și  $j$  închise

$B[i][j]$  = numărul de posibilități de a încheia situația ( $N$  deschise,  $N$  închise) din poziția  $(i, j)$ , adică  $i$  deschise,  $j$  închise

Presupunem  $X < Y$ , altfel inversăm și scădem la sfârșit.

Variem momentul în care apare  $X$  în stivă (baza), și anume poziția pe care se găsește în stivă când apare.

Construim:

$Q[i][j]$  = numărul de posibilități de procesare a  $i$  mașini având  $j$  în stivă, ținem  $i$  între  $x$  și  $y$  și  $j$  între baza și baza + diferența între  $x$  și  $y$

Momentul în care se ia o decizie (dacă  $x$  e în fața lui  $y$  sau invers), este atunci când stiva e la nivelul baza sau când urmează să procesăm pe  $y$ . Adunăm aceste momente, înmulțindu-le cu  $B$ -ul corespunzător (adică câte posibilități sunt până la sfârșit) și cu  $A$ -ul corespunzător (câte posibilități să ajungem la baza).

Întrucât precizia este doar de două zecimale, putem folosi tipul double pentru a calcula toate posibilitățile.

### 12.1.2 Rezolvare detaliată

### 12.1.3 Codul sursă

## 12.2 Matrice - ONI 2005

Se dă o matrice cu  $N$  linii și 2 coloane de numere întregi. Liniile sunt numerotate cu numere de la 1 la  $N$ , iar coloanele cu 1 și 2.

Există patru operații care pot fi executate asupra matricei:  $S_3, J_3, S_4, J_4$ .

La o operație  $J_k$  se aleg  $k$  linii (distincte) ale matricei și se permută circular în jos, iar la o operație  $S_k$  se aleg  $k$  linii (distincte) și se permută circular în sus ( $k = 3, 4$ ).

...	...		...	...		...	...		...	...
$x_1$	$x_2$		$z_1$	$z_2$		$x_1$	$x_2$		$y_1$	$y_2$
...	...		...	...		...	...		...	...
$y_1$	$y_2$	→	$x_1$	$x_2$		$y_1$	$y_2$	→	$z_1$	$z_2$
...	...		...	...		...	...		...	...
$z_1$	$z_2$		$y_1$	$y_2$		$z_1$	$z_2$		$x_1$	$x_2$
...	...		...	...		...	...		...	...
Operație $J_3$						Operație $S_3$				

Operațiile  $J_4$  și  $S_4$  sunt similare, numai că în loc de 3 linii se alege 4.

### Cerință

Scrieți un program care prin efectuarea a cel mult  $2 * N$  operații de tip  $S_3$ ,  $J_3$ ,  $S_4$ ,  $J_4$  asupra matricei date să o transforme astfel încât nici una dintre coloanele ei să nu conțină un subșir strict crescător de lungime mai mare decât  $\lceil \sqrt{N} \rceil$  (adică cel mai mic întreg mai mare sau egal decât  $\sqrt{N}$ ).

### Date de intrare

Fișierul de intrare **matrice.in** conține pe prima linie numărul natural  $N$ . Pe fiecare din următoarele  $N$  linii sunt câte 2 numere întregi separate printr-un spațiu, reprezentând elementele matricei.

### Date de ieșire

Fișierul de ieșire **matrice.out** va conține câte o operație pe linie. O operație este identificată prin tipul ei (poate fi J3, S3, J4 sau S4) și 3 numere (pentru J3 și S3) sau 4 numere (pentru J4 și S4) în ordine strict crescătoare, reprezentând rândurile asupra cărora este executată operația. Tipul operației și celelalte numere trebuie să fie separate prin exact un spațiu.

Atenție! Tipul operației este format din două caractere scrise fără spațiu între ele.

### Restricții și precizări

$$6 \leq N \leq 30000$$

Un subșir strict crescător al unui șir  $a_1, a_2, \dots, a_N$  este un șir  $a_{i1}, a_{i2}, \dots, a_{ik}$ , unde  $i1 < i2 < \dots < ik$  și  $a_{i1} < a_{i2} < \dots < a_{ik}$ .

Elementele matricei sunt numere întregi mai mari sau egale cu 0 și mai mici sau egale cu 65000.

### Exemplu

matrice.in	matrice.out
6	S4 1 3 4 5
1 2	J3 4 5 6
3 1	
4 4	
6 3	
5 5	
2 6	

### Explicație

După efectuarea operațiilor, matricea va fi

4	4
3	1
6	3
2	6
5	5
1	2

Lungimea celui mai lung subșir strict crescător de pe prima coloană este 2, iar de pe cea de-a doua este 3 (2 și 3 sunt mai mici sau egale decât  $\lceil \sqrt{N} \rceil = \lceil \sqrt{6} \rceil = 3$ ).

**Timp maxim de execuție/test:** 0.2 secunde sub Linux și 0.5 secunde sub Windows.

### 12.2.1 Indicații de rezolvare - descriere soluție \*

#### Soluția oficială

Ideea este de a permuta rândurile matricei astfel încât să respecte proprietatea cerută și anume aceea ca oricare coloană să nu conțină un subșir strict crescător de lungime mai mare decât  $\lceil N \rceil$  și de a scrie permutarea respectivă folosind operațiile permise.

Permutarea cerută se obține astfel: mai întâi se sortează rândurile matricei descrescător după valorile primei coloane. Apoi pentru grupe de câte  $\sqrt{N}$  rânduri, se sortează descrescător după valorile celei de-a doua coloane. Cel mai lung subșir crescător în prima coloană poate fi găsit numai în cadrul grupelor și deci are lungimea maxim  $\lceil N \rceil$ . În a doua coloană, cel mai lung subșir crescător poate fi obținut alegând cât mai convenabil un element din fiecare grupă (cele din cadrul grupelor sunt sortate descrescător) și deci are lungimea maximă posibilă egală cu numărul grupelor, adică  $\lceil N \rceil$ .

Pentru a obține o permutare cu ajutorul operațiilor  $S_3, J_3, S_4, J_4$ , observăm că orice transpoziție se poate scrie în funcție de aceste operații. Orice permutare se poate descompune în cicluri (suma lungimilor lor fiind egală cu  $N$ ). Cum orice ciclu de lungime  $L$  se poate descompune în  $L$  transpoziții și (vom demonstra că) orice transpoziție se poate obține prin efectuarea a 2 dintre operațiile permise, rezultă că orice permutare se poate obține prin efectuarea a cel mult  $2 * N$  operații permise.

Pentru a arăta că orice transpoziție se poate obține din operațiile  $S_3, J_3, S_4, J_4$ , considerăm cazurile:

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \xrightarrow{S_4} \begin{pmatrix} b \\ c \\ d \\ a \end{pmatrix} \xrightarrow{J_3 \text{ (pentru ultimele 3)}} \begin{pmatrix} b \\ a \\ c \\ d \end{pmatrix} \Rightarrow \begin{array}{l} \text{putem interschimbarândul } a \text{ cu } b \text{ dacă} \\ \text{după ele mai sunt cel puțin 2 rânduri} \end{array}$$

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \xrightarrow{J_1} \begin{bmatrix} d \\ a \\ b \\ c \end{bmatrix} \xrightarrow{S_1 \text{ (pentru primele 3)}} \begin{bmatrix} a \\ b \\ d \\ c \end{bmatrix} \Rightarrow \text{putem interschimba r\u00e2ndul } c \text{ cu } d \text{ dac\u0103} \\ \text{\u00eenaintea lor mai sunt cel pu\u021bin 2 r\u00e2nduri}$$

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \xrightarrow{S_1} \begin{bmatrix} b \\ c \\ d \\ a \end{bmatrix} \xrightarrow{J_1 \text{ (pentru primele 3)}} \begin{bmatrix} d \\ b \\ c \\ a \end{bmatrix} \Rightarrow \text{putem interschimba r\u00e2ndul } a \text{ cu } d \text{ dac\u0103} \\ \text{\u00eenre ele mai sunt cel pu\u021bin 2 r\u00e2nduri}$$

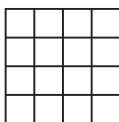
Pentru dou\u0103 r\u00e2nduri  $a$  \u0219i  $b$ , cel pu\u021bin unul dintre cazuri este adev\u00e2rat.

### 12.2.2 Rezolvare detaliat\u0103

### 12.2.3 Codul surs\u0103

## 12.3 Ziduri - ONI 2005

Un ziar are redac\u021bia la etajul unei cl\u0103diri. Acest etaj de form\u0103 p\u0103tratic\u0103 este alc\u0103tuit numai din camere de aceea\u0219i dimensiune \u0219i de form\u0103 p\u0103tratic\u0103. Pentru un etaj cu  $4 \times 4$  camere avem configura\u021bia:



Unele dintre zidurile camerelor lipsesc. Directorul \u0219i redactorul \u0219ef au fiecare biroul \u00een camere separate. Directorul are biroul \u00een camera de pe linia 1 \u0219i coloana 1, iar redactorul \u0219ef \u00een camera de pe ultima linie \u0219i ultima coloan\u0103. Deplasarea \u00eenre dou\u0103 camere vecine se poate face numai dac\u0103 ele nu au zid desp\u0103r\u021bitor. Pentru a m\u0103ri viteza de deplasare \u00eenre birourile directorului \u0219i redactorului \u0219ef se ia decizia ca unele ziduri s\u0103 fie desfiin\u021bate. Un studiu f\u0103cut de departamentul administrativ arat\u0103 c\u0103 deplasarea \u00eenre dou\u0103 camere f\u0103r\u0103 zid conduce la un cost de o unitate monetar\u0103, iar deplasarea \u00eenre dou\u0103 camere care au avut zid \u0219i a fost d\u0103rmat conduce la un cost de  $p$  unit\u0103\u021bi monetare.

#### Cerin\u021b\u0103

Determina\u021bi costul minim al unei deplas\u0103ri de la camera directorului la camera redactorului \u0219ef. Dintre toate deplas\u0103rile de cost minim, determina\u021bi num\u0103rul minim de ziduri ce trebuie d\u0103r\u0103mate \u00eentr-o astfel de deplasare.

**Date de intrare**

Fișierul de intrare **ziduri.in** conține pe prima linie  $n$  (numărul de camere de pe o linie, respectiv coloană) și  $p$ , costul trecerii de la o cameră la alta între care s-a dărâmat zidul despărțitor; cele două numere fiind separate printr-un spațiu. Pe următoarele  $n$  linii se află câte  $n$  numere naturale din mulțimea  $\{0, 1, \dots, 15\}$  separate prin câte un spațiu. Aceste numere naturale transformate în baza 2 (pe 4 biți) ne dau informații despre existența zidurilor în jurul camerei (1 pentru zid și 0 în caz contrar). De exemplu dacă un astfel de număr are reprezentarea  $abcd$  în baza 2, atunci  $a$  este pentru zidul din strea vest,  $b$  pentru cel din nord,  $c$  pentru cel din est, iar  $d$  pentru cel din sud.

**Date de ieșire**

Fișierul de ieșire **ziduri.out** va conține pe prima linie costul minim al deplasării de la director la redactorul șef, iar pe linia a doua numărul minim de ziduri dărâmate.

**Restricții și precizări**

$1 < n < 101$

$0 < p < 101$

Nu se acordă punctaje parțiale.

**Exemplu**

ziduri.in	ziduri.out
4 3	8
9 1 1 0	1
12 5 5 1	
1 5 5 4	
4 6 12 0	

**Explicație**

Configurația birourilor redacției (zidurile sunt marcate cu linie îngroșată)

D			
			R

**Timp maxim de execuție/test:** 0.1 secunde sub Linux și 0.1 secunde sub Windows

**12.3.1 Indicații de rezolvare - descriere soluție \*****Soluția 1**

Se consideră graful format astfel: pentru fiecare cameră se consideră un nod, iar între două camere vecine pe orizontală/verticală (deci două noduri) o muchie de cost 1 dacă nu există zid între cele două camere, respectiv  $p$  dacă există zid. Pe acest graf se calculează cu algoritmul lui Dijkstra costurile minime de la camera stanga-sus la toate celelalte camere. Apoi se construiește graful orientat aciclic

al tuturor drumurilor minime de la camera stanga sus la camera dreapta jos. Pe acest graf aciclic se calculează cu o parcurgere drumul dintre cele două camere care folosește număr minim de muchii formate din ziduri. Pentru o complexitate mică, algoritmul lui Dijkstra trebuie implementat cu heapuri. Astfel, complexitatea totală e de  $N^2 \log N$ .

#### Soluția 2

Pe același graf, putem afla această distanță folosind algoritmul Bellman - Ford - Moore. Deși acesta poate conduce la o complexitate de  $O(N^4)$ , ne putem baza pe faptul că  $p$  este destul de mic (mai mic decât 100 (adică maximum lui  $N$ )) și acesta reduce la  $O(N^3)$  complexitatea, cu care putem obține punctaj maxim.

### 12.3.2 Rezolvare detaliată

### 12.3.3 Codul sursă

## 12.4 Lsort - ONI 2005

Se consideră o listă simplu înlănțuită ( $L1$ ) ce conține numerele întregi de la 1 la  $N$  (într-o ordine oarecare). Se dorește construirea unei alte liste simplu înlănțuite ( $L2$ ) care să conțină numerele din lista  $L1$  în ordine crescătoare. Pentru a obține lista  $L2$ , se vor șterge elemente din  $L1$  și se vor adăuga în  $L2$ , conform procedurii descris în continuare: inițial lista  $L2$  e vidă. La primul pas putem șterge orice element din  $L1$  și acesta se adaugă în  $L2$ . Apoi, în următorii  $N - 1$  pași, se poate șterge orice element din  $L1$ , dar acesta poate fi adăugat numai la începutul sau la sfârșitul lui  $L2$ . La final,  $L1$  nu va conține nici un element, iar  $L2$  trebuie să conțină numerele de la 1 la  $N$  în ordine crescătoare. Întrucât există mai multe posibilități de a construi  $L2$ , vom defini în continuare costul construcției lui  $L2$  și vom dori să determinăm o posibilitate de construcție a lui  $L2$  care are costul minim.

Algoritmul de construcție al lui  $L2$  constă din  $N$  pași, la fiecare pas ștergându-se un element din  $L1$  și adăugând acest element în  $L2$  (conform restricțiilor precizate). La pasul  $i$  ( $1 \leq i \leq N$ ), lista  $L1$  conține  $N - i + 1$  elemente și lista  $L2$  conține  $i - 1$  elemente. Dacă se mută elementul de pe poziția  $k$  din  $L1$  în  $L2$  la pasul  $i$  ( $1 \leq k \leq N - i + 1$ ), costul acestei mutări este egal cu  $k * i$ . După mutarea elementului, lista  $L1$  va avea cu un element mai puțin (toate elementele de pe poziții mai mari decât  $k$  vor ajunge cu o poziție mai în față) și lista  $L2$  va avea cu un element mai mult. Costul total al construcției lui  $L2$  este egal cu suma costurilor mutărilor efectuate la fiecare dintre cei  $N$  pași.

#### Date de intrare



Pe prima linie a fișierului **lsort.in** se află numărul  $N$ , reprezentând numărul de elemente din  $L1$ . Următoarea linie conține numerele întregi de la 1 la  $N$ , separate prin cel puțin un spațiu, în ordinea în care se află acestea poziționate în lista  $L1$  (primul număr se află pe poziția 1 în  $L1$ , al doilea număr pe poziția 2 ș.a.m.d.).

#### Date de ieșire

Pe prima linie a fișierului de ieșire **lsort.out** se va afișa costul minim de construcție a lui  $L2$ . Pe următoarea linie se va afișa modul de construcție al acesteia. Pe această linie se va afișa ordinea în care sunt mutate elementele din lista  $L1$  în lista  $L2$ . Primul număr afișat va reprezenta numărul mutat din  $L1$  în  $L2$  la primul pas; al doilea număr va reprezenta numărul mutat la pasul 2 ș.a.m.d. Numerele afișate trebuie separate prin cel puțin un spațiu. În cazul în care există mai multe posibilități de construcție a listei  $L2$  având costul minim, se poate afișa oricare dintre ele.

#### Restricții și precizări

$$1 \leq N \leq 1000$$

#### Exemple

lsort.in	lsort.out	lsort.in	lsort.out
4	15	7	43
4 1 3 2	3 4 2 1	6 3 5 4 1 7 2	6 5 4 3 2 1 7

#### Explicație pentru primul exemplu

La pasul 1,  $L1 = [4, 1, 3, 2]$  și  $L2 = []$ . Se șterge din  $L1$  elementul 3 (care se află pe poziția 3) și se introduce în  $L2$ . Costul acestei mutări este  $3 * 1 = 3$ .

La pasul 2,  $L1 = [4, 1, 2]$  și  $L2 = [3]$ . Se șterge din  $L1$  elementul 4 (care se află pe poziția 1) și se introduce în  $L2$  (este evident că acest element trebuie adăugat la sfârșitul listei  $L2$  și nu la începutul ei; în caz contrar, lista  $L2$  nu ar mai fi sortată). Costul acestei mutări este  $1 * 2 = 2$ .

La pasul 3,  $L1 = [1, 2]$  și  $L2 = [3, 4]$ . Se șterge din  $L1$  elementul 2 (care se află pe poziția 2) și se introduce în  $L2$  (din nou, poziția unde trebuie adăugat elementul este evidentă : la începutul listei  $L2$ ). Costul acestei mutări este  $2 * 3 = 6$ .

La pasul 4,  $L1 = [1]$  și  $L2 = [2, 3, 4]$ . Se șterge din  $L1$  elementul 1 (care se află pe poziția 1) și se introduce în  $L2$ . Costul acestei mutări este  $1 * 4 = 4$ .

Costul total al construcției listei  $L2$  este  $3 + 2 + 6 + 4 = 15$ .

**Timp maxim de execuție/test:** 0.3 secunde sub Linux și 1.0 secunde sub Windows

### 12.4.1 Indicații de rezolvare - descriere soluție \*

Problema se rezolva folosind programare dinamică.

Vom calcula o matrice  $cmin[i][j]$  = costul minim al unor operații în urma cărora, în  $L2$ , am obținut intervalul de numere  $[i, j]$  ( $i, i + 1, \dots, j - 1, j$ ).

Datorită modului de construcție a lui  $L2$ , pentru a obține intervalul  $[i, j]$ , ori am adăugat la ultima mutare numărul  $i$  și înainte am obținut intervalul  $[i+1, j]$ , ori am adăugat numărul  $j$  și înainte am obținut intervalul  $[i, j-1]$ .

Vom încerca ambele variante (să îl mutăm pe  $i$  ultimul sau să îl mutăm pe  $j$  ultimul) și vom alege costul minim.

Pentru a muta oricare din cele 2 numere ( $i$  sau  $j$ ) trebuie să cunoaștem poziția lui curentă în lista L1. Această poziție se poate obține parcurgând toate elementele din intervalul  $[i, j]$  și determinând câte elemente se aflau, inițial, pe o poziție mai mică în L1 (deoarece aceste elemente au fost deja introduse în L2 și, deci, nu se mai află în L1).

Procedând astfel, obținem o complexitate  $O(N^3)$ .

Putem, însă, calcula poziția curentă a numărului  $i$  sau  $j$  în  $O(1)$  (de exemplu, în cazul lui  $i$ , poziția curentă în L1 este poziția calculată pentru intervalul  $[i, j-1]$ , luând, în plus, în considerare și elementul  $j$  - care poate sau nu să se afle înaintea lui  $i$  în L1).

Astfel, complexitatea finală este  $O(N^2)$ .

Pentru compilatoarele Borland (care nu pot memora o matrice de  $N * N$  elemente long), se observă că, pentru a calcula costul pentru intervalele de lungime  $L$  avem nevoie doar de costurile pentru intervalele de lungime  $L-1$ . Mai rămâne problema reconstituirii soluției. Pentru aceasta, putem memora 1 bit pentru fiecare pereche  $[i, j]$  (am ales pe  $i$  sau pe  $j$  ca ultim element)  $\Rightarrow N^2/8$  octeți.

### 12.4.2 Rezolvare detaliată

### 12.4.3 Codul sursă

## 12.5 Pătrat - ONI 2005

Se numește *pătrat magic* de ordin  $N$  o matrice cu 3 linii și 3 coloane cu elemente întregi nenegative (mai mari sau egale cu 0) cu proprietatea că suma elementelor oricărei linii și oricărei coloane este  $N$ .

#### Cerință

Scrieți un program care să determine numărul pătratelor magice de ordin  $N$  modulo 30103.

#### Date de intrare

Fișierul de intrare **patrat.in** conține pe prima linie numărul  $N$ .

#### Date de ieșire

Fișierul de ieșire **patrat.out** va conține numărul pătratelor magice de ordin  $N$  modulo 30103.

#### Restricții și precizări

$1 \leq N \leq 1000000$

#### Exemple

patrat.in	patrat.out	patrat.in	patrat.out
2	21	5	231

**Timp maxim de execuție/test:** 0.1 secunde pentru Linux și 0.1 secunde pentru Windows.

### 12.5.1 Indicații de rezolvare - descriere soluție \*

Numărul pătratelor magice de ordin  $N$  este  $(C_{N+2}^2)^2 - 3C_{N+3}^4$ .

Fie  $A = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}$  o matrice care respectă proprietățile date.

Observăm că dacă fixăm numerele întregi  $a_{1,1}, a_{1,2}, a_{1,3}, a_{2,1}, a_{2,2}, a_{2,3}$ , astfel încât

$$\begin{cases} a_{1,1} + a_{1,2} + a_{1,3} = N \\ a_{2,1} + a_{2,2} + a_{2,3} = N \end{cases}, \text{ atunci obținem } \begin{cases} a_{3,1} = N - a_{1,1} - a_{2,1} \\ a_{3,2} = N - a_{1,2} - a_{2,2} \\ a_{3,3} = N - a_{1,3} - a_{2,3} \end{cases}.$$

În plus,  $a_{3,1} + a_{3,2} + a_{3,3} = N$ .

Singura condiție care trebuie impusă este ca  $a_{3,1}, a_{3,2}, a_{3,3} \geq 0$ .

O ecuație de forma  $x + y + z = N$  cu  $x, y, z \geq 0$  are  $C_{N+2}^2$  soluții. (Dacă fixăm  $x$  și  $y$ ,  $z$  va rezulta ca  $N - x - y$ . Pentru un  $x$  fixat, există  $N - x + 1$  posibilități pentru  $y$ . Deci ecuația are  $\sum_{x=0}^N (N - x + 1) = (N + 1) + N + \dots + 1 = \frac{(N+1)(N+2)}{2} = C_{N+2}^2$  soluții.)

Ecuațiile  $a_{1,1} + a_{1,2} + a_{1,3} = N$  și  $a_{2,1} + a_{2,2} + a_{2,3} = N$  au deci  $(C_{N+2}^2)^2$  soluții.

Dintre acestea trebuie să le scădem pe cele de unde unul dintre  $a_{3,1}, a_{3,2}, a_{3,3}$  rezultă negativ.

$a_{3,1}$  rezultă negativ când  $a_{1,1} + a_{2,1} \geq N + 1$ .

Pentru  $a_{1,1}$  fixat, există  $N - a_{1,1} + 1$  posibilități pentru  $a_{1,2}$ . De asemenea, există  $a_{1,1}$  posibilități pentru  $a_{2,1}$  și anume  $N - a_{1,1} + 1 \leq a_{2,1} \leq N$ , iar pentru fiecare din această posibilitate (adică pentru  $a_{2,1}$  fixat) există  $N - a_{2,1} + 1$  posibilități pentru  $a_{2,2}$ .

Însumând după  $a_{1,1}$ , obținem  $\sum_{a_{1,1}=1}^N (N - a_{1,1} + 1) * (1 + 2 + \dots + a_{1,1}) = C_{N+3}^4$  posibilități. (Această sumă nu trebuie neapărat calculată; putem să facem un for după  $a_{1,1}$ .)

Analog pentru  $a_{3,2}$  și  $a_{3,3}$ .

Mai facem observația că nu se poate ca două dintre  $a_{3,1}, a_{3,2}, a_{3,3}$  să rezulte negative deodată. Presupunnd că  $a_{1,1} + a_{2,1} \geq N + 1$  și  $a_{1,2} + a_{2,2} \geq N + 1$ , ar rezulta că  $a_{2,1} + a_{2,2} \geq 2 * N - (a_{1,1} + a_{1,2}) + 2 \geq N + 2$ , ceea ce este fals.

Deci numărul cerut este  $(C_{N+2}^2)^2 - 3C_{N+3}^4$ .

### 12.5.2 Rezolvare detaliată

### 12.5.3 Codul sursă \*

```
import java.io.*; // rezultat=(comb(N+2,2))^2 - 3 comb(N+3,4)
class patrat      // atentie la 1.000.000 * 1.000.000 = depaseste int !!!
{
    static final int prim=30103;

    public static void main (String[] args) throws IOException
    {
        int n,i,rf,r1,r2,r3,r4;
        StreamTokenizer st=new StreamTokenizer(
            new BufferedReader(new FileReader("9-patrat.in")));
        PrintWriter out=new PrintWriter(new BufferedWriter(
            new FileWriter("patrat.out")));

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

        r1=comb(n+2,2);
        r2=(r1*r1)%prim;
        r3=comb(n+3,4);
        r4=(3*r3)%prim; // acum rf=r2-r4
        rf=(r2+prim-r4)%prim;
        //System.out.println(rf);
        out.print(rf);
        out.close();
    }

    static int comb(int n,int k)
    {
        if(k>n/2) k=n-k; //optim !
        int p,i,j,d;
        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(y[j],x[i]);
                y[j]=y[j]/d;
            }
        }
    }
}
```

```

        x[i]=x[i]/d;
        if(y[j]==1) break;
    }
}

p=1;
for (i=1;i<=k;i++) p=(p*(x[i]%prim))%prim; // atentie la x[i]%prim ...
return p;
} //comb

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

```

## 12.6 Cșir - ONI 2005

Un șir *circular* este un șir format numai din caracterele "A" și "B" care are următoarele proprietăți:

- are lungime  $N \geq 1$  (nu poate fi șirul vid);
- se consideră că după ultimul caracter din șir urmează primul caracter din șir.

Această proprietate implică faptul că orice șir circular are  $N$  subsecvențe de lungime  $L$  ( $1 \leq L \leq N$ ).

O subsecvență de lungime  $L$  a unui șir circular  $S$  este un șir de caractere (obișnuit, nu circular) format din  $L$  caractere aflate pe poziții consecutive în șirul  $S$ . De exemplu, șirul circular "ABAAB" are 5 subsecvențe de lungime 3: "ABA", "BAA", "AAB", "ABA" și "BAB" (ele nu sunt distincte ca valoare, însă diferă ca poziție de început în șirul din care fac parte).

Un **cșir** este un șir circular care are în plus următoarea proprietate: pentru orice  $L$  ( $1 \leq L \leq N$ ) și oricare două subsecvențe de lungime  $L$  (să le numim  $S1$  și  $S2$ ), numărul de caractere "A" din  $S1$  diferă față de numărul de caractere "A" din  $S2$  cu cel mult 1 (în valoare absolută).

Să considerăm șirul circular "BBAABAA". Acest șir nu este un **cșir**, deoarece există subsecvențele "BBAAB" și "AABAA" (de lungime 5), care conțin 2, respectiv 4 caractere "A" (diferența dintre numărul de caractere "A" este, astfel, 2). De asemenea, șirul "ABABAABAAB" nu este un **cșir**, deoarece conține subsecvențe "AABAA" și "BABAB" pentru care diferența dintre numărul de caractere "A" este mai mare decât 1 (în valoare absolută).

Șirurile circulare "ABA" și "AABABAAB" sunt, în schimb, **csir**-uri, deoarece oricare ar fi două subsecvențe  $S1$  și  $S2$  având aceeași lungime, diferența dintre numărul de caractere "A" din  $S1$  și numărul de caractere "A" din  $S2$  este mai mică sau egală cu 1 (în valoare absolută).

#### Cerință

Dându-se mai multe șiruri circulare, determinați dacă ele sunt **csir**-uri.

#### Date de intrare

Prima linie a fișierului de intrare **csir.in** conține numărul întreg  $S$ , reprezentând numărul de șiruri conținute în fișier. Pe fiecare dintre următoarele  $S$  linii se află câte un șir circular.

#### Date de ieșire

În fișierul de ieșire **csir.out** se vor scrie  $S$  linii. Pe a  $K$ -a linie din acest fișier, se va afișa 1, dacă al  $K$ -lea șir din fișierul de intrare este un **csir**, sau 0, în caz contrar.

#### Restricții și precizări

$$1 \leq S \leq 20$$

Lungimea fiecărui șir circular din fișierul de intrare este cuprinsă între 1 și 50000 (inclusiv).

Șirurile conțin numai caracterele "A" și "B" (nu și "a" sau "b").

Nu se acordă punctaje parțiale.

#### Exemplu

csir.in	csir.out
4	0
BBAABAA	0
ABABAABAAB	1
ABA	1
AABABAAB	

**Timp maxim de execuție/test:** 0.2 secunde sub Linux și 0.5 secunde sub Windows

### 12.6.1 Indicații de rezolvare - descriere soluție \*

Vom presupune că șirul conține mai multe A-uri decât B-uri (dacă nu se întâmplă astfel, toate B-urile se pot transforma în A-uri și toate A-urile în B-uri). Șirul are o structură de forma următoare:  $A^{i1}BA^{i2}BA^{i3}B..A^{ik}B$  (unde  $A^i$  reprezintă o secvență de  $i$  "A"-uri consecutive). Se poate demonstra ușor că, într-un **csir**, există doar secvențe de forma  $A^iB$  sau  $A^{i+1}B$ . În continuare, vom considera șirul compactat în care fiecare secvență de forma  $A^iB$  este notată cu A și fiecare secvență de forma  $A^{i+1}B$  este notată cu B. Acest șir compactat este tot un șir circular format numai din "A"-uri și "B"-uri, având o lungime mai mică decât șirul inițial. Se poate demonstra (nu chiar la fel de ușor) că șirul inițial este un **csir** dacă și numai dacă șirul compactat este un **csir**. Astfel, se folosește o rezolvare recursivă, care se termină când șirul conține numai "A"-uri (sau numai "B"-uri)

sau când nu mai este de forma precizată (nu conține numai secvențe de forma  $A^i B$  și  $A^{i+1} B$ ). Complexitatea rezolvării este  $O(N)$  pentru fiecare șir.

### 12.6.2 Rezolvare detaliată

### 12.6.3 Codul sursă





## Capitolul 13

# ONI 2006 clasa a XI-a



### 13.1 borg - ONI 2006

Oricine a urmărit serialul Star Trek își aduce aminte de borgi și de nava lor spațială în formă de cub. Una dintre problemele pe care și-au pus-o înainte de a construi nava a fost următoarea.

Nava borgilor are forma unui paralelipiped dreptunghic de dimensiuni  $N \times M \times H$ , împărțit în camere de dimensiune  $1 \times 1 \times 1$ . Pentru ca nava să poată funcționa, în aceste camere trebuie plasate  $K$  motoare de propulsie, în fiecare cameră putându-se plasa cel mult un motor.

O cameră poate fi identificată printr-un triplet  $(a, b, c)$ , unde  $1 \leq a \leq N$ ,  $1 \leq b \leq M$ ,  $1 \leq c \leq H$ , reprezentând coordonatele sale.

Un plan al paralelipipedului este o mulțime de camere de unul dintre următoarele 3 tipuri:

$\{(a, b, c) | a \text{ fixat}, 1 \leq b \leq M, 1 \leq c \leq H\}$  - în total sunt  $N$  plane de acest tip;

$\{(a, b, c) | b \text{ fixat}, 1 \leq a \leq N, 1 \leq c \leq H\}$  - în total sunt  $M$  plane de acest tip;  
 $\{(a, b, c) | c \text{ fixat}, 1 \leq a \leq N, 1 \leq b \leq M\}$  - în total sunt  $H$  plane de acest tip.

### Cerință

Se cere să se găsească  $R$ , numărul de moduri în care se pot plasa cele  $K$  motoare, astfel încât orice plan al paralelipipedului să conțină cel puțin o cameră ocupată de un motor.

Deoarece numărul cerut poate fi foarte mare, este suficient să aflați restul împărțirii lui  $R$  la 30103.

### Date de intrare

Fișierul de intrare **borg.in** va conține o singură linie pe care sunt scrise numerele naturale  $N$ ,  $M$ ,  $H$  și  $K$  separate prin câte un spațiu.

### Date de ieșire

Fișierul de ieșire **borg.out** va conține o singură linie pe care va fi scris un singur număr natural reprezentând restul împărțirii lui  $R$  la 30103.

### Restricții și precizări

- $1 \leq N, M, H \leq 20$
- $1 \leq K \leq N * M * H$
- 80% din teste vor avea  $K \leq 2000$

### Exemple

borg.in	borg.out	borg.in	borg.out
3 1 2 4	12	3 1 2 2	0

**Timp maxim de execuție/test:** 0.7 secunde

## 13.1.1 Indicații de rezolvare - descriere soluție \*

### Soluția oficială

Numărul total de moduri în care se pot plasa  $K$  motoare în paralelipiped, fără restricția ca fiecare plan să conțină cel puțin un motor, este  $C_{N \cdot M \cdot H}^K$ . Din acestea, trebuie să le scădem pe cele care nu sunt bune.

Dacă notăm cu  $X_i$  ( $1 \leq i \leq N$ ) numărul de moduri în care se pot plasa cele  $K$  motoare astfel încât planul  $i$  (de tip 1) să fie gol, cu  $Y_i$  ( $1 \leq i \leq M$ ) numărul de moduri astfel încât planul  $i$  (de tip 2) să fie gol și cu  $Z_i$  ( $1 \leq i \leq H$ ) numărul de moduri astfel încât planul  $i$  (de tip 3) să fie gol, atunci numărul căutat va fi

$$C_{N \cdot M \cdot H}^K - \left| \bigcup_{1 \leq i \leq N} X_i \cup \bigcup_{1 \leq i \leq M} Y_i \cup \bigcup_{1 \leq i \leq H} Z_i \right|.$$

Cardinalul reuniunii se poate dezvolta cu principiul includerii și excluderii, fiecare mulțime rezultată conținând o intersecție de  $a$  mulțimi  $X$ ,  $b$  mulțimi  $Y$  și  $c$  mulțimi  $Z$ .

Cardinalul unei astfel de mulțimi ar reprezenta numărul de moduri în care se pot pune  $K$  motoare, astfel încât  $a$  dintre planele de tip 1 să fie goale,  $b$  dintre

planele de tip 2 să fie goale și  $c$  dintre planele de tip 3 să fie goale, adică ar fi  $C_{(N-a) \cdot (M-b) \cdot (H-c)}^K$ . În dezvoltarea produsă de principiul includerii și excluderii vor fi  $C_N^a \cdot C_M^b \cdot C_H^c$  astfel de mulțimi.

Așadar numărul total va fi

$$\sum_{a=1}^N \sum_{b=1}^M \sum_{c=1}^H (-1)^{a+b+c} C_N^a \cdot C_M^b \cdot C_H^c \cdot C_{(N-a) \cdot (M-b) \cdot (H-c)}^K$$

Se observă că este nevoie de toate combinațiile până la maximum dintre  $N$ ,  $M$  și  $H$ , dar numai de cele care au  $K$  sus pentru restul. Acestea se pot calcula în complexitatea  $O(N \cdot M \cdot H \cdot K)$  folosind regula generală  $C_n^k = C_{n-1}^k + C_{n-1}^{k-1}$  ( $k \leq K$  și  $n \leq N * M * H$ ). Pentru a reduce complexitatea la  $O(N \cdot M \cdot H)$ , putem calcula  $C_n^K$  direct din  $C_{n-1}^K$ , folosindu-ne de faptul ca numărul 30103 este prim și deci orice număr are un invers modular (care înmulțit cu el dă restul 1 la împărțirea la 30103).

### 13.1.2 Rezolvare detaliată

### 13.1.3 Codul sursă

## 13.2 diamant - ONI 2006

O firmă produce un tip nou de diamante de formă dreptunghiulară și de calități diferite. Pentru a calcula calitatea unui diamant firma împarte diamantul în  $N * M$  pătrățele formând o matrice cu  $N$  linii numerotate de la 1 la  $N$  și  $M$  coloane numerotate de la 1 la  $M$ . Pătrățul de pe linia  $i$  și coloana  $j$  poate influența calitatea diamantului în felul următor ( $1 \leq i \leq N$ ,  $1 \leq j \leq M$ ):

- dacă pătrățul conține impurități este marcat cu  $-1$  și va diminua calitatea diamantului cu  $i * j$
- dacă pătrățul este simplu este marcat cu  $0$  și nu schimbă calitatea diamantului
- dacă pătrățul conține aur este marcat cu  $+1$  și va mări calitatea diamantului cu  $i * j$ .

Fiecare pătrățel va fi marcat cu unul dintre cele trei numere  $(-1, 0, +1)$ .

Un client bogat vrea să cumpere cât mai multe diamante diferite, de aceeași calitate  $X$ . Două diamante sunt diferite dacă există cel puțin un pătrățel de pe o linie  $i$  și coloană  $j$  marcat diferit în cele două diamante.

**Cerință**

Ajutați firma să poată răspunde la astfel de cereri scriind un program care pentru un anumit  $X$  găsește numărul de diamante diferite de calitate  $X$ .

#### Date de intrare

Pe prima linie a fișierului de intrare **diamant.in** sunt scrise trei numere întregi  $N M X$  separate prin câte un spațiu reprezentând numărul de linii, numărul de coloane ale unui diamant, și respectiv calitatea cerută.

#### Date de ieșire

Pe prima linie din fișierul de ieșire **diamant.out** se va afla numărul de diamante diferite cu calitatea cerută, modulo 10000.

#### Restricții și precizări

- $0 < N < 21$
- $0 < M < 21$
- $-2^{31} < X < 2^{31}$

#### Exemplu

diamant.in	diamant.out	Explicatii
2 2 7	3	Matricile corespunzătoare celor 3 diamante sunt: -1 +1 +1 0 +1 +1 +1 +1 +1 +1 0 +1

**Timp maxim de execuție/test:** 2 secunde

### 13.2.1 Indicații de rezolvare - descriere soluție \*

#### Soluția oficială

Soluția se bazează pe faptul ca valoarea maximă în modul a lui  $X$  este

$$1 * 1 + 1 * 2 + \dots + 1 * 19 + 1 * 20 + 2 * 1 + 2 * 2 + \dots + 2 * 20 + 3 * 1 + \dots + 19 * 20 + 20 * 20$$

în cazul în care valorile lui  $N$  și  $M$  sunt maxime adică 20 și avem +1 (sau -1) în toată matricea. Suma are valoarea 44100.

Odată observat acest fapt se poate deduce că o soluție asemănătoare cu cea a problemei rucsacului se încadrează în limita de timp. Capacitatea rucsacului ar fi  $X$  iar obiectele ar fi pătrățelele, greutatea fiind  $i * j$  pentru pătrățelul de pe linia  $i$  coloana  $j$ .

### 13.2.2 Rezolvare detaliată

### 13.2.3 Codul sursă

### 13.3 matrice - ONI 2006

Fie  $A$  o matrice dreptunghiulară de numere întregi cu  $N$  linii numerotate de la 1 la  $N$  și  $M$  coloane numerotate de la 1 la  $M$ . În matricea  $A$  oricare două elemente consecutive de pe aceeași linie sunt distincte.

Se definește un șir valid de numere întregi ca fiind fie un șir crescător, fie un șir descrescător, fie un șir crescător concatenat cu un șir descrescător, fie un șir descrescător concatenat cu unul crescător. Exemple de șiruri valide sunt: 1 2 3 7, 8 5 2 1, 3 5 6 2, 4 1 5 6.

Se definește o submatrice a lui  $A$  de coordonate  $(l_1, c_1, l_2, c_2)$  ca fiind matricea formată din toate elementele  $A(i, j)$ , cu  $l_1 \leq i \leq l_2$  și  $c_1 \leq j \leq c_2$ .

O submatrice a lui  $A$  este validă dacă liniile sale sunt șiruri valide.

**Atenție!** O submatrice validă poate avea pe o linie un șir crescător de numere, pe a doua un șir descrescător, pe a treia un șir crescător concatenat cu unul descrescător etc. Deci, liniile unei submatrice valide nu trebuie să fie neapărat șiruri de același tip.

Aria unei submatrice este egală cu numărul de elemente din care este formată submatricea.

#### Cerință

Se cere să se găsească o submatrice validă a lui  $A$  de arie maximă.

#### Date de intrare

Pe prima linie a fișierului de intrare **matrice.in** se află numerele  $N$  și  $M$ , separate prin spațiu.

Pe fiecare dintre următoarele  $N$  linii se află câte  $M$  numere întregi separate prin câte un spațiu, reprezentând elementele matricei  $A$ .

#### Date de ieșire

Fișierul de ieșire **matrice.out** va conține o singură linie pe care vor fi scrise coordonatele  $l_1, c_1, l_2, c_2$  (în această ordine și separate prin câte un spațiu) ale unei submatrice valide de arie maximă. În cazul în care există mai multe soluții cu arie maximă, se va afișa oricare dintre ele.

#### Restricții și precizări

- $1 \leq N, M \leq 1000$
- 70% din teste vor avea  $N, M \leq 600$
- Elementele matricei  $A$  sunt numere întregi din intervalul  $[-30000, 30000]$ .

#### Exemplu

matrice.in	matrice.out	Explicatii
2 6	1 1 2 3	Aria maximă este 6.
1 2 5 7 9 10		O altă soluție de arie maximă ar putea fi
3 4 3 5 1 10		1 1 1 6 sau 1 2 2 4 sau 1 3 2 5 sau 1 4 2 6

**Timp maxim de execuție/test:** 1 secundă

#### 13.3.1 Indicații de rezolvare - descriere soluție \*

*Soluția oficială*

Se parcurg în ordine coloanele matricei. Pentru o coloană  $j$  se calculează pentru fiecare linie  $i$  o valoare  $Left[i][j]$  ca fiind lungimea maximă a unui șir valid de pe linia  $i$  care are ultimul element pe coloana  $j$ .  $Left[i][j]$  se poate determina din  $Left[i][j-1]$  memorând ultima poziție (coloană) de pe fiecare linie unde șirul și-a schimbat monotonia (deoarce două elemente consecutive de pe o linie sunt distincte, șirurile nu vor putea fi decât strict crescătoare/descrescătoare). Ca o observație, este suficient să memorăm doar ultima coloană a acestei matrice  $Left$ .

Problema revine la a afla submatricea de arie maximă care are coloana din dreapta  $j$ , știind pentru fiecare linie lungimea maximă a unui șir valid de pe linia  $i$  care se termină pe coloana  $j$  ( $Left[i][j]$ ).

Acest lucru se poate face în  $O(N)$  pentru fiecare coloană, ținând o stivă de elemente crescătoare. De exemplu, pentru o matrice cu 4 linii considerăm că pentru o coloană  $j$  am calculat următoarele valori  $Left[i][j] : 3, 4, 1, 2$ . Introducem 3 și 4 în stivă, iar când introducem 1 va trebui să scoatem din stivă pe 4 și apoi pe 3. De fiecare dată când scoatem un număr din stivă vedem ce dreptunghiuri pot apărea care să îl conțină.

Această soluție are complexitatea  $O(M * N)$  și obține 100 puncte.

O soluție care pentru fiecare coloană, având calculate valorile  $Left[i][j]$ , rezolvă problema în  $O(N^2)$  (deci are complexitate totală  $O(M * N^2)$ ) va obține 70 puncte.

O soluție care pentru fiecare coloană calculează de fiecare dată valorile  $Left[i][j]$  (mergând în stânga atât cât este nevoie), deci având complexitatea totală  $O(N^2 * M^2)$ , va obține 40 puncte.

### 13.3.2 Rezolvare detaliată

### 13.3.3 Codul sursă

## 13.4 Petrom - ONI 2006

Firma Petrom are  $n$  benzinării amplasate de-a lungul autostrăzii Soarelui. Benzinăriile sunt numerotate de la 1 la  $n$  în ordinea amplasării pe autostradă.

Pozițiile benzinăriilor sunt cunoscute, fiind specificate prin distanțele  $d_1, d_2, \dots, d_n$  ( $d_i$  reprezintă distanța de la sediul firmei Petrom până la benzinăria  $i$ ).

Sediul firmei Petrom este amplasat la intrarea pe autostrada Soarelui.

În  $k$  dintre aceste benzinării firma dorește să amenajeze depozite de combustibil, care să alimenteze benzinăria respectivă, dar și alte benzinării învecinate. Fiecare benzinărie va fi alimentată de la depozitul cel mai apropiat.

Costul de transport pentru o amplasare a depozitelor dată va fi egal cu suma distanțelor de la fiecare benzinărie la depozitul de la care se alimentează.

#### Cerință

Scrieți un program care să determine benzinăriile în care trebuie să construite depozite, astfel încât costul de transport să fie minim.

#### Date de intrare

Fișierul de intrare **petrom.in** conține pe prima linie două numere naturale separate prin spațiu  $n$  și  $k$ , reprezentând numărul de benzinării și respectiv numărul de depozite care trebuie construite.

Pe următoarele  $n$  linii sunt scrise  $n$  numere naturale  $d_1, d_2, \dots, d_n$ , câte un număr pe o linie, reprezentând distanțele de la sediul Petrom la cele  $n$  benzinării.

#### Date de ieșire

Fișierul de ieșire **petrom.out** va conține pe prima linie costul de transport (minim posibil).

Pe următoarele  $k$  linii sunt scrise benzinăriile în care vor fi construite depozite pentru a obține costul minim, câte o benzinărie pe o linie.

#### Restricții și precizări

$$1 \leq n \leq 400$$

$$1 \leq k \leq 300$$

$$k \leq n$$

$$0 < d_1 < d_2 < \dots < d_n \leq 30000$$

Dacă există mai multe soluții puteți determina oricare dintre acestea.

Pentru fiecare test, se acordă 40% din punctaj pentru determinarea costului minim și 100% pentru rezolvarea integrală.

#### Exemplu

petrom.in	petrom.out	Explicatii
6 3	8	Depozitul 1 va fi construit în benzinăria 2 și alimentează benzinăriile 1, 2, 3.
5	2	(Cost: 1+0+6=7)
6	4	Depozitul 2 va fi construit în benzinăria 4 și alimentează benzinăriile 4, 5. (Cost: 0+1=1)
12	6	Depozitul 3 va fi construit în benzinăria 6 și alimentează benzinăriile 6. (Cost: 0)
19		
20		
27		

Timp maxim de execuție/test: 0.2

### 13.4.1 Indicații de rezolvare - descriere soluție \*

#### Soluția oficială

Vom precalcula mai întâi o matrice cost:

$cost[a][b]$  = costul de transport pentru cazul în care un depozit alimentează benzinăriile din intervalul  $[a, b]$ ,  $a < b$

$$cost[a][a] = 0$$

Costul optim se obține amplasând un depozit în benzinăria situată în mijlocul intervalului.

Soluția problemei se obține prin programare dinamică.

Subprobleme:

Să se determine o amplasare optimă a  $i$  depozite în benzinăriile  $1, \dots, j$

Vom memora soluțiile subproblemelor în matricea  $cm_{in}$ :

$cm_{in}[i][j]$  = costul minim de transport în condițiile în care construim  $i$  depozite în benzinăriile  $1, \dots, j$

Relația de recurență:

$cm_{in}[1][j] = cost[1][j]$ , pentru  $j = 1, \dots, n$

$cm_{in}[i][j] = \min\{cm_{in}[i-1][p-1] + cost[p][j] | p = i, i+1, \dots, j\}$

Pentru reconstituirea soluției vom utiliza o matrice  $back$

$back[i][j]$  = poziția  $p$  pentru care se obține costul minim  $cm_{in}[i][j]$

### 13.4.2 Rezolvare detaliată

### 13.4.3 Codul sursă

## 13.5 ratina - ONI 2006

Limba ratină are doar  $N$  cuvinte, numerotate de la 1 la  $N$ . Două sau mai multe cuvinte se numesc  $k$ -asemenea dacă au primele  $k$  litere identice.

Gradul de asemănare între  $t$  cuvinte este  $k$  dacă cele  $t$  cuvinte sunt  $k$ -asemenea, dar nu sunt  $(k+1)$ -asemenea.

#### Cerință

Scrieți un program care pentru un set de  $t$  cuvinte dat, răspunde la interogări de genul: "Care este gradul de asemănare între cuvintele  $x_1x_2\dots x_t$ " ?

#### Date de intrare

Fișierul de intrare **ratina.in** va conține pe prima linie două numere naturale  $N$   $M$ , separate printr-un spațiu, reprezentând numărul de cuvinte din limba ratină, respectiv numărul de interogări.

Următoarele  $N$  linii vor conține cuvintele din limba ratină, câte un cuvânt pe o linie. Mai exact, pe linia  $i+1$  este scris cuvântul cu numărul de ordine  $i$ . Cuvintele sunt formate din litere mici din alfabetul englez.

Urmează  $M$  linii, fiecare linie reprezentând câte o interogare exprimată astfel: primul număr de pe linie este un număr natural  $t$  cuprins între 2 și 10 reprezentând numărul de cuvinte din interogare, apoi vor urma cele  $t$  numere de ordine ale cuvintelor din interogare, separate prin câte un spațiu.

#### Date de ieșire



Fișierul de ieșire **ratina.out** va conține  $M$  linii, câte una pentru fiecare întrebare din fișierul de intrare. Pe linia  $i$  va fi scris gradul de asemănare al cuvintelor din întrebarea  $i$ .

#### Restricții și precizări

$0 < N < 10001$

$0 < \text{lungimea maximă a unui cuvânt} < 2000$

$0 < \text{suma lungimilor tuturor cuvintelor} < 200000$

$1 < M < 100001$

Pentru teste cu  $N \leq 200$ , se acordă 50 de puncte din care 30 de puncte pentru teste și cu  $M \leq 10000$ .

#### Exemplu

ratina.in	ratina.out	Explicații
6 5	3	Prima întrebare cere gradul de asemănare între
asdf	2	cuvintele asdf și asdefff, care este 3 (deoarece
asdefff	3	cele două cuvinte au primele 3 litere identice,
gata	0	dar nu și primele 4 litere).
gara	4	Cea de a doua întrebare cere gradul de asemănare
pesistem		între cuvintele gata și gara, care este 2.
pestesistem		Cea de a treia întrebare cere gradul de asemănare
2 1 2		între cuvintele pesistem și pestesistem care este 3.
2 3 4		Cea de a patra întrebare cere gradul de asemănare
2 5 6		între cuvintele asdf, gata și pesistem care este 0.
3 1 3 5		Ultima întrebare este evidentă: un cuvânt de
2 1 1		lungime $k$ este $k$ -asemenea cu el însuși.

**Temp maxim de execuție/test:** 1 secundă

### 13.5.1 Indicații de rezolvare - descriere soluție \*

#### Soluția oficială

Problema are mai multe soluții.

Dacă răspundem la fiecare întrebare parcurgând cuvintele pe toată lungimea lor până la găsirea unei diferențe (soluție de complexitate  $O(M \cdot L \cdot C)$  ( $L$ =lungimea maximă,  $C$  numărul de cuvinte din întrebare)) obținem 30 de puncte.

Dacă preprocesăm pentru oricare 2 cuvinte gradul de asemănare, obținem 50 de puncte.

Pentru a obține punctajul maxim este necesar să răspundem mai repede la o astfel de întrebare.

Soluția comisiei folosește un arbore de prefixe pentru a reține dicționarul și totodată pentru fiecare caracter din cuvânt ce nod din arbore îi corespunde. Având aceste date preprocesate putem răspunde la o întrebare în  $O(C \cdot \log L)$  folosind căutarea binară. Mai multe cuvinte au prefixul de lungime  $k$  în cazul în care cuvintelor le corespunde același nod din arbore pentru acea poziție.

Un arbore de prefixe este un arbore în care fiecare nod are  $S$  fii unde  $S$  este dimesiunea alfabetului, în cazul nostru 26. Nodul rădăcină este special, îl putem nota cu caracterul #, acesta are 26 de fii notați cu 'a','b', ..., 'z' acești fii având la rândul lor alți 26 de fii. Când introducem un cuvânt în arbore începem cu rădăcina și mergem în fiu care corespunde primului caracter din acesta în fiul care corespunde celui de-al doilea caracter s.a.m.d în acest mod găsim ce nod în arbore îi corespunde unei anumite poziții dintr-un cuvânt.

### 13.5.2 Rezolvare detaliată

### 13.5.3 Codul sursă

## 13.6 vitale - ONI 2006

În orașul Etsivograt există  $n$  intersecții numerotate de la 1 la  $n$ . Între unele perechi de intersecții există străzi. În total sunt  $m$  străzi, iar rețeaua stradală formată din acestea a fost concepută astfel încât între oricare două intersecții există cel puțin o legătură care poate fi directă sau poate trece prin alte intersecții.

După trecerea anotimpului rece, starea străzilor a devenit deplorabilă, așa că se decide asphaltarea acestora. Asphaltarea fiecărei străzi are un cost cunoscut.

Având resurse limitate, primarul hotărăște să asfalteze câteva străzi care să asigure cel puțin o legătură (directă sau indirectă, adică trecând prin alte intersecții) între oricare două intersecții, iar suma costurilor asfaltării acestor câteva străzi să fie minimă. Consilierii lui îi prezintă lista tuturor posibilităților de asphaltare ce asigură legături (directe sau indirecte) între toate intersecțiile și pentru care suma costurilor este minimă. Primarul observă că există străzi care apar în toate aceste posibilități de asphaltare. El consideră aceste străzi ca fiind "vitale".

#### Cerință

Scrieți un program care determină numărul de străzi vitale care se află în rețeaua stradală a orașului Etsivograt, precum și care sunt aceste străzi vitale.

#### Date de intrare

Fișierul de intrare **vitale.in** conține pe prima linie două numere naturale  $n$   $m$  separate printr-un spațiu reprezentând numărul de intersecții, respectiv numărul de străzi din oraș. Pe următoarele  $m$  linii se află câte trei numere naturale  $a$   $b$   $c$  separate prin câte un spațiu;  $a$  și  $b$  reprezintă numerele de ordine ale două intersecții distincte din oraș între care există o stradă, iar  $c$  reprezintă costul asfaltării străzii care unește intersecțiile  $a$  și  $b$ .

#### Date de ieșire

Fișierul de ieșire **vitale.out** va conține pe prima linie un număr natural  $x$ , reprezentând numărul de străzi vitale din oraș. Pe fiecare dintre următoarele  $x$  linii, vor fi scrise câte două numere naturale reprezentând numerele de ordine ale intersecțiilor care delimitează câte o stradă vitală iar primul număr va fi strict mai mic decât al doilea. Aceste  $x$  linii vor fi sortate în ordine lexicografică, cu alte cuvinte notând cu  $a_i$  și  $b_i$  cele două numere de pe linia  $i + 1$  și cu  $a_j$  și  $b_j$  cele două numere de pe linia  $j + 1$ , dacă  $i < j$  atunci  $a_i < a_j$  sau  $a_i = a_j$  și  $b_i < b_j$ .

#### Restricții și precizări

$$1 \leq n \leq 50000$$

$$1 \leq m \leq 100000$$

$$1 \leq c \leq 1000000000$$

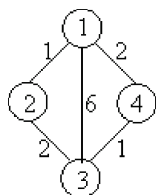
Între oricare două intersecții poate exista cel mult o stradă. Străzile sunt bidirecționale.

#### Exemplu

vitale.in	vitale.out
4 5	2
1 2 1	1 2
2 3 2	3 4
4 3 1	
1 4 2	
1 3 6	

#### Explicații

Intersecțiile și străzile din exemplu corespund configurației din figură.



Posibilitățile de asfaltare care corespund cerințelor sunt formate din străzile: 1-2, 1-4, 3-4 și 1-2, 2-3, 3-4. Sunt două muchii vitale (care apar în ambele posibilități) și anume 1-2 și 3-4.

**Timp maxim de execuție/test:** 1.5 secunde

### 13.6.1 Indicații de rezolvare - descriere soluție \*

*Soluția oficială*

Există mai multe soluții de rezolvare în complexitate  $O(m \log m)$ .

Dăm una dintre ele.

Ideea pornește de la algoritmul lui Kruskal pentru arbore de cost minim care adaugă muchii în graf în ordinea crescătoare a costurilor.

Pentru toate muchiile de cost fixat  $x$  ne uităm la componentele conexe ale grafului ce conține muchiile de cost mai mic decât  $x$ , acest graf va conține niște componente conexe. Putem să contractăm acele componente conexe în noduri și să ignorăm muchiile de cost mai mare sau egal cu  $x$  care evident nu sunt muchii vitale.

Acum observăm că muchiile vitale de cost  $x$  sunt muchiile critice în graful care conține nodurile menționate anterior și muchiile de cost  $x$ .

Contractarea nodurilor în componente conexe se realizează folosind structuri de mulțimi disjuncte, iar determinarea muchiilor critice se face cu binecunoscutul algoritmul pentru aflarea componentelor biconexe de complexitate  $O(m)$ . Acest algoritmul are complexitate  $O(m \log m)$  în faza de sortare, fazele de contracție ale grafului vor avea complexitate totală maxim  $O(m \log * n)$  folosind structuri de mulțimi disjuncte, iar fazele de determinare ale muchiilor critice durează  $O(m)$  în total, pentru că la pasul de cost  $x$  determinarea componentelor biconexe are complexitate  $O(mx)$  unde  $mx$  este numărul de muchii de cost  $x$ , cum în total numărul de muchii este  $m$ , obținem complexitatea  $O(m)$ . Astfel complexitatea finală este  $O(m \log m)$ .

Algoritmi neoptimi la care ne-am gândit au fost generarea unui arbore parțial de cost minim, iar apoi eliminarea câte unei muchii din graf după care determinăm arborele de cost minim pe graful obținut. Dacă acest arbore are cost diferit de cel original, evident muchia eliminată a fost una critică. Acest algoritmul are complexitatea  $O(n * m)$  pentru că avem  $n - 1$  muchii într-un arbore parțial, iar după ce muchiile grafului sunt sortate algoritmul Kruskal are complexitate  $O(m)$ . Acesta ar fi luat în jur de 30-40 de puncte.

Alt algoritmul ar fi determinarea randomizată a mai multor arbori parțiali și returnarea muchiilor comune tuturor arborilor. Acesta ar fi luat în jur de 40 - 50 de puncte.

### 13.6.2 Rezolvare detaliată

### 13.6.3 Codul sursă

## Capitolul 14

# ONI 2007 clasa a XI-a



### 14.1 Descompunere - ONI 2007

Considerăm trei numere naturale nenule:  $n$ ,  $k$  și  $x$ .

Denumim o  $kx$ -descompunere a numărului  $n$  o posibilitate de a scrie numărul  $n$  ca sumă de  $k$  numere naturale nenule astfel încât diferența între oricare doi termeni ai sumei este cel puțin egală cu  $x$ .

#### **Cerință**

Fiind date trei numere naturale  $n$ ,  $k$  și  $x$ , să se determine câte  $kx$ -descompuneri distincte există. Două  $kx$ -descompuneri sunt distincte dacă diferă prin cel puțin un termen.

#### **Date de intrare**

Fișierul **desc.in** conține pe prima linie trei valori naturale nenule reprezentând numerele  $n$ ,  $k$  și  $x$ .

#### **Date de ieșire**

Fișierul **desc.out** va conține o singură valoare reprezentând restul împărțirii numărului de  $kx$ -descompuneri distincte la numărul 10007.

#### Restricții și precizări

- pentru 20% din teste  $0 < n \leq 200$ ;
- pentru celelalte 80% din teste,  $200 < n \leq 10000$ ;
- $1 \leq x, k \leq n$

#### Exemplu

desc.in	desc.out
20 2 3	8

#### Explicații

Numărul de  $kx$ -descompuneri în acest caz este 8. Acestea sunt formate din numerele 1 și 19; 2 și 18; 3 și 17; 4 și 16; 5 și 15; 6 și 14; 7 și 13; 8 și 12

desc.in	desc.out
2000 19 7	3184

**Limită de memorie:** 32 Mb, din care 1 Mb pentru stivă.

**Timp maxim de execuție/test:** 0.1 secunde

### 14.1.1 Indicații de rezolvare - descriere soluție \*

*Stelian Ciurea*

Fie o  $kx$ -descompunere, adică

$$N = a_1 + a_2 + \dots + a_k$$

În aceasta vom presupune că

$$a_1 < a_2 < a_3 < \dots < a_k$$

sau

$$a_1 \leq a'_2 + x \leq a'_3 + x \leq \dots \leq a'_k + (k-1)x$$

în care  $a_1 \leq a'_2 \leq a'_3 \leq \dots \leq a'_k$

Dacă o rescriem, rezultă

$$N = a_1 + a'_2 + a'_3 + \dots + a'_k + x + 2x + \dots + (k-1)x$$

adică

$$N - x(1 + 2 + \dots + (k-1)) = a_1 + a'_2 + a'_3 + \dots + a'_k$$

ceea ce reprezintă o posibilitate de a scrie numărul  $N - x * k(k-1)/2$  ca sumă de  $k$  numere naturale care pot fi și egale.

Aceste transformări sunt bijective, adică oricărei  $kx$ -descompuneri a numărului  $N$  îi corespunde o descompunere în  $k$  numere care pot fi și egale a numărului  $N - x * k(k - 1)/2$ .

Numărul de posibilități de a scrie un număr oarecare - să îl notăm  $A$  - ca o sumă de  $k$  numere naturale nenule care pot fi și egale este dat de o formulă cunoscută (care apare și în unele manuale mai vechi de clasa a X-a) denumită formula Stirling:

$$S(A, k) = S(A - k, 1) + S(A - k, 2) + \dots + S(A - k, k)$$

Aplicând această formulă pentru  $A = N - x * k(k - 1)/2$  obținem rezultatul cerut.

### 14.1.2 Rezolvare detaliată

$$P(n + k, k) = P(n, 1) + P(n, 2) + \dots + P(n, k)$$

unde  $P(n, 1) = P(n, n) = 1$  și  $n = a_1 + a_2 + \dots + a_k$  și  $a_1 \geq a_2 \geq \dots \geq a_k \geq 1$ .

### 14.1.3 Codul sursă

## 14.2 Felinare - ONI 2007

A venit timpul Anarhiei în Orașul Trist! Ca revoltă împotriva manifestărilor subculturale, vrei să pui Orașul pe butuci.

În urma unei descinderi ilegale la Primărie, ai "împrumutat" o hartă și ți-ai dat seama că există  $M$  străzi unidireționale între cele  $N$  intersecții ale Orașului Trist.

Fiecare intersecție are câte două felinare. Primul luminează o jumătate din fiecare stradă care pleacă din intersecția respectivă, iar al doilea luminează jumătate din fiecare stradă care intră în intersecție. De exemplu, prima jumătate a străzii dintre intersecțiile  $A$  și  $B$  este luminată de primul felinar din intersecția  $A$ , iar cea de-a doua jumătate este luminată de al doilea felinar din intersecția  $B$ .

Un felinar stins nu luminează deloc. O stradă este sigură doar atunci când e luminată în totalitate.

### Cerință

În primul rând, trebuie să te asiguri că nicio stradă nu va fi complet luminată, astfel încât siguranța cetățenilor să fie redusă la minim. Dar acest obiectiv nu te mulțumește, așa că în plus îți dorești un număr maxim de felinare aprinse, pentru

a da o grea lovitură bugetului Primăriei din Orașul Trist. Odată îndeplinite aceste condiții, Revoluția poate începe.

#### Date de intrare

Fișierul de intrare **felinare.in** conține pe prima linie două numere naturale strict pozitive  $N$  și  $M$ , reprezentând numărul de intersecții și numărul de străzi din Orașul Trist. Pe fiecare din următoarele linii se află o pereche de numere naturale  $A$  și  $B$  cuprinse între 1 și  $N$  reprezentând o stradă care pleacă din intersecția  $A$  și ajunge în intersecția  $B$ .

#### Date de ieșire

Fișierul de ieșire **felinare.out** va conține pe prima linie un singur număr natural reprezentând numărul maxim de felinare ce pot fi aprinse. Pe următoarele  $N$  linii se vor afla numere cuprinse între 0 și 3, cu semnificația următoare:

- 0 : ambele felinare din intersecție sunt stinse;
- 1 : numai primul dintre felinarele din intersecție este aprins;
- 2 : numai al doilea dintre felinarele din intersecție este aprins;
- 3 : ambele felinare din intersecție sunt aprinse.

#### Restricții și precizări

$$1 \leq N \leq 8191$$

$$1 \leq M \leq 20000$$

Nu există străzi care să unească o intersecție cu ea însăși.

Pentru determinarea numărului maxim de felinare se acordă 40% din punctaj.

#### Exemplu

felinare.in	felinare.out
4 4	6
1 2	2
4 1	3
4 2	3
4 3	2

**Limită de memorie:** 32 Mb, din care 1 Mb pentru stivă.

**Timp maxim de execuție/test:** 0.1 secunde

### 14.2.1 Indicații de rezolvare - descriere soluție \*

*Tiberiu Florea*

Pornind de la graful inițial, construim un nou graf bipartit, după cum urmează: în stânga  $N$  noduri corespunzătoare felinarelor de tipul 1, iar în dreapta  $N$  noduri corespunzătoare felinarelor de tipul 2.

O muchie  $i \rightarrow - > j$  în graful inițial devine o muchie între nodul  $i$  din stânga și nodul  $j$  din dreapta în noul graf bipartit.



Se observă că o stradă  $i \rightarrow - > j$  este complet iluminată  $i \rightarrow - > j$  dacă și numai dacă atât felinarul corespunzător nodului  $i$  din stânga cât și felinarul corespunzător nodului  $j$  din dreapta sunt aprinse.

În concluzie, având dat un graf bipartit vrem să alegem un număr maxim de noduri astfel încât să nu existe nici o muchie cu ambele noduri adiacente alese.

Pentru a rezolva problema, avem nevoie mai întâi de câteva definiții.

Într-un graf bipartit, un "suport minim" reprezintă o mulțime de noduri cu cardinal minim pentru care orice muchie a grafului este adiacentă cu cel puțin unul dintre nodurile mulțimii.

Unul dintre cele mai importante rezultate de teoria grafurilor, Teorema lui König, spune că într-un graf bipartit cuplajul maxim și suportul minim sunt egale. Pe baza acestei teoreme, un suport minim se poate determina relativ ușor pornind de la orice cuplaj maxim. O "mulțime independentă maximală" (ceea ce ne cere, de fapt, problema), este complementul oricărui suport minim.

Suportul minim se poate calcula în modul următor:

```
// presupunem ca a fost calculat deja un cuplaj maxim
// S este suportul minim, initial multime vida
// C(j-dreapta) e nodul i-stanga cu care e cuplat j-dreapta (daca exista)

procedura calculeaza(i-stanga) // doar pt. noduri care nu sunt in suport
  pentru j-dreapta vecin al lui i-stanga
    daca j-dreapta nu e in S
      S <- S+{j-dreapta}
      S <- S-{ C(j-dreapta) } // exista mereu C(j-dreapta)
      calculeaza( C(j-dreapta) )

pentru fiecare i din stanga
  daca i e cuplat
    S <- S+{i}

pentru fiecare i din stanga
  daca i nu e cuplat
    calculeaza(i)
```

Pentru 40 de puncte este suficient să calculăm un cuplaj maxim și soluția egală cu  $2 \cdot N$ -valoarea cuplajului. Pentru celelalte 60 de puncte trebuie să determinăm un suport minim și apoi o mulțime independentă maximală pornind de la acest cuplaj. Complexitatea totală a algoritmului este  $O(N \cdot M)$ .

### 14.2.2 Rezolvare detaliată

### 14.2.3 Codul sursă

## 14.3 Joc - ONI 2007

Gică și Petrică locuiesc în Orașul Vesel. Acolo există o rețea stradală care conține doar străzi cu sens unic iar pentru a te deplasa între două intersecții trebuie să plătești o taxă specifică fiecărei intersecții și fiecărei străzi prin care treci (inclusiv intersecția din care pleci și intersecția în care ajungi). Aceste taxe nu deranjează însă pe nimeni, deoarece sunt valori monetare naturale între 0 și 10. Pe cei doi nu îi interesează harta propriu-zisă a Orașului, însă au găsit tabelul sumelor minime pe care trebuie să le plătească pentru a se plimba între oricare două intersecții și s-au gândit să-l folosească pentru a juca un joc interesant (Această matrice conține numai valori finite).

Astfel, având matricea  $A$  în care  $A[i, j]$  reprezintă costul minim pentru a ajunge din intersecția  $i$  în intersecția  $j$  ( $A[i, i]$  este taxa care trebuie plătită în nodul  $i$ ), fiecare dintre cei doi mută alternativ, după cum urmează: jucătorul aflat la mutare își alege un număr natural strict pozitiv  $k$ , și scade  $k$  din toate elementele unei linii sau coloane ale matricei, cu condiția ca toate elementele matricei să rămână nenegative. Gică mută primul, iar jucătorul care, atunci când îi vine rândul, nu mai poate efectua o mutare corectă, pierde. Se consideră că cei doi prieteni joacă perfect, însemnând că dacă unul dintre ei are la un moment dat o strategie de câștig indiferent de mutările adversarului, nu va efectua o mutare care să ducă la pierderea oricărei strategii de câștig.

### Cerință

Dându-se matricea  $A$  cu semnificația din enunț, aflați care dintre cei doi este câștigătorul jocului.

### Date de intrare

Fișierul **joc.in** conține între 1 și 20 de seturi de date de intrare. Pe prima linie a fiecărui set se află un număr natural pozitiv  $N$  reprezentând dimensiunile matricei  $A$ , iar pe următoarele  $N$  linii se află câte  $N$  numere naturale, reprezentând elementele matricei  $A$ .  $N = 0$  marchează sfârșitul seturilor de date ce compun fișierul de intrare.

### Date de ieșire

Fișierul de ieșire **joc.out** va conține un număr de linii egal cu numărul de seturi de date. Pe fiecare dintre aceste linii se află una dintre valorile: 1 în cazul în care jocul respectiv este câștigat de Gică, 2 în cazul în care jocul respectiv este câștigat de Petrică.

### Restricții și precizări

$1 \leq \text{numărul testelor} \leq 20$

$2 \leq N \leq 100$

Rezolvarea corectă a testelor ce conțin cel mult 10 seturi de date cu  $N \leq 5$  garantează obținerea a 50% din punctaj.

#### Exemplu

joc.in	joc.out
2	1
8 25	2
25 9	
2	
3 8	
9 3	
0	

**Limită de memorie:** 32 Mb, din care 1 Mb pentru stivă.

**Timp maxim de execuție/test:** 0.1 secunde

### 14.3.1 Indicații de rezolvare - descriere soluție \*

*Tiberiu Florea*

Se pot obține 50 de puncte cu diverse soluții exponențiale (bazate, eventual, pe arbori de joc), fără să ținem cont de modul în care este obținută matricea.

Pentru punctajul maxim, facem următoarea observație: în graful inițial (care nu este citit), pentru a ajunge de la nodul  $i$  la nodul  $j$  trebuie să plătim cel puțin taxa din nodul  $i$  și taxa din nodul  $j$ . Astfel, fiecare matrice  $A$  respectă condiția  $A[i, j] \geq A[i, i] + A[j, j]$  pentru orice  $i$  diferit de  $j$ .

Această proprietate se păstrează și în urma operațiilor permise, de scădere a unui număr de pe o linie sau coloană. Un jucător pierde atunci când pe fiecare linie și pe fiecare coloană există cel puțin câte un zero și, ținând cont de proprietatea precedentă, primele elemente ale matricii care devin nule sunt elementele diagonalei principale.

Indiferent dacă la un anumit pas scădem  $k$  de pe o linie sau de pe o coloană, un singur element de pe diagonala principală va scădea cu  $k$ . Se observă că jocul este echivalent cu NIM în varianta clasică pe elementele diagonalei principale. Complexitate:  $O(N^2)$ .

### 14.3.2 Rezolvare detaliată

### 14.3.3 Codul sursă

## 14.4 Logaritmi - ONI 2007

Fie expresia:

$$\log_{a_1} b_1 * \log_{a_2} b_2 * \dots * \log_{a_n} b_n$$

Un calculator trebuie să evalueze această expresie aducând-o la forma unui singur număr real. Pentru aceasta, el poate face următoarele calcule:

- Produs = produsul a doua numere reale în  $t_1$  unități de timp;
- Reducere = înlocuirea expresiei  $\log_a b * \log_b c$  cu  $\log_a c$  în  $t_2$  unități de timp;
- Calcul = calculul unui logaritm, rezultatul fiind un număr real; pentru a calcula  $\log_a b$  îi sunt necesare  $t_3 * (a - b)^2$  unități de timp.

### Cerință

Să se determine timpul minim pentru a calcula o expresie dată.

### Date de intrare

Fișierul **log.in** conține:

- pe prima linie o valoare numerică naturală  $n$  cu semnificația din enunț;
- pe a doua linie trei valori numerice naturale  $t_1 t_2 t_3$  separate prin câte un spațiu, cu semnificația din enunț;
- pe fiecare din următoarele  $n$  linii câte două valori numerice naturale  $a_i b_i$  cu semnificațiile din enunț.

### Date de ieșire

Fișierul **log.out** va conține o singură valoare reprezentând numărul de unități de timp necesare evaluării expresiei.

### Restricții și precizări

Pentru 70% din teste  $0 < n \leq 500$ ; pentru celelalte 30% din teste  $n \leq 10000$ ;  
 $1 < a_i, b_i < 100$   $1 \leq t_1, t_2, t_3 \leq 100$

Factorii expresiei inițiale sau ai oricăreia dintre expresiile rezultate pe parcursul evaluării NU pot fi comutați între ei.

### Exemplu

log.in	log.out
3	13
2 1 3	
2 3	
3 4	
4 5	

**Explicații:**

Se calculează fiecare din cei trei logaritmi, rezultă trei numere, fiecare calcul necesită 3 unități de timp; se înmulțesc primele două numere în 2 unități de timp, apoi rezultatul se înmulțește cu al treilea număr tot în 2 unități; în total:  $3 + 3 + 3 + 2 + 2 = 13$  unități.

**Exemplu**

log.in	log.out
4	9
2 1 2	
2 2	
3 4	
4 4	
4 5	

**Explicații:**

Primul logaritm se calculează în 0 unități; al doilea și al treilea se reduc la un logaritm în 1 unitate iar acest logaritm se calculează în 2 unități; al patrulea se calculează în 2 unități; au rezultat trei numere, care pot fi aduse la unul singur prin două înmulțiri, fiind necesare  $1 + 2 + 2 + 2 + 2 = 9$  unități de timp.

**Limită de memorie:** 32 Mb, din care 1 Mb pentru stivă.

**Timp maxim de execuție/test:** 1 secundă

**14.4.1 Indicații de rezolvare - descriere soluție \***

*Stelian Ciurea*

**Soluția 1** (Stelian Ciurea)

Pentru 70 puncte, problema se poate rezolva în complexitate  $O(n^3)$  folosind un algoritm asemănător cu cel pentru înmulțirea optimă a unui șir de matrice. Astfel se calculează o matrice în care fiecare element  $o[i, j, 0]$  reține timpul minim necesar pentru a calcula expresia formată din logaritmi și a o aduce la forma unui logaritm, iar  $o[i, j, 1]$  reține timpul minim necesar pentru a calcula expresia formată din logaritmi și a o aduce la forma unui număr real.

Soluția se găsește în  $o[1][n][1]$ .

**Soluția 2** (Tiberiu Florea, Daniel Pășăilă)

Pentru 100 puncte era necesară o soluție în  $O(n^2)$ . Pentru asta trebuie să folosim un tablou  $V$ , unde  $V[i]$  reprezintă costul minim pentru a calcula primii  $i$  logaritmi.  $V[i]$  se calculează ușor în timp liniar, considerând toate secvențele posibile din care ar putea face parte logaritmii  $i$  (înainte de a fi transformat într-un real). Relațiile de recurență se deduc ușor.

### 14.4.2 Rezolvare detaliată

### 14.4.3 Codul sursă

## 14.5 Maxq - ONI 2007

Johnie a început să se joace cu un vector de numere. El dispune inițial de un vector  $V$  cu  $N$  numere întregi (numerotate de la 0 la  $N - 1$ ) și poate efectua următoarele operații:

- schimbarea elementului de pe poziția  $p$  cu un alt număr întreg;
- aflarea subsecvenței de sumă maximă din  $V$  inclusă între indicii  $a$  și  $b$ ;

#### Cerință

Ajutați-l pe Johnie să efectueze repede operațiile de mai sus.

#### Date de intrare

Fisierul **maxq.in** conține pe prima linie numărul  $N$  reprezentând dimensiunea vectorului.

Pe următoarea linie se găsesc  $N$  elemente reprezentând valorile inițiale ale vectorului. Următoarea linie conține  $M$ , reprezentând numărul de operații. Pe fiecare dintre următoarele  $M$  linii sunt descrise cele  $M$  operații în forma următoare:

- $0 \ i \ p$ : numărul 0 de la început codifică faptul că operația curentă este una de schimbare; astfel elementul de pe poziția  $i$  a vectorului se înlocuiește cu numărul întreg  $p$ ;
- $1 \ a \ b$ : numărul 1 de la început codifică faptul că operația curentă este o întrebare; astfel se cere să se afle subsecvența de sumă maximă din vector inclusă între indicii  $a$  și  $b$  ( $a \leq b$ );

#### Date de ieșire

Fisierul **maxq.out** trebuie să conțină un număr de linii egal cu numărul de întrebări din fișierul de intrare. Pe linia  $i$  se cere să se afișeze un singur număr reprezentând suma maximă ce se poate obține în contextul întrebării  $i$  din fișierul de intrare ( $i = 1, 2, \dots$ ); în cazul în care vor exista doar subsecvențe de sumă negativă se va afișa 0.

#### Restricții și precizări

- $1 \leq N \leq 200000$ ;
- $1 \leq M \leq 200000$ ;
- toate elementele vectorului apar în intervalului  $[-100000, 100000]$ ;

- definim o subsecvență ca fiind un șir de termeni consecutivi din vector, iar suma ei se obține adunând elementele ce o compun;
- există cel puțin o întrebare.
- pentru 20% din teste se garantează  $N \leq 5000$

**Exemplu**

maxq.in	maxq.out
5	4
1 10 4 1 9	16
4	12
1 0 3	
0 1 1	
1 0 3	
1 2 4	

**Explicație**

Pentru prima întrebare se alege subsecvența formată de elementul pe poziția 2 din vector. Pentru a 2-a întrebare se aleg primele 3 elemente din vector (elementul de pe poziția 1 a fost schimbat). Pentru a 3-a întrebare se aleg toate elementele din intervalul cerut.

**Limită de memorie:** 32 Mb, din care 1 Mb pentru stivă.

**Timp maxim de execuție/test:** 1.2 secunde

**14.5.1 Indicații de rezolvare - descriere soluție \***

*Daniel Păsăilă*

Problema se poate rezolva în mai multe moduri. Prima posibilitate ar fi folosirea unui arbore de intervale. Astfel, în fiecare nod al arborelui se țin următoarele valori:

- $A[\text{nod}]$  - valoarea subsecvenței de sumă maximă situată la începutul intervalului curent
- $B[\text{nod}]$  - valoarea subsecvenței de sumă maximă situată la sfârșitul intervalului curent
- $C[\text{nod}]$  - valoarea subsecvenței de sumă maximă situată oriunde în intervalul curent
- $D[\text{nod}]$  - suma elementelor din intervalul curent

Astfel, fiecare update se poate efectua în  $O(\log N)$  (se apelează procedura pentru fiii nodului curent, apoi se calculează valorile pentru intervalul curent). Query-ul se efectuează parcurgând recursiv intervalele consecutive din arbore ce sunt incluse în intervalul curent (complexitate  $O(\log N)$ ). Soluția aceasta obține 100 puncte.

O altă modalitate de rezolvare a problemei ar fi împărțirea șirului de la intrare în  $\sqrt{N}$  secvențe. Pentru fiecare secvență trebuie ținute aceleași valori ca și în cazul arborelui de intervale, iar query-ul se efectuează asemănător. Soluția poate lua de la 60 puncte la 100 puncte, în funcție de implementare.

### 14.5.2 Rezolvare detaliată

### 14.5.3 Codul sursă

## 14.6 Tric - ONI 2007

Din cei  $n$  participanți la olimpiada de informatică se pot distinge  $m$  perechi de prieteni. Aceste perechi au câteva proprietăți interesante:

- dacă  $A$  este prieten cu  $B$ , atunci  $B$  este prieten cu  $A$ ;
- dacă  $A_1$  este prieten cu  $A_2$ ,  $A_2$  cu  $A_3$ , ...,  $A_{k-1}$  cu  $A_k$  și  $A_k$  cu  $A_1$ , atunci există cel puțin o pereche  $(i, j)$ ,  $1 \leq i, j \leq k$ , astfel încât:
  - $A_i$  și  $A_j$  sunt prieteni și
  - $(i \bmod k) + 1 \neq j$  și  $(j \bmod k) + 1 \neq i$

Se numește triunghi de prieteni un set de 3 prieteni  $A$ ,  $B$  și  $C$ , cu proprietatea că  $A$  este prieten cu  $B$ ,  $B$  cu  $C$  și  $C$  cu  $A$ .

#### Cerință

Aflați câte triunghiuri de prieteni există.

#### Date de intrare

Pe prima linie a fișierului de intrare **tric.in** se găsesc, separate prin spații, numerele naturale  $n$  și  $m$ . Pe următoarele  $m$  linii se găsesc perechi de numere  $A$ ,  $B$ , între 0 și  $n - 1$ , cu semnificația că  $A$  este prieten cu  $B$ .

#### Date de ieșire

Pe singura linie a fișierului de ieșire **tric.out** afișați numărul de triunghiuri de prieteni.

#### Restricții și precizări

- $2 \leq n \leq 100000$
- $1 \leq m \leq 100000$
- pentru 20% din teste,  $n \leq 300$



- pentru 50% din teste,  $n \leq 1000$

**Exemplu**

tric.in	tric.out
4 4	1
0 1	
1 2	
2 0	
2 3	

**Limită de memorie:** 64 Mb, din care 1 Mb pentru stivă.

**Timp maxim de execuție/test:** 0.4 secunde

**14.6.1 Indicații de rezolvare - descriere soluție \***

*Ștefan Ciobâcă*

Problema cere gășirea tuturor ciclurilor de grad 3 din graful dat la intrare. Graful acesta este cordal (fiecare ciclu de lungime 4 trebuie să aibă cel puțin o coardă), fapt care ne permite să numărăm triunghiurile în timp  $O(n + m)$ .

Se realizează o ordonare a nodurile  $v_1, v_2, \dots, v_n$ , cu proprietatea că  $v_i$ , împreună cu toții vecinii săi care se găsesc la dreapta lui în ordonare, formează o clică. Deoarece graful este cordal, această ordonare se poate realiza întotdeauna. Odată stabilită această ordine, pentru orice  $i$ , putem număra triunghiurile formate de nodul  $v_i$  împreună cu nodurile  $v_{i+1}, v_{i+2}, \dots, v_n$  în timp  $O(1)$  (dacă  $t$  e numărul de vecini ai lui  $v_i$  aflați la dreapta acestuia, sunt  $t * (t - 1) / 2$  astfel de triunghiuri).

Pentru a găsi ordonarea, există doi algoritmi cunoscuți: LexBFS și MCS. Cel mai simplu este MCS și funcționează astfel:

Alegem nodurile din ordonare pe rând, de la  $v_n$  la  $v_1$ .

- asociem fiecărui nod numărul de vecini aflați la stânga sa, inițializând aceste valori cu 0

- presupunând că am ales  $v_{i+1}, v_{i+2}, \dots, v_n$ , alegem  $v_i$  ca fiind nodul cu valoarea asociată cea mai mare

- incrementăm valoarea asociată tuturor vecinilor lui  $v_i$

Algoritmul se poate implementa în  $O(n + m)$ , dacă se țin toate nodurile care au aceeași valoare asociată într-un bucket.

Se poate demonstra că acest algorithm găsește o ordonare cu proprietatea de mai sus (vezi Tarjan și Yannakakis, "Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs").

**14.6.2 Rezolvare detaliată**

**14.6.3 Codul sursă**