Chronicle

of

Central-European Olympiads in Informatics

♦

a Collection of Tasks

Zalaegerszeg, 2001

HUNGARY

Chronicle
of
Central-European Olympiads in Informatics

♦

a Collection of Tasks

Zalaegerszeg, 2001

HUNGARY

Central-European Olympiad in Informatics (CEOI)

The first International Olympiad in Informatics (IOI) was organised in 1989 under the auspices of UNESCO in Sofia (Bulgaria), where students from 13 countries tried their skills. A year later in Minsk (Belarus), 4-member teams of 25 countries participated; in 1992 contestants of 46 countries went to Bonn (Germany) and IOI 1994 in Haninge (Sweden) attracted teams from 49 countries.

Seeing the growing success of the olympiad, in 1993 the Romanian delegation suggested that a competition of the kind be organised for younger students of Central European countries. At the same time they invited teams from Austria, Croatia, the Czech Republic, Hungary, Poland, Slovakia and Slovenia. Besides the Czech, Croatian, Hungarian, Polish and Romanian teams, the first CEOI in Cluj in 1994, organised by Clara Ionescu, had guest teams from Serbia, Moldova and Turkey. By last year each actively participating country had been a host of a CEOI and the second round began.

1994

Cluj, Romania

Participants: *Croatia, Czech Republic, Hungary, Moldova, Poland, Romania, Turkey, Yugoslavia*

1995

Szeged, Hungary

http://ceoi.inf.elte.hu/ceoi95/

Participants: *Belarus, Croatia, Czech Republic, Estonia, Hungary, Lithuania, Poland, Romania, Slovakia, Ukraine, Yugoslavia*

1998

Zadar, Croatia

http://public.srce.hr/hsin/ceoi98/

Participants : *Bosnia-Herzegovina, Croatia, Czech Republic, Germany, Hungary, Poland, Romania, Slovakia, Slovenia*

1999

Brno, Czech Republic

http://www.fi.muni.cz/ceoi/

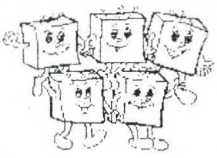
Participants: *Bosnia-Herzegovina, Croatia, Czech Republic, Germany, Hungary, Poland, Romania, Slovakia, Slovenia, USA*

1996

Bratislava, Slovakia

http://turing.fmph.uniba.sk/www/ceoi/

Participants: *Croatia, Czech Republic, Hungary, Poland, Romania, Slovakia, Slovenia*



1997

Nowy Sącz, Poland

http://www.mimuw.edu.pl/oi/ceoi97/

Participants: *Belarus, Croatia, Estonia, Germany, Hungary, Latvia, Lithuania, Netherlands, Poland, Romania, Slovakia, Ukraine, USA, Yugoslavia*

Strona główna

2000

Cluj, Romania

http://ceoi.ubbcluj.ro/

Participants: *Croatia, Czech Republic, Germany, Hungary, Moldova, Netherlands, Poland, Romania, Slovakia, Slovenia, USA*

CEOI 2000

2001

Zalaegerszeg, Hungary

http://ceoi.inf.elte.hu/

Participants: *Austria, Croatia, Czech Republic, Estonia, Finland, Germany, Hungary, Italy, Netherlands, Poland, Romania, Slovakia, Slovenia*

CEOI 2001

Regulations

for the Central-European Olympiad in Informatics

The Olympiad is organised by the Ministry of Culture and Education or another appropriate institution of one of the eight Central-European countries. According to the rules accepted by the initiators of CEOI, teams of nine Central-European countries, i.e. Austria, Croatia, the Czech Republic, Germany, Hungary, Poland, Romania, the Slovak Republic and Slovenia, are invited as regular participants. Moreover, the host country may invite guest participants as well; (in 2001, teams of the following five countries are also welcome: Estonia, Finland, Italy, the Netherlands, and Switzerland)

The Olympic Committee (OC) of CEOI consists of the nine team leaders and the representative of the host country, who chairs the meetings of the OC.

A host, which is willing to organize a CEOI in a certain year in their country, has to announce their intent at least one year before that CEOI (during the previous CEOI competition days). Selection of the next host is made by the OC by majority votes.

Revision of the Regulation of CEOI is adopted by the OC by 2/3 majority of votes.

Enlarging or decreasing the set of CEOI countries can only be adopted by consensus.

Goals

The CEOI aims at motivating secondary school students of Central Europe to

- *get more interested in informatics and information technology in general;*
- *test and prove their competence in solving problems with the help of computers;*
- *exchange knowledge and experience with other students of similar interest and qualification;*
- *establish personal contacts with young people of the Central-European region.*

Additionally, the CEOI may

- *provide training for the students participating in the International Olympiad in Informatics (IOI);*
- *initiate discussions on and co-operation in informatics education in the secondary schools of the Central-European countries.*

General regulations

Each team is composed of up to four secondary school students, a team leader, and a deputy team leader. Only the teams should pay travel costs to and from the place of the competition, the organisers cover all local expenses. Accompanying persons, observers are welcome but they should pay for their stay. Interested persons are advised to contact the local organisers.

The official language is English.

Students may use their mother tongue. The programming problems will be formulated in English, and then translated by the team leaders to the mother tongue of his/her team. Both versions will be given to the students. Team leaders must be able to speak and write in English, as well as the language of their team.

The computers will be IBM PCs or compatibles with MS-DOS and other selected software packages. Only the computers and the software with built-in help facilities provided by the organisers may be used in the competition. In particular, the use of printed material will also be forbidden.

The programming languages of the contest are Turbo Pascal, Turbo C++ and QBasic; the precise versions of these languages will be updated each year. The compilers and programming environments for the above mentioned programming languages as well as the MS-DOS editor will be installed on the hard disk.

Team composition

The students have to be in school during the year when the contest is held and have to be at most 19 years old.

The team leader will be member of the International Jury. He/she will also assist in the evaluation of the solutions of his/her own students.

Observers and persons accompanying a delegation have to pay a fee.

International Jury

The International Jury is composed of the team leaders of the participating countries and the president nominated by the host country.

The International Jury selects the problems to be solved in the competition from a set of problems prepared and proposed by the Scientific Committee. The selection procedure (in line with the procedure applied at the IOI'94 in Sweden) is the following.

- *The chairperson of the Scientific Committee distributes the proposals. Their number equals to the number of the problems to be solved by the competitors.*
- *The jury members may either accept or, in case of major ambiguous formulation or other serious reasons, deny the proposals by voting. When and if a proposal is denied, another prepared proposal will be offered to the jury. For such cases, the Scientific Committee should prepare at least two extra proposals for each round. The jury must not change the text of the accepted proposals, except for minor reformulation that are needed to avoid smaller ambiguities.*
- *The team leaders into the national languages of the teams will translate the selected problems.*

The host countries for the CEOI in the following two years will be determined too, if possible.

Scientific Committee

The Scientific Committee (SC) consists of the chairperson and a number of experts (SC-members) of the host country. It becomes active well before the beginning of the Olympiad, and has the task of selecting and preparing the problem proposals.

Evaluation Committee

The Evaluation Committee (EC) consists of the chief evaluator, a number of the experts of the host country (host evaluators), and a representative of a guest country, preferably the organiser of the next or the previous CEOI (guest evaluator). The task of the Evaluation Committee is to test and evaluate the solutions of the competitors. The task of the guest evaluator is to test the solutions of the competitors of the host country. All EC-members are expected to understand and speak English.

Problems, competition

The competition consists of two rounds in two days, like at the IOI. In both rounds, the work time is five hours, and the competitors will be given one to four problems to solve.

Within the first half an hour the competitors may submit written questions (either in English or their national language) to the jury concerning the formulation and interpretation of the problems. Only questions that can be answered with "Yes", "No" or "No comment" may be accepted. The answers will be produced by the members of the SC and approved by the chairperson of the SC as soon as possible.

When the competition ends, each competitor should make identical copies of her/his solutions on two diskettes; one diskette for her/his team leader, and another one for the evaluation procedure as back-up versions of the original solutions, left on hard disk.

No special hardware requirements or software packages (e.g. no graphic packages) will be needed to solve the problems. All problems and eventual additions or corrections will be given to the competitors in writing; oral communication will be avoided.

Evaluation

When the working time is over, an evaluator in the presence of the competitor and her/his team leader, using previously unpublished test data will check the solutions of each competitor, left on the hard disk. The role of the competitor and the team leader is restricted to advice the evaluator when she/he needs advice; in particular, neither the team leader nor the competitor may operate the computer. Moreover, the team leader has the right to express his agreement or disagreement, in writing, with the evaluator (explained below).

The evaluation will be based on the test data and the responses of the programs only. In the evaluation session, the evaluator runs the programs with the prepared test data, and compares the answers against the expected results. Then the team leader and the evaluator should discuss the answers, and try to agree upon the preliminary evaluation.

At the end of the evaluation session, both the evaluator and the team leader fill out and sign two identical copies of the report forms, one for the evaluator, and the other one for the team leader; in case of disagreement, the team leader may explain his position to the evaluation committee on these report forms.

The evaluation procedure concludes with the meeting of the Evaluation Committee, where the evaluation reports will be discussed, and disagreements will be dissolved by voting. It is the responsibility of the Evaluation Committee that a just and balanced evaluation is achieved.

Finally, the chief evaluator presents the anonymous (!) results to the International Jury to take final decisions. If a team leader can not accept the results of the evaluation, she/he may address her/himself to the International Jury.

Results and prizes

The International Jury will determine the minimum scores for the gold, silver and bronze medals. The proportion of these gold, silver and bronze medals should approximately be 1:2:3. About 50% of the competitors should receive medals. Each competitor will receive a certificate of participation.

The medals, certificates and eventual other prizes will be given to the competitors at the official closing ceremony.

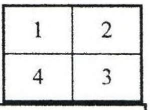
1st Central-European Olympiad in Informatics

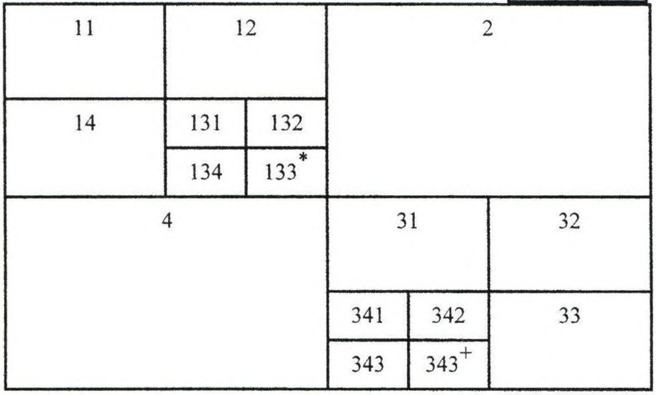
Cluj – Romania, May 27-31., 1994

Day 1 – Problem A

Normalised Squares

For the purposes of computer graphics a part of the screen is always coded by a positive integer consisting only of digits 1,2,3,4. See the pictures:





```
*...133
+...343
DDDDRR
```

Normalized square * is coded by 133, normalized square + is coded by 343. There is no limit on the scale of the normalized squares. If you want to shift square * to the place +, you may shift it for example 4 times down and then 2 times to the right.

Task

Write a program into the file **SQUARE.PAS** which does repeatedly the following:

a) Reads a code of a normalized square from one line of the input file **SQUARE.DAT** as a positive integer, consisting only of digits 1,2,3,4 and a sequence of shifts **L** (left), **R** (right), **U** (up), **D** (down) as a sequence of letters **L**,**R**,**U**,**D** from the next line of the input file **SQUARE.DAT**;

b) Writes the final position of the normalized square into the output file **SQUARE.RES** or writes message

```
OUT OF THE BORDER
```

c) Writes an empty line into the output file **SQUARE.RES**, until an empty line is read.

In each data set, the number of digits in the code is less or equal to 35.

The time limit is at most 1 minute for each data set.

```
OUT OF BORDER
```

Example:

If the file **SQUARE.DAT** contains	the output file **SQUARE.RES** should contain

`133`
`DDDDRR`
`1`
`DD`
	`343`

Day 1 – Problem B

Subsets

Let n be a positive integer. We consider an ordering <, called **lexicographic ordering**, defined on the subsets of {1,2,...,n}. Let $S_1=\{x_1,...,x_i\}$ and $S_2=\{y_1,...,y_j\}$ be two distinct subsets of {1,2,...,n} with $x_1<x_2<...<x_i$ and $y_1<y_2<...<y_j$. Then we say that $S_1<S_2$ if there exists k with $0\leq k\leq \min\{i,j\}$ such that $x_1=y_1,...,x_k=y_k$ and either k=i or $x_{k+1}<y_{k+1}$.

The lexicographic ordering associates to each subset a natural number, as shown in the above example.

For example, the subsets of {1,2,3} listed in lexicographic order are:

{}	1
{1}	2
{1,2}	3
{1,2,3}	4
{1,3}	5
{2}	6
{2,3}	7
{3}	8

Your program should read lines from an ASCII file called **P6.TXT**. Each line has one of the following forms:

1 n k

2 n k_1 k_2 . . . k_i

If the line has the first form, you have to print on the display the subset of {1,2,...,n} whose associated number is k (supposing that $k\leq 2^n$).

If the line is of the second form, you have to print the number associated with the subset $\{k_1,...,k_i\}$ of {1,...,n} (supposing that $1\leq k_1<k_2<...<k_i\leq n$).

Your program should be able to produce the answer for each input line in at most 3 minutes, assuming that $n\leq 30$.

Day 1 – Problem C

Objects

We construct objects according to the following rules:

1. The initial object is a square with the side equal to 1;

2. A new object is obtained concatenating two objects on the side which has the same length.
3. Each time a new object is constructed, it is supposed that an infinite instances of this object are available.

Requirements:

1. A positive integer n is introduced from the keyboard. Display the number of objects having the area at most n, which can be constructed if only two objects of the same dimensions will be concatenated. Two objects with the same dimensions are considered identical. For example, a valid output for n=20 is:

   (1,1) (2,1) (4,2) (8,2) (16,3)

   where in each pair the first number represents the area of an object and the second one represents the number of distinct objects having this area
2. The same requirements must be satisfied when we are allowed to concatenate objects of different dimensions, according to the rules 1 - 3. For example, a valid output for n=10 is:

   (1,1) (2,1) (3,1) (4,2) (5,1) (6,2) (7,1) (8,2) (9,2) (10,2)

   the pairs having the same meaning as in requirement 1).

The input is a single line introduced from the keyboard, containing a pair:

i n

where i∈{1,2}, meaning the number of the requirement and n is the above specified integer number.

The output is a text file containing pairs of elements, as shown above.

The maximum value for n is 10000. Your program should be able to produce the answer for each data test in at most 1 minute.

Day 2 - Problem A

Black or white

Write a program which reads three positive integers n, p, q. Decide whether or not there exists a sequence of n integers such that the sum of any p consecutive elements is positive and the sum of any q consecutive ones is negative. If the answer is **YES**, your program has to produce such a sequence.

The values of n, p, q are read from the keyboard; the output consists of **NO** or **YES** followed by a sequence of n integers written on display.

Examples:

The input is:
```
n=4
p=2
q=3
The output is
NO
```

The input is:
```
n=6
p=5
q=3
The output is
YES
-3 5 -3 -3 5 -3
```

Day 2 – Problem B

Odd and even

Let us consider a country with N towns. A system of roads formed from direct connections between towns is given; all these direct connections are considered to have the length equal to 1. This system is called **even-odd** if there are two towns linked by a road of even length, as well as by a road of odd length.

a) Find out if the system of roads is even-odd or not.

b) If the answer for a) is negative, find one of the subsets X of towns which has a maximal number of elements and satisfies the following condition: for any two towns in X, if there is a path linking them, then its length is even.

The name of the input file is introduced from the keyboard. Its first line contains the value of N; the following lines contain pairs I,J with the meaning that the towns I and J are directly linked.

The value of N is at most 300.

Display the output on the screen in an inteligible form.

Examples:

If the input file contains:
```
5
1 2
2 3
3 4
4 5
5 1
```
a valid output is:
```
YES
```

If the input file contains:
```
3
1 2
```
then a valid output is:
```
NO
X has 2 elements
X: 2 3
```

Day 2 – Problem C

Expressions

An expression contains additions and multiplications. The time needed to evaluate an addition (+) is p and the time needed for a multiplication (*) is q. The time needed to evaluate the expression **AoB** is equal to the time needed to evaluate the operation **o** plus the maximum of the times needed to evaluate the subexpressions A, and B respectively.

The operands are variables consisting of a single character, whose evaluation time is considered as being equal to 0.

Write a program which reads data from an input file that contains the values of p and q and expressions, each expression on a separate line. The parantheses are used compulsory to indicate the order of operations. For each expression one asks:

a) To find and print the time needed to evaluate that expression;

b) To find and print an equivalent arithmetic expression with a minimum evaluation time, as well as this minimum evaluation time on next line.

The equivalent transformations permitted are:

x+y=y+x; x*y=y*x; (commutativity)

x+(y+z)=(x+y)+z; x*(y*z)=(x*y)*z (associativity).

The results must be printed in the output file that contains an empty line between two expressions.

For example, if the input file contains:

```
1 1
((a+(b+(c+d)))*e)*f
((((a*b)*c)*d)+e)+(f*g)
```

then the output file must be:

```
5
((a+b)+(c+d))*(e*f)
3
5
(((a*b)*(c*d))+e)+(f*g)
4
```

Final results

Name	Country	Final points
Gold medal		
Alexandru Sălcianu	Romania	*580*
Jirí Hajek	Czech Republic	*522*
Marx Dániel	Hungary	*502*
② *Silver medal*		
Michal Wala	Poland	*490*
Krysztof Sobosiak	Poland	*478*
Virgil Palanciuc	Romania	*466*
Blahut György Gábor	Hungary	*450*
Vladimir Brankov	Yougoslavia	*445*
Marek Stocki	Poland	*430*
Filip Dugandzich	Yougoslavia	*428*
③ *Bronz medal*		
Adrian Soviani	Romania	*424*
Jakub Pawlewicz	Poland	*422*
Kovács Gábor	Hungary	*409*
Iulio Vasilescu	Romania	*393*

Igor Irich	Yougoslavia	*382*
Milosh Djermanovich	Yougoslavia	*380*
Renato Zeleznjak	Croatia	*374*
David Stanovsky	Czech Republic	*372*
☹	***No medal***	
Jan Kratochvil	Czech Republic	*342*
Vladimir Ufnarovschi	Moldavia	*320*
Posta Zoltán	Hungary	*285*
Igor Vulkman	Croatia	*267*
Iuri Smicov	Moldavia	*243*
Drazen Pavkovic	Croatia	*220*
Lev Lande	Moldavia	*212*
Vjeskolav Babic	Croatia	*206*
Mustafa Dogru	Turkey	*172*
Petr Kanovsky	Czech Republic	*145*
Arda Bavrak	Turkey	*98*
Baha Eren Cevik	Turkey	*73*
Alexandru Andoni	Moldavia	*73*

2nd Central-European Olympiad in Informatics

Szeged – Hungary, May 29 - June 3, 1995

Day 1 – Problem A

Idle machine

A computer records the starting and finishing time of each application program in the measure of 1/100 seconds. A daily statistic is a set of pairs (a, b) of non negative integers such that $0<a\leq b<8640000$. The pair (a, b) means that a program started at time a and finished at time b. The time points a and b belong to busy time. Arbitrary number of programs can run in parallel. Write a program that computes a) the length (b-a+1) of the largest closed time interval [a, b] when the machine was idle and b) the length (d-c+1) of the largest closed time interval [c, d] when the machine executed at least one program. A time point t belongs to the idle time if there is no program running at time t.

Input Data

The first line of text file **DAY1A.DAT** contains N, the number of the intervals ($1\leq N\leq 1000$). Each of the rest N lines of the file contains two non negative integers, the starting and finishing time of a program.

Example input:

```
9
30000 35000
10000 20000
15000 16000
40000 44000
77000 220000
13000 41000
60000 67000
50000 55000
65000 70000
```

Output Data

The text file **DAY1A.SOL** has to contain the following two integers in this order:

a) The length (b-a+1) of the largest interval of time [a, b] when the machine was idle.
b) The length (d-c+1) of the largest closed time interval [c, d] when the machine executed at least one program.

Example output:

In our example input the **DAY1A.SOL** file contains the following two lines

```
8419999
143001
```

Maximum number of points: 25.

Day 1 - Problem B

Alarm chain

The students of a class have decided to form an alarm chain. Every student chooses a unique successor, to whom he directly delivers received messages. Whenever a student receives a message forwards it to his or her successor.

Such an assignment is called an alarm chain if the following holds. Suppose somebody sends a message to his or her successor, who in turns transmits the message to his or her successor, etc. The message eventually arrives to every student, including the starting student.

Clearly, not every assignment is alarm chain. Write a program which for an arbitrary input assignment, computes the minimal number of necessary modifications that transforms the input assignment to an alarm chain.

Input Data

The first line of the text file **DAY1B.DAT** contains N, the number of the students ($1 \le N < 256$). Each of the following N lines contains two names (strings) separated by the character '>'. The second name terminated by end-of-line character. The names are at most 20 characters long. A line of the form A>B means that the successor of student A is B, i.e. A directly delivers messages to B.

Example input:

```
10
Anita>Peter
Andrew>Julia
David>Andrew
Natalie>Gabriella
Edith>David
Peter>Anita
Gabriella>Julius
Adam>David
Julia>Gabriella
Julius>Julia
```

Output Data

Write the minimal number of necessary modifications to the text file **DAY1B.SOL** as an integer.

Example output:

The output for our example would be:
4

Maximum number of points: 35.

Day 1 - Problem C

Fair sharing

Two brothers, Alan and Bob want to share a set of gifts. Each of the gifts should be given to either Alan or Bob, and none of the gifts can be split. Each gift has a positive integer value. Let A and B denote the total value of gifts received by Alan and Bob, respectively. The aim is to minimize the absolute value of the difference A-B. Write a program which computes the values A and B.

Input Data

The first line of the text file **DAY1C.DAT** contains N, the number of the gifts (1≤N≤ 100). The rest of the input contains N positive integers, the values of the gifts. Each value is ≤200.

Example input:

```
7
28 7 11 8 9 7 27
```

Output Data

Write the two integers A and B to the text file **DAY1C.SOL**.

Example output:

The output for our example would be:
48 49

Maximum number of points: 40.

Day 2 - Problem A

Garden

There are N trees in a garden. The shape of the garden is square with 1000 m long sides. We are looking for a rectangle with largest area that does not contain trees inside. The sides of the rectangle must be parallel to the corresponding sides of the garden. Each side of the rectangle may contain trees and may lie on a side of the garden. Trees are given by (x, y) coordinates of their positions measured in meter. A tree is considered to be a point without extension.

The origin of the coordinate system is the lower left corner of the garden and the axes are parallel to the sides.

Input Data

The first line of text file **DAY2A.DAT** contains N, the number of the trees (1≤N≤600). In each of the next N lines you will find integer coordinates (x, y) (0<x<1000, 0<y<1000), the positions of the trees.

Example input:

```
7
280 100
200 600
400 200
135 800
800 400
600 800
900 210
```

Output Data

Write the area of the largest rectangle as an integer value in the text file **DAY2A.SOL**.

Example output:

The output for our example would be:
360000

Maximum number of points: 30.

Day 2 - Problem B

Party

The president of a company is organizing a party for his employees. The company has a hierarchical structure; that is, the supervisor relation forms a tree rooted at the president. In order to make the party fun for all attendees, the president does not want both an employee and his or her supervisors to sit at the same table.

The problem is to compute the minimal number of tables necessary to make a seating arrangement that satisfies the above requirement. The input of the program is the immediate supervisor relation of the company. The supervisor relation is defined in terms of the immediate supervisor relation as follows. A person P is supervisor of a person Q if P is the immediate supervisor of Q, or there are persons P_1, ... ,P_k ($1<k$) such that $P=P_1$, $Q=P_k$ and P_i is the immediate supervisor of P_{i+1} ($i=1, \ldots ,k-1$).

The number of attendees is N ($1 \le N \le 200$), and there are T ($2 \le T \le 10$) seats at each table.

Input Data

The employees of the company are identified by the first N ($1 \le N \le 200$) natural numbers. The president of the company is identified by number 1, and has no supervisor. The first line of the text file **DAY2B.DAT** contains T, the number of seats per table ($2 \le T \le 10$). The second line contains the number N ($1 \le N \le 200$) of the attendees, each employee identified by numbers from 1 to N will attend the party. In the third line you find N numbers. The i-th number in the line is the immediate supervisor of the i-th person. The first number in the line is 0, indicating that the president has no supervisor.

Example input:

```
4
13
0 1 9 9 9 2 2 1 1 7 8 8 10
```

Output Data

Write the minimal number of tables needed to seat all attendees in the specified way in the text file **DAY2B.SOL**.

Example output:

The output for our example would be:

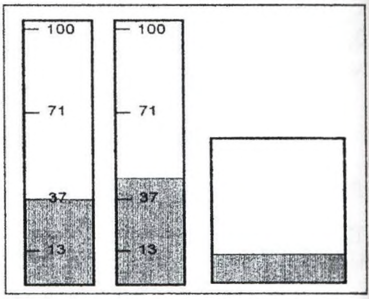
```
5
```

Maximum number of points: 30.

Day 2 - Problem C

Measuring glasses

We are given three glasses. The volume of each glass is 100 unit (cm^3). The first two glasses have marks, same on both glasses, each mark indicating the volume measured from the bottom up to the mark. That volume is written beside the mark (see figure bellow). Initially the first glass contains 100 unit volume of fluid, and the others are empty. Write a program that decides whether one unit volume of fluid can be separated in the third glass, and if so, computes the minimal number of steps needed to do it. In each step, after the operation at least one of the used glass must contain fluid up to a mark or empty. During the procedure we can keep track of the actual contents of each three glasses.



Input Data

The first line of the text file **DAY2C.DAT** contains **N**, the number of the marks on the first two glasses (1≤N≤20). The rest of the file contains **N** integers in increasing order; that is the volumes corresponding to the marks on the glasses. For each volume mark V the inequalities (1≤V≤100) hold and the lergest value is 100.

Example input:

```
4
13 37 71 100
```

Output Data

Write the string "YES" into the first line of the text file **DAY2C.SOL** if one unit can be separated in the third glass and write "NO" otherwise. If the answer is yes for the first question then write the minimal number of steps in the second line.

Example output:

The output for our example would be:

```
YES
8
```

Maximum number of points: 40.

Final results

Name	Country	Final points
	[illegible]	
Vladimir Brankov	Yugoslavia	*195*
Daniel Kral	Czech Republic	*185*
Martin Hajduch	Slovakia	*179*
②	***Silver medal***	
Piotr Zielinski	Poland	*175*
Mikulas Patocka	Czech Republic	*173*
Ovidiu Ghiorghioiu	Romania	*173*
Jan Kratochvil	Czech Republic	*158*
Martin Irman	Slovakia	*157*
Jakub Pawlewicz	Poland	*150*
Lakatos Roland	Hungary	*149*
Gosztolya Gábor	Hungary	*146*
Marcin Sawicki	Poland	*146*
③	***Bronz medal***	
Marius Vlad	Romania	*145*
Martin Domány	Slovakia	*142*
Cristian Tapus	Romania	*138*
Vaidas Gasiunas	Lithuania	*136*
Kirilov Olexy	Ukraine	*135*
Heno Ivanov	Estonia	*133*
Jaroslav Blagojevic	Yugoslavia	*132*
Magnus Hiie	Estonia	*130*
Mikolaj Gawron	Poland	*126*
Costin Esanu	Romania	*126*
☹	*No medal*	
Tomas Laurinavicius	Lithuania	*122*
Serguei Sinkevitch	Belarus	*121*
Milos Dermanovic	Yugoslavia	*116*
Stanislav Mikes	Czech Republic	*115*
Dubinsky Leonid	Ukraine	*113*
Vladimir Marko	Slovakia	*109*
Vaidotas Urba	Lithuania	*108*
Lyutko Pavlo	Ukraine	*108*
Igor Skochinsky	Belarus	*105*
Kovács Gábor Zsolt	Hungary	*85*
Erik Laansoo	Estonia	*84*
Mikas Grigaliunas	Lithuania	*80*
Elek Róbert	Hungary	*72*
Vladimir Filipovic	Yugoslavia	*70*
Iaroslav Tkatchev	Belarus	*68*
Damir Milotic	Croatia	*64*
Oleg Mürk	Estonia	*62*
Mladen Faraguna	Croatia	*58*

Andrei Loutchkov	Belarus	*55*
Marin Kovacic	Croatia	*53*
Potsepaev Roman	Ukraine	*53*
Ivana Bokun	Croatia	*0*

rd Central-European Olympiad in Informatics

Bratislava - Slovakia, October 9-13., 1996

Day 1 - Problem A

Encoding grid

A kind of grid is sometimes used for encoding messages. Our naval fleet officials decide to use this kind of encoding messages for communication with their ship captains. The encoding grid is a square sheet of paper divided into 2N x 2N little squares. N^2 of little squares are cut out (see figure 1).

Let us briefly describe the encoding process. Take text of message of length 4N^2. Then put an encoding grid onto a blank sheet of paper and begin to write first N^2 letters of message into the cut little squares (one letter per square, write letters in all cut squares in the first line from left to right, then in the second line etc.). See example on figure 2 for message "HELLOYELLOWWORLD". After finishing the first step rotate the encoding grid 90 degrees clockwise and continue similarly writing letters. After two more steps all letters should be written (see example on figure 3).

`O###`
`##O#`
`O##O`
`####`
	`H###`
	`##E#`
	`L##L`
	`####`
		`HOOY`
		`LREO`
		`LWEL`
		`LLDW`

fig. 1	*fig. 2*	*fig. 3*

It is necessary to have a correctly constructed grid. It means that when it is used in the way described above after each rotation new N^2 blank squares will appear under cut squares of the grid.

The naval fleet officials encoded thousands of messages with their grid and then they wanted to send them to ships. Unfortunately they lost the grid. Fortunately the naval admiral remembered the original text of one of the messages.

Your task is to get this original message text and encoded text and find one possible correctly constructed encoding grid with which the message text could have been encoded.

Input Data

The first line of input file contains an integer N (1≤N≤10). The second line contains 4N^2 capital letters representing the original message. The next 2N lines contain the encoded text (each line 2N capital letters).

Output Data

The output file contains a correctly constructed grid with which given message could have been encoded. In each of 2N lines one line of grid (2N characters) will be displayed where "O" (capital letter 'O' represents cut square and "#" uncut square.

Example:

```
GRID.IN                              GRID.OUT
2                                    O###
HELLOYELLOWWORLD                     ##O#
HOOY                                 O##O
LREO                                 ####
LWEL
LLDW
```

Day 1 - Problem B

Ships

The Palmia country is divided by a river into the north and south bank. There are N towns on both the north and south bank. Each town on the north bank has its unique friend town on the south bank. No two towns have the same friend.

Each pair of friend towns would like to have a ship line connecting them. They applied for permission to the government. Because it is often foggy on the river the government decided to prohibit intersection of ship lines (if two lines intersect there is a high probability of ship crash).

Your task is to write a program to help government officials decide to which ship lines they should grant permission to get maximum number of non intersecting ship lines.

Input Data

The input file consists of several blocks. In the first line of each there are 2 integers X,Y separated with exactly one space, X represents the length of the Palmia river bank ($10\leq X\leq 6000$), Y represents the width of the river ($10\leq Y\leq 100$). The second line contains the integer N, the number of towns situated on both south and north riverbanks ($1\leq N\leq 5000$). On each of the next N lines there are two non-negative integers C,D separated with space ($C,D\leq X$), representing distances of the pair of friend towns from the western border of Palmia measured along the riverbanks (C for the town on the north bank, D for the town on the south bank). There are no two towns on the same position on the same riverbank. After the last block there is a single line with two numbers 0 separated by a space.

Output Data

For each block of the input file the output file has to contain in consecutive lines the maximum possible number of ship lines satisfying the conditions above.

Example:

```
SHIPS.IN
30 4
```

```
7
22 4
2 6
10 3
15 12
9 8
17 17
4 2
0 0

SHIPS.OUT
4
```

Day 1 - Problem C

Highway tolls

In Palmia country they have N cities connected by highways (each highway connects exactly two cities in both directions). It is possible to reach every city from any other city using one or several highways. Till this year the highway tolls were collected on highways. In the middle of each highway there was a toll collector who collected 1 Palmia Coin (1 PC) from each car passing by.

Government officials decided to reduce the number of toll collectors and introduce highway stamps. If a car will want to enter a highway it must have this higway stamp placed on the window.

Officials decided that the highway stamp for one year will cost the same as 100 travels between two farthest cities. The distance between two cities is the minimum number of highways you need to use to get from the first city to the second one.

Your task is to write a program which computes the cost of the highway stamp.

Input Data

The input file consists of several blocks of data. Each block at the first line contains the integers N and M separated by a space where N is the number of cities and M the number of highways in Palmia country ($1 \le N \le 1000$, $1 \le M \le 2000$). Cities are numbered by integers from 1 to N. The next M lines contain each one pair of integers A,B ($1 \le A$, $B \le N$) separated by a space representing that there is a highway between the cities A and B. After the last block there is a single line with two numbers 0 separated by a space.

Output Data

For each block of input file the output file contains a single line with the cost of the highway stamp.

Example:

```
TOLLS.IN
4 4
1 2
2 3
```

```
4 2
3 4
0 0

TOLLS.OUT
200
```

Day 2 - Problem A

Bin packing

A factory has a bin packing robot at the end of an assembly line. There are exactly two bins open, and the robot packs items into any one of the open bins, one by one sequentially. The bins are identical, and one bin can accommodate a set of items up to a given weight limit. More precisely, the robot can perform the following four instructions:

1. Pack the current item into bin 1.
2. Pack the current item into bin 2.
3. Close bin 1 and open a new empty bin in place of the closed bin.
4. Close bin 2 and open a new empty bin in place of the closed bin.

A pack instruction can only be executed if the weight of the current item plus the sum of the weights of the items already in the bin does not exceed the limit. You are given the sequence of weights of items and a weight limit that is constant for all the bins. Write a program that computes the minimal number of bins needed by the robot to pack all items into them.

Input Data

The input file contains integer numbers. The first line of input file contains the weight limit L ($1 \leq L \leq 100$). This limit applies to all bins. The second line contains the number of items N ($1 \leq N \leq 5000$). Each of the following N lines contains the weight of an item. Each weight is at least 1 and at most L.

Output Data

Write on the first line of output file the minimum number of bins needed to pack all items.

Example:

```
BINPACK.IN                          BINPACK.OUT
8                                   3
6
4
2
5
3
5
4
```

Day 2 - Problem B

Cutting rectangles

You are given a rectangle whose side lengths are integer numbers. You want to cut the rectangle into the smallest number of squares, whose side lengths are also integer numbers. Cutting can be performed with a cutting machine that can cut only from side to side across, parallel with one side of the rectangle. Obtained rectangles are cut separately.

Input Data

The input file contains two positive integers in the first line: the lengths of the sides of the rectangle. Each side of the rectangle is at least 1 and at most 100.

Output Data

The output file consist of one line on which your program should write the number of squares resulting from an optimal cutting.

Example:

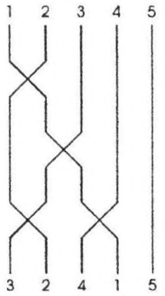
```
CUTS.IN                  CUTS.OUT
5 6                      5
```

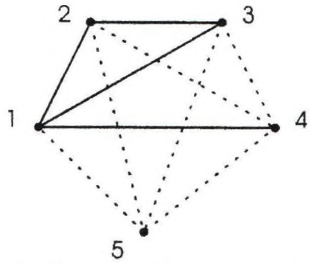
Day 2 - Problem C

Electrician

The electrician Fero has just finished the installation of the new cable in the building of the CEOI secretariat. The cable consists of N wires and connects the computer room and CEOI secretary office. In the computer room Fero labeled the wires with numbers from 1 to N from left to right. Because the cable was long, sometimes some of wires crossed between the computer room and the office (each pair of wires crosses at most once; at each moment only neighbouring wires may cross). So in the office the wires were not ordered the same way (see figure 1).

To know how these wires are crossed Fero drew the graph on the wall near the cable end in the computer room. The vertices represented wires and edges represented wire crosses. There is an edge between vertices a and b if and only if the wires a and b are crossed somewhere on the way (see figure 2 for graph representing situation in figure 1).





The cable does not function properly now. Unfortunately Fero broke his leg yesterday. We have another electrician but he is not so experienced in graph theory. He needs to know the order of wires at the end of the cable in the CEOI secretary office to correct this fault.

Input Data

In the input file there are several blocks of data. The first line of each block contains two numbers N and M separated by a space, where N ($1 \le N \le 100$) is the number of wires in the cable and M is the number of wire crosses. This is followed by M lines each containing pair of integers A, B separated by a space ($1 \le A,B \le N$) representing the cross of cable wires A and B somewhere on the way. After the last block there is a line with two numbers 0.

Output Data

For each block of the input file the output file contains a single line with the list of wire numbers in order as they appear at the second end of the cable (N numbers separated by space) or the message "IMPOSSIBLE" if no order is represented by given graph.

Example:

```
ELECTRIC.IN
5 4
1 2
1 3
2 3
1 4
0 0

ELECTRIC.OUT
3 2 4 1 5
```

Final results

Name	Country	Final points
	[illegible]	
Adam Borowski	Poland	*182*
Stanislav Funiak	Poland	*180*

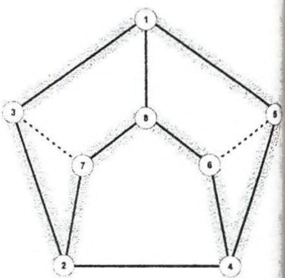
②	***Silver medal***	
Erik Kopczyński	Poland	*173*
Miroslav Dudik	Slovakia	*165*
Piotr Zieliński	Poland	*144*
Vladimir Marko	Slovakia	*127*
③	***Bronz medal***	
Andrej Gąsienica-Samek	Poland	*122*
Vlad Petric	Romania	*122*
Martin Hajduch	Slovakia	*117*
Mitja Šlenc	Slovenia	*112*
Zoran Majstrović	Croatia	*108*
Zvonimir Bujanović	Croatia	*107*
Sergiu Ştefanov	Romania	*105*
☹	***No medal***	
Vinko Petričević	Croatia	96
Ciprian Şerbu	Romania	95
András Nagy	Hungary	92
Věroslav Kaplan	Czech Republic	*90*
Zsolt Husz	Romania	*83*
László Tóth	Hungary	*83*
Mikuláš Patočka	Czech Republic	*78*
Jan Kratochvíl	Czech Republic	*64*
Klemen Kenda	Slovenia	*61*
Damir Milotić	Croatia	*51*
Zsolt Zsila	Hungary	*44*
Martin Dráb	Czech Republic	*41*
Balázs Peller	Hungary	*40*
Klemen Štembal	Slovenia	*24*

4th Central-European Olympiad in Informatics

Nowy Sacz - Poland, July 17-24., 1997

Day 1 - Problem A

The Cave



There is a lot of caves in Byteland. This is the map of one of them:

All caves in Byteland have the following properties:

- All the chambers and passages are on the same level.
- No two passages cross each other.
- Some of the chambers are situated on the outer circle. They are called **outer chambers**.
- All the other chambers lie inside the outer circle and are called **inner chambers**.
- There is an entrance to the cave leading to one of the outer chambers.
- There are exactly three passages leading from every chamber to three different other chambers. If a chamber is an outer chamber then two of the passages lead to two neighbouring outer chambers on the circle and exactly one leads to an inner chamber.
- Passages connecting outer chambers are called **outer passages** and the remaining ones are called **inner passages**.
- It is possible to walk from one chamber to another using only inner passages but there is only one way to do that if we allow to walk through every inner passage at most once.
- Not all passages are equally hard to go through. They are divided into two groups: easy and hard.

It has been decided to open all the caves to the public. To assure a "smooth" and safe flow through a cave, visitors should follow a route marked in advance and visit every chamber of this cave exactly once.

An exception from this rule is the entrance chamber where every tour begins and ends, you are allowed to visit this chamber exactly twice. The tour route should be fit for an average visitor and contain as few hard passages to walk through as possible.

Example:

Consider the cave showed in the figure. The entrance chamber is `1`. The dashed passages are hard to walk through.

Route `1, 5, 4, 6, 8, 7, 2, 3` contains no hard passages at all. The final chamber, number `1,` is implied and not listed.

Task

Write a program that:

- reads the description of a cave from the text file **CAV.IN**;
- finds a route through the cave that begins and ends in the entrance chamber, lets the visitors see all the other chambers only once and contains as few hard passages as possible;
- writes the result to the text file **CAV.OUT**.

Input Data

There are two integers n, k (separated by a single space) in the first line of the text file **CAV.IN**.

The integer n, $3 < n \leq 500$, is the number of all chambers in a cave and k is the number of all its outer chambers, $k \geq 3$. The chambers are numbered from 1 to n. Chamber 1 is the entrance chamber. Chambers 1, 2, ..., k are outer chambers. They do not have to appear on the outer circle in this order.

The next 3n / 2 lines contain descriptions of the passages. Each description consists of three integers a, b, c separated by single spaces. The integers a and b are numbers of chambers at the ends of a passage. The integer c equals 0 or 1, where 0 means that the passage is easy to walk through and 1 that it is hard.

Output Data

Your program should write a sequence of n integers separated by single spaces to the first line of the text file **CAV.OUT**; the sequence has to begin with 1 (the entrance chamber). The following n – 1 integers should be consecutive chambers on the computed route.

Example:

For the text file `CAV.IN`:

```
8 5
1 3 0
3 2 0
7 3 1
7 2 0
8 7 0
1 8 0
6 8 0
6 4 0
6 5 1
5 4 0
2 4 0
5 1 0
```

one of the correct solutions is the text file `CAV.OUT`:

```
1 5 4 6 8 7 2 3
```

Day 1 - Problem B

Hexadecimal numbers

The base of the hexadecimal system is 16. In order to write down numbers in this system one uses 16 digits 0,1,...,9,A,B,C,D,E,F. The capital letters A,..,F stands for 10,..,15, respectively. For instance the value of the hexadecimal number CF2 is $12 * 16^2 + 15 * 16 + 2 = 3314$ in the decimal system. Let X be the set of all positive integers whose hexadecimal representations have at most 8 digits and do not contain repetitions of any digit. The most significant digit in such a representation is not zero. The largest element in X is the number with the hexadecimal representation FEDCBA98, the second largest is the number FEDCBA97, the third is FEDCBA96 and so forth.

Task

Write a program that:

- reads positive integer n from the text file **HEX.IN**, n does not exceed the number of elements of X;
- finds the n-th largest element of X;
- writes the result to the text file **HEX.OUT**.

Input Data

The first line of the file **HEX.IN** contains integer n in the decimal representation.

Output Data

Your program should write the n-th largest element in X in the hexadecimal representation to the first line of the text file **HEX.OUT**

Example:

For the text file `HEX.IN`:	the correct solution is the text file `HEX.OUT`:
`11`	`FEDCBA87`

Day 1 - Problem C

Integer intervals

An integer interval [a,b], a < b, is a set of all consecutive integers beginning with a and ending with b.

Task

Write a program that:

- reads the number of intervals and their descriptions from the text file **INT.IN**;
- finds the minimal number of elements in a set containing at least two different integers from each interval;

- writes the result to the text file **INT.OUT**.

Input Data

The first line of the text file **INT.IN** contains the number of intervals n, $1 \le n \le 10000$. Each of the following n lines contains two integers a, b separated by a single space, $0 \le a < b \le 10000$. They are the beginning and the end of an interval.

Output Data

Your program should write one integer to the first line of the text file **INT.OUT**; this should be the minimal number of elements in a set containing at least two different integers from each interval.

Example:

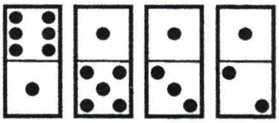
For the text file `INT.IN`:	the correct solution is the text file `INT.OUT`:

`4`
`3 6`
`2 4`
`0 2`
`4 7`
	`4`

Day 2 - Problem A

Dominoes

A domino is a flat, thumbsized tile, the face of which is divided into two squares, each left blank or bearing from one to six dots. There is a row of dominoes laid out on a table:



The number of dots in the top line is 6+1+1+1=9 and the number of dots in the bottom line is 1+5+3+2=11. The gap between the top line and the bottom line is 2. The gap is the absolute value of difference between two sums.

Each domino can be turned by 180 degrees keeping its face always upwards.

What is the smallest number of turns needed to minimise the gap between the top line and the bottom line?

For the figure above it is sufficient to turn the last domino in the row in order to decrease the gap to 0. In this case the answer is 1.

Task

Write a program that:

- reads the number of dominoes and their descriptions from the text file **DOM.IN**,
- computes the smallest number of turns needed to minimise the gap between the top line and the bottom line,
- writes the result to the text file **DOM.OUT**.

Input Data

The first line of the text file **DOM.IN** contains an integer n, $1 \le n \le 1000$. This is the number of dominoes laid out on the table.

Each of the next n lines contains two integers a, b separated by a single space, $0 \le a, b \le 6$. The integers a and b written in the line i + 1 of the input file, $1 \le i \le 1000$, are the numbers of dots on the i-th domino in the row, respectively, in the top line and in the bottom one.

Output Data

Your program should write exactly one integer to the first line of the text file **DOM.OUT**; this integer should be the smallest number of turns needed to minimise the gap between the top line and the bottom line.

Example:

For the text file `DOM.IN`:	the correct solution is the text file `DOM.OUT`:

4
6 1
1 5
1 3
1 2
	1

Day 2 - Problem B

River crossing

Citizens of Byteland adore all sports in which logical thinking is as important as physical skills. One of these sports is crossing the Hex River – the widest river in Byteland. There are n poles numbered 1...n (from left to right) stretching across the river. The citizens have to cross the river by going from the **left bank** to one **pole** perhaps to another and so on and then to the **right bank**. The left bank is located one pole to the left of pole 1; the right bank is located one pole to the right of pole n.

At time 0, the citizen is on the left bank of the Hex River with the goal of reaching the right bank of the river as quickly as possible. At any given time, each pole is either **up** or **down** and the citizen is standing on a pole or standing on a bank of the river. A citizen can stand on a pole only if it is up; such a pole is **available**.

Each pole is down at time 0 and then spends a time units up, b time units down, a time units up, b time units down, etc. The constants a and b are defined separately for each pole. For example the pole with a=2 and b=3 will be down at time 0, up at time 1 and 2, down at time 3, 4, 5 and so on.

At time t+1, a citizen can choose to be on any available pole or river bank within 5 poles of his location at time t or even to stay on his current pole (if available) or bank. For example from pole 5 you can reach any of the poles 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 or the left bank.

Task

Write a program that:

- reads the number of data blocks from the text file **RIV.IN** (each block contains a complete set of data for the problem);
- for each block
  - o reads the number of poles and descriptions of their behaviour;
  - o computes the first possible time the citizen can stand on the right bank, if it is reachable;
  - o writes the result to the text file **RIV.OUT**.

Input Data

The first line of the input file RIV.IN contains the number of data blocks x, $1 \le x \le 5$. The following lines comprise the x blocks. The first block starts on the second line of the input file; each subsequent block starts directly after the previous one.

The first line of each block contains an integer n, $5 < n \le 1000$, the number of poles.

Each of the following n lines in the block contains two integers a, b separated by a single space, $1 \le a, b \le 5$. The integers in line i + 1 (in the block), $1 \le i \le n$, describe the behaviour of the pole i.

Output Data

For the k-th block, $1 \le k \le x$, write to the k-th line of the text file **RIV.OUT** the first time the citizen can reach the right bank or the word NO, when such a crossing is impossible.

Example:

For the text file `RIV.IN`:

```
2
10
1 1
1 1
1 1
1 1
1 1
1 1
1 1
1 1
1 1
1 1
10
1 1
1 1
1 1
1 1
2 1
1 1
```

```
1 1
1 1
1 1
1 1
```

the correct solution is the text file `RIV.OUT`:

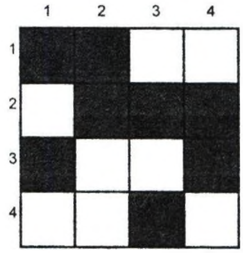
```
NO
4
```

Day 2 - Problem C

Shooting contest

Welcome to the Annual Byteland Shooting Contest. Each competitor will shoot to a target which is a rectangular grid. The target consists of r × c squares located in r rows and c columns. The squares are coloured white or black. There are exactly two white squares and r – 2 black squares in each column. Rows are consecutively labelled 1,..,r from top to bottom and columns are labelled 1,..,c from left to right. The shooter has c shots.

A volley of c shots is **correct** if exactly one white square is hit in each column and there is no row without white square being hit. Help the shooter to find a correct volley of hits if such a volley exists.



Example:

Consider the following target:
Volley of hits at white squares in rows `2, 3, 1, 4` in consecutive columns `1, 2, 3, 4` is correct.

Task

Write a program that:

- reads the number of data blocks from the text file **SHO.IN**; each block contains a complete set of data for the problem;
- for each block
  - reads the target size and the layout of white squares;
  - verifies whether any correct volley of hits exists and if so, finds one of them;
  - writes the result to the text file **SHO.OUT**;

Input Data

The first line of the input file **SHO.IN** contains the number of data blocks x, $1 \le x \le 5$. The following lines constitute x blocks. The first block starts in the second line of the input file; each next block starts directly after the previous one.

The first line of each block contains two integers r and c separated by a single space, $2 \le r \le c \le 1000$. These are the numbers of rows and columns, respectively. Each of the next c lines in the block contains two integers separated by a single space. The integers in the input line i + 1 in the block,

1 ≤ i ≤ c, are labels of rows with white squares in the i-th column.

Output Data

For the i-th block, 1 ≤ i ≤ x, your program should write to the i-th line of the file **SHO.OUT** either a sequence of c row labels (separated by single spaces) forming a correct volley of hits at white squares in consecutive columns 1, 2, ..., c, or one word NO if such a volley does not exists.

Example:

For the text file `SHO.IN`:

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

one of the correct solutions is the text file `SHO.OUT`:

```
2 3 1 4
NO
```

Final results

Name	Country	Final points
①	*Gold medal*	
Timo Burkard	Germany	*186*
Daniel Adkins	USA	*185*
Krists Boitmanis	Latvia	*170*
Valentin Gheorghita	Romania	*163*
Matt Craighead	USA	*161*
②	*Silver medal*	
Mihai Badoiu	Romania	*155*
Adrian Sox	USA	*148*
Dmitro Nalyvaiko	Ukraine	*142*
Ján Svoreň	Slovakia	*138*
Eryk Kopczyński	Poland	*133*
Alin Sîmpălean	Romania	*132*
Tomasz Waleń	Poland	*129*
Piotr Sankowski	Poland	*128*
Ovidiu Ghiorghioiu	Romania	*126*

③	*Bronz medal*	
Oleg Myrk	Estonia	123
Marius Gedminas	Lithuania	123
Bobby Knock	USA	123
Christian Müller	Germany	117
Arunas Miliunas	Lithuania	117
Andrzej Gąsienica-Samek	Poland	116
Zeljko Svedic	Croatia	115
Janis Sermulins	Latvia	113
Zsolt Felfoldi	Hungary	106
Vladimír Koutný	Slovakia	104
Milan Stanojevich	Yugoslavia	103
Remigijus Staniulis	Lithuania	102
Zoran Majstrovic	Croatia	97
Uwe Bubeck	Germany	96
☹	*No medal*	
Dragan Milenkovich	Yugoslavia	93
Jaroslav Blagojevich	Yugoslavia	91
Kristo Kuuskyll	Estonia	89
Tomas Kaulakys	Lithuania	89
Joost Batenburg	Netherlands	87
Zvonimir Bujanovic	Croatia	83
Renars Gailis	Latvia	81
Michael Schneider	Germany	77
Juris Krikis	Latvia	77
Tanel Linnas	Estonia	76
Sergiy Henkin	Ukraine	76
Sergiy Didenko	Ukraine	74
David Vegh	Hungary	69
Ognjen Arandjelovich	Yugoslavia	69
Richard Králôvič	Slovakia	68
Istvan Marhefka	Hungary	63
Dávid Pál	Slovakia	59
Daniel Varkonyi	Hungary	53
Pawel Kartavtsev	Belarus	47
Dmitry Mitskevich	Belarus	44
Wilke Havinga	Netherlands	44
Andrey Kirilenko	Belarus	39
Alexandre Grishuk	Belarus	36
Ognjen Fabris	Croatia	35
Yuriy Manoylo	Ukraine	34
Kristjan Rummel	Estonia	12

5th Central-European Olympiad in Informatics

Zadar - Croatia, May 20-27., 1998

Day 1 - Problem A

Squares

We are given N squares in the coordinate plane whose sides are parallel to the co-ordinate axes. All the corners have integer coordinates and the squares do not touch or overlap.

You are required to count the number of squares visible from the origin point O, O = (0,0).

A square is **visible** from the origin point O if there are two distinct points A and B on one of its sides such that the interior of the triangle OAB has no common points with any of the remaining squares.

Input Data

The first line of the input file **SQUARES.IN** contains the integer N, 1 ≤ N ≤ 1000, the number of squares.

Each of the following N lines describes a square by specifying integers X, Y and L separated by single blank characters, 1 ≤ X,Y,L ≤ 10000. X and Y are coordinates of the lower left corner (the corner with the least X and Y coordinates) and L is the side length.

Output Data

The first and the only line of the output file **SQUARES.OUT** should contain the number of squares that are visible from the origin.

Examples:

```
SQUARES.IN                         SQUARES.OUT
3                                  3
2 6 3
1 4 1
3 4 1

SQUARES.IN                         SQUARES.OUT
4                                  2
1 2 1
3 1 1
2 4 2
3 7 1
```

Day 1 - Problem B

Cards

Alice and Bob have a set of N cards labelled with numbers 1 ... N (so that no two cards have the same label) and a shuffle machine. We assume that N is an odd integer.

The shuffle machine accepts the set of cards arranged in an arbitrary order and performs the following operation of **double shuffle** : for all positions i, $1 \le i \le N$, if the card at the position i is j and the card at the position j is k, then after the completion of the operation of double shuffle, position i will hold the card k.

Alice and Bob play a game. Alice first writes down all the numbers from 1 to N in some random order: $a_1, a_2, ..., a_N$. Then she arranges the cards so that the position a_i holds the card numbered a_{i+1}, for every $1 \le i \le N-1$, while the position a_N holds the card numbered a_1.

This way, cards are put in some order $x_1, x_2, ..., x_N$, where x_i is the card at the ith position.

Now she sequentially performs S double shuffles using the shuffle machine described above. After that, the cards are arranged in some final order $p_1, p_2, ..., p_N$ which Alice reveals to Bob, together with the number S. Bob's task is to guess the order $x_1, x_2, ..., x_N$ in which Alice originally put the cards just before giving them to the shuffle machine.

Input Data

The first line of the input file **CARDS.IN** contains two integers separated by a single blank character : the odd integer N, $1 \le N \le 1000$, the number of cards, and the integer S, $1 \le S \le 1000$, the number of double shuffle operations.

The following N lines describe the final order of cards after all the double shuffles have been performed such that for each i, $1 \le i \le N$, the (i+1)st line of the input file contains p_i (the card at the position i after all double shuffles).

Output Data

The output file **CARDS.OUT** should contain N lines which describe the order of cards just before they were given to the shuffle machine.

For each i, $1 \le i \le N$, the ith line of the output file should contain x_i (the card at the position i before the double shuffles).

Examples:

```
CARDS.IN                            CARDS.OUT
5 2                                 2
4                                   5
1                                   4
5                                   1
3                                   3
2

CARDS.IN                            CARDS.OUT
7 4                                 4
6                                   7
```

3	5
1	6
2	1
4	2
7	3
5	

Day 1 – Problem C

Subtract

We are given a sequence of N positive integers a = [a_1, a_2, ..., a_N] on which we can perform contraction operations.

One contraction operation consists of replacing adjacent elements a_i and a_{i+1} by their difference a_i-a_{i+1}. For a sequence of N integers, we can perform exactly N-1 different contraction operations, each of which results in a new (N-1) element sequence.

Precisely, let `con(a,i)` denote the (N-1) element sequence obtained from [a_1, a_2, ..., a_N] by replacing the elements a_i and a_{i+1} by a single integer a_i-a_{i+1} :

con(a,i) = [a_1, ..., a_{i-1}, a_i-a_{i+1}, a_{i+2}, ..., a_N]

Applying N-1 contractions to any given sequence of N integers obviously yields a single integer.

For example,

applying contractions `2, 3, 2` and `1` in that order to the sequence `[12,10,4,3,5]` yields 4, since :

```
con([12,10,4,3,5],2) = [12,6,3,5]
con([12,6,3,5]   ,3) = [12,6,-2]
con([12,6,-2]    ,2) = [12,8]
con([12,8]       ,1) = [4]
```

Given a sequence `a1, a2, ..., aN` and a target number `T`, the problem is to find a sequence of `N-1` contractions that applied to the original sequence yields `T`.

Input Data

The first line of the input file **SUBTRACT.IN** contains two integers separated by blank character : the integer N, $1 \leq N \leq 100$, the number of integers in the original sequence, and the target integer T, $-10000 \leq T \leq 10000$.

The following N lines contain the starting sequence : for each i, $1 \leq i \leq N$, the (i+1)st line of the input file contains integer a_i, $1 \leq a_i \leq 100$.

Output Data

Output file **SUBTRACT.OUT** should contain N-1 lines, describing a sequence of contractions that transforms the original sequence into a single element sequence containing only number T. The ith line of the output file should contain a single integer denoting the ith contraction to be applied.

You can assume that at least one such sequence of contractions will exist for a given input.

Examples:

```
SUBTRACT.IN                          SUBTRACT.OUT
4 5                                  1
10                                   2
2                                    1
5
2

SUBTRACT.IN                          SUBTRACT.OUT
5 4                                  2
12                                   3
10                                   2
4                                    1
3
5
```

Day 2 - Problem A

Soldiers

N soldiers of the land Gridland are randomly scattered around the country.

A position in Gridland is given by a pair (x,y) of integer coordinates. Soldiers can move - in one move, one soldier can go one unit up, down, left or right (hence, he can change either his x or his y coordinate by 1 or -1).

The soldiers want to get into a horizontal line next to each other (so that their final positions are (x,y), (x+1,y), ..., (x+N-1,y), for some x and y). Integers x and y, as well as the final order of soldiers along the horizontal line is arbitrary.

The goal is to minimise the total number of moves of all the soldiers that takes them into such configuration.

Two or more soldiers must never occupy the same position at the same time.

Input Data

The first line of the input file **SOLDIERS.IN** contains the integer N, $1 \leq N \leq 10000$, the number of soldiers.

The following N lines of the input file contain initial positions of the soldiers : for each i, $1 \leq i \leq N$, the $(i+1)^{st}$ line of the input file contains a pair of integers x[i] and y[i] separated by a single blank character, representing the coordinates of the i^{th} soldier, $-10000 \leq x[i],y[i] \leq 10000$.

Output Data

The first and the only line of the output file **SOLDIERS.OUT** should contain the minimum total number of moves that takes the soldiers into a horizontal line next to each other.

Examples:

```
SOLDIERS.IN                          SOLDIERS.OUT
3                                    4
1 0
2 4
3 2

SOLDIERS.IN                          SOLDIERS.OUT
5                                    8
1 2
2 2
1 3
3 -2
3 3
```

Day 2 – Problem B

Roads

N cities named with numbers 1 ... N are connected with one-way roads. Each road has two parameters associated with it : the road length and the toll that needs to be paid for the road (expressed in the number of coins).

Bob and Alice used to live in the city 1. After noticing that Alice was cheating in the card game they liked to play, Bob broke up with her and decided to move away - to the city N. He wants to get there as quickly as possible, but he is short on cash.

We want to help Bob to find **the shortest path** from the city 1 to the city N **that he can afford** with the amount of money he has.

Input Data

The first line of the input file **ROADS.IN** contains the integer K, $0 \le K \le 10000$, maximum number of coins that Bob can spend on his way.

The second line contains the integer N, $2 \le N \le 100$, the total number of cities.

The third line contains the integer R, $1 \le R \le 10000$, the total number of roads.

Each of the following R lines describes one road by specifying integers S, D, L and T separated by single blank characters :

- S is the source city, $1 \le S \le N$
- D is the destination city, $1 \le D \le N$
- L is the road length, $1 \le L \le 100$
- T is the toll (expressed in the number of coins), $0 \le T \le 100$

Notice that different roads may have the same source and destination cities.

Output Data

The first and the only line of the output file **ROADS.OUT** should contain the total length of the shortest path from the city 1 to the city N whose total toll is less than or equal K coins.

If such path does not exist, only number -1 should be written to the output file.

Examples:

```
ROADS.IN                                  ROADS.OUT
5                                         11
6
7
1 2 2 3
2 4 3 3
3 4 2 4
1 3 4 1
4 6 2 1
3 5 2 0
5 4 3 2
```

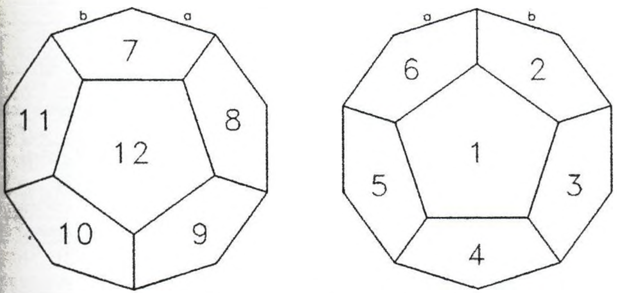
```
ROADS.IN                                  ROADS.OUT
0                                         -1
4
4
1 4 5 2
1 2 1 0
2 3 1 1
3 4 1 0
```

Day 2 - Problem C

Ball

Professor Balltazar is a big football fan. His birthday was just a couple of days before he was going to leave for the World Cup Football '98 in France, so his friends gave him as a present a dodecahedron-shaped puzzle as an entertainment while watching the boring games.

The puzzle has 12 equal pentagon sides, labelled with numbers 1 ... 12. The figure below shows the two hemispheres of the dodecahedron, together with the side labelling we will use in this problem. Hemispheres are "glued" together in such a way that the side 7 is adjacent to the sides 8, 12, 11, 2 and 6 (sides are adjacent if they share an edge). In particular, edges *a* and *b* on the left hemisphere will be glued to the edges *a* and *b* on the right hemisphere shown below.



In addition to that, there are 12 pentagon-shaped tiles, also labelled from 1 to 12. Every edge on each of the tiles is marked with a number from the set {0, 1, 2}. Each tile can be placed on each of the twelve sides in any of the 5 positions obtained by rotating the tile around its centre.

To solve the puzzle, we need to put each tile on some of the twelve dodecahedron sides in some position, so that every two adjacent tiles have their common edge marked with the same number.

Help Professor Balltazar to solve the puzzle !

Input Data

The input file **BALL.IN** contains 12 lines. For each i, $1 \le i \le 12$, the i^th^ line describes the i^th^ tile by specifying 5 numbers from the set {0, 1, 2} separated by single blank characters. This sequence shows the edge marking of the i^th^ tile starting from an arbitrary edge (called the **i^th^ reference edge**) going in the clockwise direction.

Output Data

Output file **BALL.OUT** should contain the description of a solved puzzle in 12 lines with 2 integers in each line. For each i, $1 \le i \le 12$, the i^th^ line should contain integers t[i] and n[i] separated by a single blank character describing the tile and its position on the i^th^ side :

The i^th^ side will hold the tile labelled t[i].

The tile can be placed on the i^th^ side in five different positions. The exact position is specified by n[i], which denotes the label of the adjacent side which is in the direction of the t[i]^th^ reference edge (looking from the centre of the i^th^ side). Precisely, the t[i]^th^ reference edge is placed on the dodecahedron edge shared by the sides labelled i and n[i].

If the puzzle can not be solved, only number -1 should be written to the output file.

Examples:

```
BALL.IN                 BALL.OUT
0 0 1 1 2               1 2
0 2 1 0 1               3 7
2 0 1 0 1               12 4
0 0 1 2 1               7 9
```

```
0 2 1 1 2                    9 1
2 0 1 2 1                    11 8
0 2 1 2 1                    8 2
2 2 1 0 1                    4 6
1 2 2 0 0                    5 4
0 2 1 0 2                    2 12
0 2 1 2 0                    6 3
2 0 1 2 0                    10 7

BALL.IN                      BALL.OUT
1 0 2 0 2                    1 2
2 2 2 1 2                    2 7
1 1 0 0 0                    8 2
1 1 0 2 1                    7 1
2 1 1 1 1                    11 4
1 2 2 1 1                    12 2
2 1 2 2 1                    5 2
2 2 0 1 0                    3 12
0 1 2 1 2                    10 5
2 2 1 0 0                    9 3
1 2 0 2 0                    6 10
2 2 2 0 1                    4 7
```

Final results

Name	Country	Final points
	Gold medal	
Andrej Gasienica-Samek	Poland	*194*
Eryk Kopczynski	Poland	*194*
Jan Senko	Slovak Republic	*191*
Tomasz Czajka	Poland	*185*
②	***Silver medal***	
Jan Valky	Slovak Republic	*164*
Mihai Scortaru	Romania	*162*
MICHAL Forišek	Slovak Republic	*158*
András Förhécz	Hungary	*119*
András Rokob	Hungary	*119*
Angel Stelian Proorocu	Romania	*119*
③	***Bronz medal***	
Zsolt Felföldi	Hungary	*106*
Tudor Ioan Leu	Romania	*104*
Diodor Bitan	Romania	*103*
Jens Zumbragel	Germany	*95*
Zvonimir Bujanovič	Croatia	*94*
Simon Giesecke	Germany	*91*
Balázs Rácz	Hungary	*91*
Zoran Majstrovič	Croatia	*89*
Pawel Wolff	Poland	*85*
Milan Mesič	Croatia	*73*

☹	*No medal*	
Tobias Thierer	Germany	*65*
Alexis Gaulke	Germany	*63*
Jan Lunter	Slovak Republic	*61*
Krunoslav Funtak	Croatia II	*51*
Zdenek Dvorak	Czech Republic	*51*
Jure Leskovec	Slovenia	*42*
Dean Čupovič	Bosnia and Herzegovina	*39*
Krešimir Ćosič	Croatia	*37*
Matija Kazalicki	Croatia II	*37*
Jiri Benc	Czech Republic	*37*
Ognjen Fabris	Croatia II	*24*
Matej Rizman	Slovenia	*24*
Pavel Nejedly	Czech Republic	*22*
Marko Grbič	Zadar	*18*
Admir Tuzovič	Bosnia and Herzegovina	*17*
Tarik Karamustafič	Bosnia and Herzegovina	*12*
Jaromir Malenko	Czech Republic	*10*
Franjo Posilovič	Croatia II	*9*
Ivan Zorič	Zadar	*8*
Esad Hajdarevič	Bosnia and Herzegovina	*7*
Andraž Tori	Slovenia	*3*
Gašper Fele-Žorž	Slovenia	*0*
Dario Mandir	Zadar	*0*
Branimir Putnikovič	Zadar	*0*

6th Central-European Olympiad in Informatics

Brno – Czech Republic, September 2-9., 1999

Day 1 – Problem A

Orders

The stores manager has sorted all kinds of goods in an alphabetical order of their labels. All the kinds having labels starting with the same letter are stored in the same warehouse (i.e. in the same building) labelled with this letter. During the day the stores manager receives and books the orders of goods which are to be delivered from the store. Each order requires only one kind of goods. The stores manager processes the requests in the order of their booking.

You know in advance all the orders which will have to be processed by the stores manager today, but you do not know their booking order. Compute all possible ways of the visits of warehouses for the stores manager to settle all the demands piece after piece during the day.

Input Data

Input file **ORDERS.IN** contains a single line with all labels of the requested goods (in random order). Each kind of goods is represented by the starting letter of its label. Only small letters of the English alphabet are used. The number of orders doesn't exceed 200.

Output Data

Output file **ORDERS.OUT** will contain all possible orderings in which the stores manager may visit his warehouses. Every warehouse is represented by a single small letter of the English alphabet – the starting letter of the label of the goods. Each ordering of warehouses is written in the output file only once on a separate line and all the lines containing orderings have to be sorted in an alphabetical order (see the example). No output will exceed 2 megabytes.

Example:

```
ORDERS.IN
bbjd

ORDERS.OUT
bbdj
bbjd
bdbj
bdjb
bjbd
bjdb
dbbj
dbjb
```

```
djbb
jbbd
jbdb
jdbb
```

Day 1 - Problem B

Phone numbers

In the present world you frequently meet a lot of call numbers and they are going to be longer and longer. You need to remember such a kind of numbers. One method how to do it in an easy way is to assign letters to digits as shown in the following picture:

1 *i j*	*2* *a b c*	*3* *d e f*
4 *g h*	*5* *k l*	*6* *m n*
7 *p r s*	*8* *t u v*	*9* *w x y*
	0 *o q z*	

This way every word or a group of words can be assigned a unique number, so you can remember words instead of call numbers. It is evident that it has its own charm if it is possible to find some simple relationship between the word and the person itself. So you can learn that the call number 941837296 of a chess playing friend of yours can be read as WHITEPAWN, and the call number 2855304 of your favourite teacher is read BULLDOG.

Write a program to find the shortest sequence of words (i.e. one having the smallest possible number of words) which corresponds to a given number and a given list of words. The correspondence is described by the picture above.

Input Data

The first line of input file **PHONE.IN** contains the call number, the transcription of which you have to find. The number consists of at most 100 digits. The second line contains the total number of the words in the dictionary (maximum is 50000). Each of the remaining lines contains one word, which consists of maximally 50 small letters of the English alphabet. The total size of the input file doesn't exceed 300KB.

Output Data

The only line of output file **PHONE.OUT** contains the shortest sequence of words which has been found by your program. The words are separated by single spaces. If there is no solution to the input data, the line contains text `No solution.'. If there are more solutions having the minimum number of words, you can choose any single one of them.

Example:

```
PHONE.IN
7325189087
5
it
your
reality
real
our

PHONE.OUT
reality our
```

(next possibility 'real it your' corresponding to the same number is longer).

```
PHONE.IN
4294967296
5
it
your
reality
real
our

PHONE.OUT
No solution.
```

because no given word contains letters g and h which are necessary to obtain the digit 4.

Day 1 - Problem C

Parity game

Now and then you play the following game with your friend. Your friend writes down a sequence consisting of zeroes and ones. You choose a continuous subsequence (for example the subsequence from the third to the fifth digit inclusively) and ask him, whether this subsequence contains even or odd number of ones. Your friend answers your question and you can ask him about another subsequence and so on. Your task is to guess the entire sequence of numbers.

You suspect some of your friend's answers may not be correct and you want to convict him of falsehood. Thus you have decided to write a program to help you in this matter. The program will receive a series of your questions together with the answers you have received from your friend. The aim of this program is to find the first answer which is provably wrong, i.e. that there exists a sequence satisfying answers to all the previous questions, but no such sequence satisfies this answer.

Input Data

The first line of input file **PARITY.IN** contains one number, which is the length of the sequence of zeroes and ones. This length is less or equal to 1000000000. In the second line, there is one positive integer which is the number of questions asked and answers to them. The number of questions and answers is less or equal to 5000. The remaining lines specify questions and answers. Each line contains one question and the answer to this question: two integers (the position of the first and last digit in the chosen subsequence) and one word which is either `even' or `odd' (the answer, i.e. the parity of the number of ones in the chosen subsequence, where `even' means an even number of ones and `odd' means an odd number).

Output Data

There is only one line in output file **PARITY.OUT** containing one integer X. Number X says that there exists a sequence of zeroes and ones satisfying first X parity conditions, but there exists none satisfying X+1 conditions. If there exists a sequence of zeroes and ones satisfying all the given conditions, then number X should be the number of all the questions asked.

Example 1:

```
PARITY.IN                      PARITY.OUT
10                             3
5
1 2 even
3 4 odd
5 6 even
1 6 even
7 10 odd
```

Example 2:

```
PARITY.IN                      PARITY.OUT
10                             5
5
1 2 even
1 4 even
2 4 odd
1 10 even
3 10 even
```

Day 2 - Problem A

Sightseeing trip

There is a travel agency in Adelton town on Zanzibar island. It has decided to offer its clients, besides many other attractions, sightseeing the town. To earn as much as possible from this attraction, the agency has accepted a shrewd decision: it is necessary to find the shortest route which begins and ends at the same place. Your task is to write a program which finds such a route.

In the town there are N crossing points numbered from 1 to N and M two-way roads numbered from 1 to M. Two crossing points can be connected by multiple roads, but no road connects a crossing point with itself. Each sightseeing route is a sequence of road numbers y_1, ..., y_k, $k>2$. The road y_i ($1 \le i \le k-1$) connects crossing points xi and x_{i+1}, the road y_k connects crossing points x_k and x_1. All the numbers $x_1,...,x_k$ should be different.

The length of the sightseeing route is the sum of the lengths of all roads on the sightseeing route, i.e. $L(y_1)+L(y_2)+...+L(y_k)$ where $L(y_i)$ is the length of the road y_i ($1 \le i \le k$). Your program has to find such a sightseeing route, the length of which is minimal, or to specify that it is not possible, because there is no sightseeing route in the town.

Input Data

The first line of input file **TRIP.IN** contains two positive integers: the number of crossing points $N \le 100$ and the number of roads $M \le 10000$. Each of the next M lines describes one road. It contains 3 positive integers: the number of its first crossing point, the number of the second one, and the length of the road (a positive integer less than 500).

Output Data

There is only one line in output file **TRIP.OUT**. It contains either a string 'No solution.' in case there isn't any sightseeing route, or it contains the numbers of all crossing points on the shortest sightseeing route in the order how to pass them (i.e. the numbers x_1 to x_k from our definition of a sightseeing route), separated by single spaces. If there are multiple sightseeing routes of the minimal length, you can output any one of them.

Example 1:

```
TRIP.IN                                 TRIP.OUT (one of correct answers)
5 7                                     1 3 5 2
1 4 1
1 3 300
3 1 10
1 2 16
2 3 100
2 5 15
5 3 20
```

Example 2:

```
TRIP.IN                                 TRIP.OUT (the only correct answer)
4 3                                     No solution.
1 2 10
1 3 20
1 4 30
```

Day 2 - Problem B

A Game on the Chessboard

On the chessboard of size 4x4 there are 8 white and 8 black stones, i.e. one stone on each field. Such a configuration of stones is called a game position. Two stones are adjacent if they are on fields with a common side (i.e. they are adjacent in either horizontal or vertical direction but not diagonal). It means that each stone has at most 4 neighbours. The only legal move in our game is exchanging any two adjacent stones. Your task is to find the shortest sequence of moves transforming a given starting game position into a given final game position.

Input Data

The starting game position is described in the first 4 lines of input file **GAME.IN**. There are 4 symbols in each line, which define the colour of each stone in the row from the left to, the right. The lines describe the rows of the chessboard from the top to the bottom. Symbol `0' means a white stone and symbol `1' a black one. There is no space symbol separating the symbols in the line. The fifth line is empty. The next 4 following lines describe the final game position in the same way.

Output Data

The first line of output file **GAME.OUT** contains the number of the moves. The following lines describe the sequence of moves during the game. One line describes one move and it contains 4 positive integers (R_1, C_1, R_2, C_2) separated by single spaces. These are the coordinates of the adjacent fields for the move, i.e. fields (R_1, C_1) and (R_2, C_2), where R_1 (or R_2) is the number of the row and C_1 (or C_2) is the number of the column. The rows on the chessboard are numbered from 1 (top row) to 4 (bottom row) and the columns are numbered from 1 (the leftmost column) to 4 (the rightmost one) as well (i.e. the coordinates of the left upper field are [1,1]). If there are multiple shortest sequences of moves transforming the starting position to the final position, you can output any one of them.

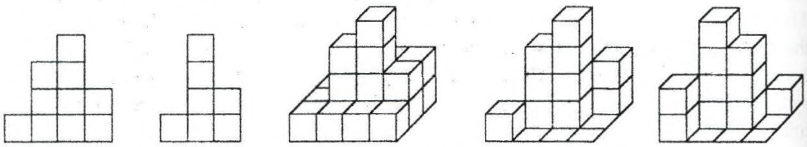
Example:

```
GAME.IN                    GAME.OUT (one of the correct solutions)
1111                       4
0000                       1 2 2 2
1110                       1 4 2 4
0010                       3 2 4 2
                           4 3 4 4
1010
0101
1010
0101
```

Day 2 - Problem C

Block town

Children like playing with blocks (cube wooden bricks). They usually build high towers, but small Johny dreams of different plans. He is going to build a big town. His daddy has bought him a rectangular table; its width is K blocks and its length is L blocks exactly. Johny decided to project a plan of such a town before he starts building the town itself. He has drawn a square-shaped network on the table consisting of KxL squares. He wants to place the towers consisting of one or more blocks on some of the squares of the network drawn; the remaining squares will be empty. Because of the table being so large, Johny is not going to plan exactly for every square how many blocks he will put on it. He only wants to decide about front and right sight shapes of his town. He drew these two views (two-dimensional projections of the planned town) on a paper. You can see an example of these drawings and the adequate town made of wooden bricks in the pictures:


See the figures from the left: side view, front view, maximal town, minimal town (both front and back view)!

Johny's daddy is afraid they don't have enough blocks to finish building Johny's planned town. You are asked for writing a program to compute the minimal and maximal amount of blocks with which a town corresponding to Johny's plans can be built. Moreover the program can decide about the possibility of building a town satisfying the views.

Input Data

The first line of input file **TOWN.IN** contains two positive integers K, L – the width and the length of the table (expressed as numbers of bricks). Neither the width nor the length of the table is greater than 100000 bricks. The following lines of the input file contain the description of the front view of the town. The description consists of a series of heights of visible buildings on each square from the left to the right (the height is measured by the number of the blocks, too). There is only one number on each line, i.e. the number of the lines with the front view description of the town equals K – the width of the table. Similarly the next L lines of the input file contain the right sight view of the town. The heights of the wooden block towers are now specified from the front line to the back line. You may suppose there is no building in the town with height exceeding 5000 blocks. The maximal number of blocks needed for building the entire town does not exceed 2000000000.

Output Data

Output file **TOWN.OUT** contains only one line. If it is not possible to build a town with the front and right sight views given, only a text `No solution.' is written there. In the other case two numbers will be written on the line and separated by a single space. The first one is the minimal and the second one is the maximal amount of blocks small Johny can use to build his town in accordance with his project.

Example 1:

```
TOWN.IN
4 3
1
3
4
```

```
2
1
4
2

TOWN.OUT
10 21
```

Example 2:

```
TOWN.IN
2 2
4
1
1
3

TOWN.OUT
No solution.
```

Final results

Name	Country	Final points
①	*Gold medal*	
Andrzej Gasienica-Samek	Poland	*200*
Mihai Patrascu	Romania	*187*
Radu Stefan	Romania	*187*
Daniel Wright	USA	*172*
②	***Silver medal***	
Bogdan Dumitru	Romania	*168*
Benjamin Mathews	USA	*167*
Krzysztof Onak	Poland	*164*
Pavel Charvat	Czech Republic B	*160*
Bogdan Batog	Romania	*160*
Pavel Nejedly	Czech Republic A	*158*
Andras Rokob	Hungary	*153*
Tobias Thierer	Germany	*148*
③	***Bronz medal***	
Dominik Schultes	Germany	*141*
Jakub Bystron	Czech Republic	*135*
Michal Nowakiewicz	Poland	*134*
Percy Liang	USA	*129*
Zdenek Dvorak	Czech Republic A	*126*
Jiri Svoboda	Czech Republic B	*125*
Marcin Meinardi	Poland	*125*
Mario Zivic	Croatia	*123*
Jan Oravec	Slovakia	*120*
Frane Saric	Croatia	*116*
Ádám Novák	Hungary	*116*

	David Pal	Slovakia	*113*
☹		***No medal***	
	Christian Hett	Germany	*110*
	Josef Zlomek	Czech Republic A	*109*
	Vladimir Koutny	Slovakia	*109*
	Michal Illich	Czech Republic A	*108*
	Szilvester Safar	Hungary	*99*
	Michal Breznicky	Slovakia	*96*
	Domagoj Vlah	Croatia	*94*
	Michael Kreil	Germany	*88*
	Andras Tori	Slovenia	*87*
	David Cheng	USA	*83*
	Matija Kazalicki	Croatia	*82*
	Kristof Csillag	Hungary	*77*
	Ondrej Rucky	Czech Republic B	*75*
	Kamil Sevecek	Czech Republic J	*58*
	Pavel Simecek	Czech Republic J	*57*
	Esad Hajdarevic	Bosnia-Herczegovina	*46*
	Jiri Cvachovec	Czech Republic J	*41*
	Rihad Seherac	Bosnia-Herczegovina	*36*
	Estela Kunalic	Bosnia-Herczegovina	*35*
	Jure Leskovec	Slovenia	*30*
	Blas Novak	Slovenia	*27*
	Mojca Miklavec	Slovenia	*19*
	Bojan Hrnkas	Bosnia-Herczegovina	*16*
	Vaclav Fiala	Czech Republic J	*4*

7th Central-European Olympiad in Informatics

Cluj – Romania, August 24-31., 2000

Day 1 - Problem A

X-Planet

All the inhabitants of the planet X build their houses in a triangular shape. In order to save time and effort, they use a special construction method. Everything starts with a straight wall. After that, for the construction of every new house they just add two new walls to an already existing wall, thus resulting in a closed, triangular shaped house. Of course, the two new added walls might also be used as starting walls for new houses. Sometimes, using this construction pattern, they arrive at situations where some houses are completely enclosed in others (like in the figure). However, this is not a problem, because kids might live in the included houses.

To light up their houses, the X-habitants install a light bulb on every corner of the final construction (a bulb in a corner is common to all the houses sharing that corner). On each corner, there is a switch whose touch toggles the state (on/off) of the bulb from both that corner and also that of all the bulbs of the connected corners. Two corners are considered connected if they are placed at the end of the same wall.

Task

Write a program that finds a switch touch sequence that will turn on all the light bulbs, starting from a given state.

Input Data

File name: **X.IN**

Line 1: N

- an integer, the number of corners of the building, labelled from 1 to N;

Lines 2..2×N – 2: I J

- Two integers, separated by a blank, representing a wall between the corners I and J;

Line 2×N – 1: S_1 S_2 ... S_N

- N integers (separated by blanks) whose values are 0's or 1's, corresponding to the state of the N light bulbs;
- 0 means off; 1 means on.

The input data are guaranteed to represent a building that can be constructed in the specified pattern.

Output Data

File name: **X.OUT**

Line 1: L_1 L_2 ... L_K

- K integers, separated by blanks, representing the list of the labels of the switches to be touched.
- If there are several solutions, only one is required.
- If there is no possible solution, you should write in the output file a single line containing the number 0.

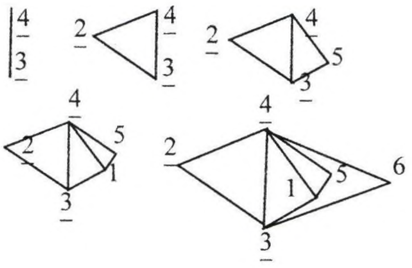
Limits

- 3 ≤ N ≤ 1000

Example:

```
X.IN                    X.OUT
6                       1 6
1 3
1 4
1 5
2 3
2 4
3 4
3 6
4 5
4 6
0 1 1 1 0 0
```

In the figure below you can see a possible building steps for the example file; the labels for the bulbs initially illuminated are underlined.



Day 1 - Problem B

Roads

The Romanian Ministry of Transport decides to upgrade the Romanian roads. Each road is bidirectional and directly connects two towns. No two towns are directly connected by more than one road. The existing road network ensures direct or indirect links between any two towns in Romania.

However, upgrading the roads implies closing, in turn, each road and performing the necessary repairs. During these operations, it is necessary to preserve the road network connectivity. In order to do so, the Minister decides that new roads should be built, so that no matter which single road is closed at any given moment (exactly one road at a time), the traffic between any given pair of towns should still be possible. Of course, the number of these newly added roads should be minimized. Furthermore, no new road may directly connect two directly connected towns.

Task

Write a program that determines the minimum number of the additional roads to be built and the pairs of towns to be connected with them. If there are several optimal solutions, only one is required.

Input Data

File name: **ROADS.IN**

Line 1: N M

- two integers, separated by a blank, representing respectively the number of the towns and the number of the roads. The towns are labelled from 1 to N.

Lines 2..M+1: T_1 T_2

- two integers separated by a blank, representing the two towns connected by a road.

Output Data

File name*:* **`ROADS.OUT`**

Line 1: `K`

- an integer, representing the minimum number of additional roads.

Lines 2..K+1: C_1 C_2

- two integers separated by a blank, representing a pair of towns (cities) to be connected by a road.

Remark

- the order of town pairs in the output file is irrelevant.

Limits

- $3 \le N \le 2500$
- $2 \le M \le 20000$

Example:

```
ROADS.IN                        ROADS.OUT
4 3                             2
1 2                             1 4
2 3                             1 3
2 4
```

Day 1 – Problem C

The Caterpillar

Definition: A **caterpillar** is a particular kind of tree with the following property: there is a central chain such that each of the tree's nodes lie either on the central *chain*, or they are adjacent to that chain. Below there are shown two caterpillars. The darkened nodes indicate the chain.

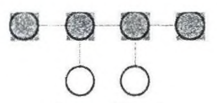

Caterpillar 1

Figure 1

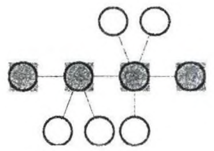

Caterpillar 2

Figure 2

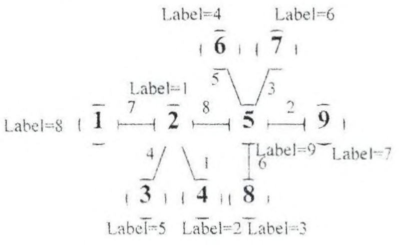
The chain is not necessarily unique. For example, another possible chain for the second caterpillar is 3-2-5-9.

Task

Given a caterpillar with N nodes, write a program which labels the nodes with numbers from 1 to N such that:

- each label from 1 to N is used exactly once;
- no two edges in the tree have the same absolute value of the difference between the labels of their adjacent nodes.

For the second caterpillar above, a possible labeling is indicated below along with the corresponding differences on the edges:



Input Data

File name: **CP.IN**

Line 1: N

- one integer, the number of nodes;

Lines 2..N: X Y

- two integers, separated by a blank, which are two adjacent nodes connected by an edge.

The input data is correct, and the input tree is a caterpillar.

Output Data

File name: **CP.OUT**

Line 1: **L1 L2 ... LN**

- N integers, separated by blanks, where Li represents the label associated with the node i.

If there are multiple solutions, only one is required. If no labeling is possible, the output file should contain only one line with the word: IMPOSSIBLE

Limits

- $2 \leq N \leq 10000$

Example:

The following input (CP.IN) describes the caterpillar from the Figure 2 and the output (CP.OUT) the labeling from the Figure 3.

```
CP.IN
9
1 2
6 5
5 7
4 2
2 3
8 5
2 5
5 9
```

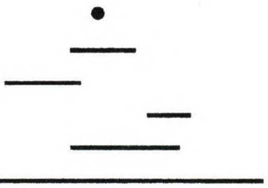
CP.OUT (one possible solution)

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

Day 2 – Problem A

Falling

Consider a game based on the device shown in the image below:



The device consists in a set of horizontal platforms of various lengths, placed at various heights. The lowest platform is the floor (it is placed at height 0 and has an infinite length).

From a given position, a ball is released in free fall at moment 0. The ball falls at a constant speed of 1 meter per second. When the ball reaches a platform it starts rolling towards one or the other of its edges, at the choice of the player, at the same speed of one meter per second. When it reaches the edge of the platform, it continues its vertical free fall. The ball is not allowed to free fall for more then MAX meters at one time (between two platforms).

Task

Write a program that finds a way to roll the ball on the platforms so that it reaches the floor as soon as possible without breaking.

Input Data

File name: **FALL.IN**

Line 1: N X Y MAX

- Four integers, separated by a space: the number of platforms (excluding the floor), the starting position of the ball (horizontal and vertical coordinates), and the maximum allowed free fall distance; the platforms are numbered from 1 through N.

Lines 2..N+1: $X1_i$ $X2_i$ H_i

- Three integers, separated by spaces; the i-th platform is situated at height H_i, between horizontal coordinates $X1_i$ and $X2_i$ inclusive ($X1_i < X2_i$, i=1..N).

Remarks:

- Ignore the ball diameter and platform thickness. If the ball falls exactly on the edge of a platform, it is considered a fall onto that platform.
- No two platforms have points in common.
- There will always be a solution for the test data.
- All the dimensions are given in meters.

Output Data

File name: **FALL.OUT**

Line 1: TIME

- An integer: the time when the ball touches the floor, according to your solution.

The rest of the lines, to the end of the file:

P T D

- Three integers, separated by spaces. The ball touches the platform P at moment T and rolls in direction D (0 for left and 1 for right).
- The impact with the floor must not appear in these lines.

  The impacts must be output such that the successive values of T appear in increasing order.

Remark

- If there are several solutions possible, only one is required.

Limits

- $1 \le N \le 1000$
- $-20000 \le X1_i, X2_i \le 20000$ (i=1..N)
- $0 < H_i < Y \le 20000$

Example:

FALL.IN	FALL.OUT
3 8 17 20	23
0 10 8	2 4 1
0 10 13	1 11 1
4 14 3	3 16 1

Day 2 - Problem B

Sticks

Consider the following game between two players:

N rows of sticks lie on a table with S_i sticks in row i ($1 \le i \le N$). The sticks in each row i are labeled sequentially from 1 to S_i. The players alternately make a move. Each move consists of eliminating one, two or three sticks from the same row. These sticks must be labeled sequentially, i.e., they must be contiguous in the row.

For example, if there is a row with 10 sticks and the first player eliminates the sticks labeled 4, 5, 6, only sticks 1, 2, 3, 7, 8, 9, and 10 remain. The second player, on his turn, has the possibility to eliminate the sticks 1, 2, 3 but not 3, 7, 8 since they are not numbered sequentially (of course, there are other valid moves).

The winner is the player performing the last move, i.e., eliminating the last stick(s) from the table.

Task

Write a program that implements a winning strategy, playing against a cyber-opponent.

Input Data

File name: **STICKS.IN**

Line 1: N

- An integer, the number of stick rows ($1 \leq N \leq 10$).

Line 2: S_1 S_2 ... S_N

- N integers, separated by blanks, the number of sticks in each row ($1 \leq S_i \leq 10$).

Line 3: X

- The only possible values for X are 0 and 1. If X=0, your program will make the first move; otherwise, the cyber-opponent will make the first move.

Remark

The test input data guarantees that a winning strategy exists for your program.

Your program will play against a machine. The interaction between your program and its opponent will be made possible by the following software interface:

- Pascal: STICKS unit
- C/C++: sticks.h header

with the following routines:

```
procedure putMove(nr,label1,label2:Integer);          (Pascal)
void putMove(int nr, int label1, int label2);         (C/C++)
```

– *called to announce your program's move, that is, "eliminate from row `nr` the sticks labeled with numbers from `label1` to `label2` inclusively"*

```
procedure getMove(var nr,label1,label2:Integer);      (Pascal)
void getMove(int *nr, int *label1, int *label2);      (C/C++)
```

– *called to obtain your opponent's move, in the same format as above;*
C/C++ only: pass pointers to the variables in which you want to obtain the data

Your program should alternately call these two routines until there are no more sticks on the table. Our module will terminate the game if your program makes an invalid move (of course you will get no points in this case).

Example: **possible call succession in PASCAL**

```
STICKS.IN     getMove(nr, k1, k2); -> nr=2, k1=2, k2=2
2             putMove(1, 1, 1);
```

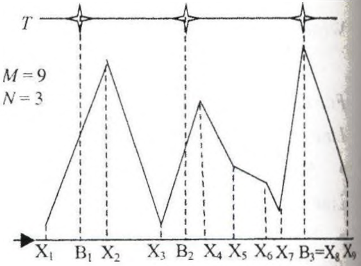
```
1 3             getMove(nr, k1, k2); -> nr=2, k1=1, k2=1
1               putMove(2, 3, 3);
                **** your program wins ****
```

Day 2 – Problem C

Enlightened landscape

Consider a landscape composed of connected line segments

Above the landscape, N light bulbs are hang at the same height T in various horizontal positions. The purpose of these light bulbs is to light up the entire landscape. A landscape point is considered lit if it can "see" a light bulb directly, that is, if the line segment which links the point with a bulb does not contain any other landscape segments point.



Task

Write a program that determines the minimum number of light bulbs that must be switched on in order to illuminate the entire landscape.

Input Data

Input file name: **LIGHT.IN**

Line 1: M

- An integer, the number of landscape height specifications, including the first and the last point of the landscape.

Lines 2..M+1: X_i H_i

- Two integers, separated by a space: the landscape height H_i at horizontal position X_i, $1 \le i \le M$; for $1 \le i \le M-1$ we have $X_{i+1} > X_i$; any two consecutive specified points identify a segment of line in the landscape.

Line M+2: N T

- Two integers, separated by a space, the number of light bulbs and their height coordinate (altitude). The bulbs are numbered from 1 to N)

Line M+3: B_1 B_2 ... B_N

- N integers, separated by spaces: the horizontal coordinates of the light bulbs $B_{i+1} > B_i$, $1 \le i \le N-1$;

Output Data

File name: **LIGHT.OUT**

Line 1: K

- An integer: the minimum number of light bulbs to be switched on.

Line 2: L_1 L_2 ... L_K

- K integers, separated by spaces: the labels of the light bulbs to be switched on specified in increasing order of their horizontal coordinates.

Limits

- $1 \leq M \leq 200$
- $1 \leq N \leq 200$
- $1 \leq X_i \leq 10000$ for $1 \leq i \leq M$
- $1 < T \leq 10000$
- $1 \leq H_i \leq 10000$ for $1 \leq i \leq M$
- $T > H_i$ for any $1 \leq i \leq M$
- $X_1 \leq B_1$ and $B_N \leq X_M$
- The task always has a solution for the test data. If there are multiple solutions, only one is required.

Example:

```
LIGHT.IN                         LIGHT.OUT
6                                2
1 1                              1 4
3 3
4 1
7 1
8 3
11 1
4 5
1 5 6 10
```

Final results

Name	Country	Final points
Reid Barton	USA	*490*
Radu Stefan	Romania	*472*
Tomasz Czajka	Poland	*421*
②	*Silver medal*	
Jan Oravec	Slovakia	*362*
Marian Dvorsky	Slovakia	*356*
Mihai Patrascu	Romania	*351*
Denis Bogdanas	Moldova	*328*
Florin Ghetu	Romania	*282*
Ioana Ileana	Romania	*275*
Tomasz Malesinski	Poland	*270*
Angel Proorucu	Romania	*260*
Peter Pallos	Hungary	*245*
③	*Bronz medal*	
Jozef Tvarozek	Slovakia	*235*
John Danaher	USA	*228*
Daniel Dumitran	Romania	*222*

	Daniel Jaspaer	Germany	*204*
	Dominic Battre	Germany	*200*
	Arcadie Cracan	Romania	*200*
	Grzegorz Stelmaszek	Poland	*197*
	Percy Liang	USA	*194*
	Gabor Gyebnár	Hungary	*183*
	Marius Andrei	Romania	*183*
	Mario Zivic	Croatia	*180*
	Michael Siepmann	Germany	*180*
	Krzysztof Onak	Poland	*180*
☹		***No medal***	
	Tomas Zathurecky	Slovakia	*173*
	Vincent Groenhuis	The Netherlands	*170*
	Andras Rokob	Hungary	*169*
	Liviu Paunescu	Romania	*162*
	Stefan de Koning	The Netherlands	*159*
	Miroslav Trmac	Czech Republic	*156*
	Ivo List,	Slovenia	*156*
	Dumitru Codreanu	Moldova	*124*
	Corneliu Margine	Moldova	*124*
	Adam Novak	Hungary	*121*
	Alexandru Popa	Romania	*121*
	Daniel Oprean	Romania	*118*
	Gregory Price	USA	*114*
	Roman Krejcik	Czech Republic	*111*
	Mojca Miklavec	Slovenia	*104*
	Damir Tubin	Croatia	*100*
	Pavel Cizek	Czech Republic	*97*
	Vlad Dascalu	Romania	*94*
	Andraz Tori	Slovenia	*77*
	Frane Saric	Croatia	*59*
	Blaz Fortuna	Slovenia	*45*
	Damir Korencic	Croatia	*35*
	Martin Beranek	Czech Republic	*21*
	Andrei Bejenari	Moldova	*21*
	Danny Oude Bos	The Netherlands	*21*
	Martin Trautmann	Germany	*14*

8th Central-European Olympiad in Informatics

Zalaegerszeg – Hungary, August 10-17., 2001

Day 1 - Problem A

Chain of Atoms

Problem

Scientists are investigating a special kind of molecule. It is known that the molecule consists of *N* different atoms, labeled from 1 to *N*, and the structure of the molecule is a linear chain. The scientists are equipped with a distance meter. At one time, this instrument can measure the distance between any two atoms in the molecule. The distance, reported by the instrument, is the absolute value of the difference of the two atom positions in the chain. If you access no atom for more than three times no damage is done to the molecule. If you access one or more atoms four times the molecule gets damaged. If you access one or more atoms five times the molecule gets destroyed.

You are to write a program that discovers the complete structure of the molecule by determining the sequence of the atoms in the molecule.

Atoms with labels:

Positions:

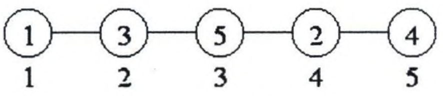


Figure 1

LIBRARY

To operate the distance meter, you are given a library `meter` with three operations:

- `Size`, to be called once at the beginning, without arguments; it returns the value of *N*. `Size` must be called before any call to `Span`.
- `Span`, to be called with two atom labels as arguments; it returns the distance between the atoms.
- `Answer`, to be called at the end, must be used to submit your solution. `Answer` has two integer arguments, *i* and *x*, and it tells that the label of the atom at position *i* in the molecule is *x*. For each *i* ($1 \le i \le N$), `Answer` must be called exactly once in *ascending* order of *i*. There are always two symmetric solutions, and you can submit any one of them.

The dialogue between your program and the library procedures is logged in the text file **CHAIN.OUT**. This log file also contains your answer and the error message in case of an error.

for Pascal programmers: include the import statement

```
    uses meter;
```

in the source code.

Instructions for C/C++ programmers: use the instruction

```
#include "meter.h"
```

in the source code, create a project file `chain.gpr` in the task directory, add the files `chain.c` (`chain.cpp`) and `meter.o` into this project, and then *compile* and/or *make* your program.

Experimentation

You can experiment with the library by creating a text file **CHAIN.IN**. The file must contain two lines. The first line should contain the integer *N*, the number of atoms. The second line should contain a test case: a sequence of the atom labels, consisting of *N* different integers between 1 and *N*.

Example for experimentation:

CHAIN.IN
```
5
1 3 5 2 4
```

CHAIN.OUT
```
Size=5
Span(1,2)=3
Span(1,3)=1
Span(2,3)=2
Span(4,5)=2
Span(1,4)=4
Span(3,5)=1
Your Answer
1 3 5 2 4
Max. Access Atom:
3
```

Constraints

- For the number of atoms, *N*, we have $5 \le N \le 10000$.
- Accessing any atom by `Span` more than four times will abort your program.
- Your program must not read or write any files.
- For the atom labels *x* and the atom positions *i*, we have $1 \le i, x \le N$.
- FreePascal library file names: `meter.ppw` and `meter.ow`.
- Pascal function declarations:
```
    function Size: integer;
    function Span(x, y: integer):integer;
    procedure Answer(i, x: integer);
```
- C/C++ library file names: `meter.h` and `meter.o`.
```
    int Size(void);
    int Span(int x, int y);
    void Answer(int i, int x);
```

Grading

If your answer is correct and no atom was accessed more than *three* times then you obtain full score. If your answer is correct but some atom was accessed *four* times then you obtain 50% of the scores. If the answer is incorrect or any atom was accessed *five* times, then you receive 0 score.

Day 1 - Problem B

Printed Circuit

Problem

A *printed circuit* is a board that consists of *nodes* and *wire segments* connecting pairs of nodes. We consider a special kind of printed circuits where the nodes are arranged in a rectangular grid, and the wire segments connect (vertically or horizontally) adjacent nodes only. A printed circuit is called *connected* if any two distinct nodes are connected with a series of wire segments. Given is a printed circuit where wire segments already connect several adjacent nodes. We have to add new wire segments in order to make the printed circuit connected. The cost of a new wire segment is 1 if it is vertical and 2 if it is horizontal in the grid.

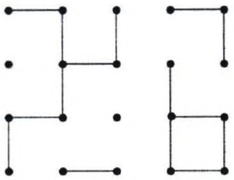


Figure 1

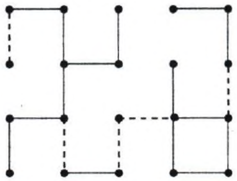


Figure 2

You are to write a program that computes a least cost completion of a given circuit. Your program should solve the following three subtasks:

A. Determine the number of new wire segments of a least cost completion.

B. Compute the value of the least cost.

C. Produce a list of wire segments of a least cost completion.

Input Data

The first line of the file **CIRCUIT.IN** contains two integers, ***N*** and ***M***. ***N*** ***(1≤N≤100)*** is the number of rows and ***M*** ***(1≤M≤100)*** is the number of columns in the grid. The nodes of the circuit are identified by their coordinates; the node in the upper left corner is at ***(1,1)***, and the node in the lower right corner is at ***(N, M)***. Each of the next ***N*** lines contains ***M*** integers. The number in row ***i*** and column ***j*** of these lines describes the wire segments between the pair of nodes ***(i, j)*** and ***(i+1, j)***, and also between the pair of nodes ***(i, j)*** and ***(i, j+1)*** in the following way.

- 0 means that neither the pair of nodes ***(i, j)*** and ***(i+1, j)*** nor the pair of nodes ***(i, j)*** and ***(i, j+1)*** are connected by wire segments.
- 1 means that only the pair of nodes ***(i, j)*** and ***(i+1, j)*** is connected by a wire segment.
- 2 means that only the pair of nodes ***(i, j)*** and ***(i, j+1)*** is connected by a wire segment.
- 3 means that both the pair of nodes ***(i, j)*** and ***(i+1, j)***, and the pair of nodes ***(i, j)*** and ***(i, j+1)*** are connected by wire segments.

Note that not all 4 values are valid at certain positions (e.g. at position (***N***, ***M***) only value 0 is valid).

Output Data

The first line of the file **`CIRCUIT.OUT`** should contain two integers, ***K*** and ***V***. The integer ***K*** is the number of new wire segments of a least cost completion; and the integer ***V*** is the value of the least cost. The rest of the file must contain ***K*** lines; the list of wire segments of a least cost completion (in arbitrary order). Each line must contain three numbers describing exactly one new wire segment: ***i, j*** and ***d***, where ***(i, j)*** are the coordinates of a node, and ***d*** is either 1 or 2. 1 means that the new wire segment connects the nodes ***(i, j)*** and ***(i+1, j)***, and 2 means that the new wire segment connects the nodes ***(i, j)*** and ***(i, j+1)***

Example input and output:

The example input file corresponds to the circuit in Figure 1. The example output file is a possible least cost completion, and is depicted in Figure 2.

```
CIRCUIT.IN          CIRCUIT.OUT
4 5                 5 6
2 1 1 2 1           1 1 1
0 3 0 1 0           3 2 1
3 0 0 3 1           3 3 1
0 2 0 2 0           3 3 2
                    2 5 1
```

Grading

If your answer is correct for subtask A then you obtain 1 point. You obtain 2 additional points if the answer is correct also for subtask B. If your answer is correct for all the three subtasks then you obtain 5 points. Note that it is not necessary in order to score points for subtask A (B) that the output file contains any output for subtask C.

Day 1 - Problem C

Round Trip

Problem

Your team decided to take a round trip in the host country after the competition. You want to travel to a destination city and return to the starting city. The only requirement your team specified is that the forward route to the destination city and the return route back to the starting city must contain the least possible number of common roads. (A route can not contain any road twice or more times.)

You are to write a program that computes two routes between the starting city and the destination city so that the number of common roads in the two routes is as small as possible.

Input Data

The first line of the file **`TRIP.IN`** contains two integers, S and D ($S \neq D$), the labels of the starting and the destination cities, respectively. The second line contains two integers, N and M, where N ($3 \leq N \leq 1000$) is the number of the cities and M

(2≤M≤100000) is the number of the roads between the cities. The cities are labeled from 1 to *N*. Each of the next *M* lines in the file contains two integers, *P* and *Q* ($1 \le P, Q \le N, P \ne Q$), meaning that there is a two-way road between city *P* and city *Q*. There is at most one road between any two cities.

Output Data

The first line of the file **TRIP.OUT** must contain one integer, the least possible number of common roads of the forward and the return routes. The second line should contain a forward route as a sequence of city labels, including the starting and ending city. The third line should contain a return route as a sequence of city labels (again including the starting and ending cities). If there are more possible pairs of routes with the same least number of common roads then your program may output any one of them. If there is no route from the starting city to the destination city then the first and only line must contain -1.

Example input and output:

TRIP.IN	TRIP.OUT
1 6	2
7 8	1 3 2 4 5 7 6
2 1	6 5 4 2 1
1 3	
2 3	
4 2	
4 5	
5 6	
7 5	
6 7	

Grading

If the first line of the output file contains the correct answer then you obtain 2 points. If the first line contains a correct solution and the second and third lines contain correct routes then you obtain 3 additional points.

Day 2 – Problem A

Components of a Bitmap

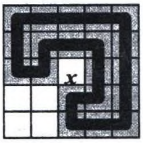
Problem

Black and white pictures are usually stored as bitmaps. A *bitmap* is a rectangular grid of pixels.

A *polyline* between pixels ***P*** and ***Q*** is a sequence of black pixels $P=P_1, P_2, \ldots, P_k=Q$, where P_i and P_{i+1} ($i=1, \ldots, k-1$) are (vertically or horizontally) adjacent pixels. A polyline $P_1, P_2, \ldots, P_k$ is *closed* if $P_1=P_k$, and $P_i \ne P_j$ ($i=1, \ldots, k-1, j=2, \ldots, k$) for $i<j$ unless $i=1$ and $j=k$ (that is the polyline does not contain the same pixel twice).

A *set of black pixels* ***S*** is *connected* if for each pair of pixels (***P***,***Q***) in ***S*** there is at least one polyline ***L*** between ***P*** and ***Q*** with all pixels of ***L*** belonging to ***S***.

A *component* of a bitmap is a maximal connected set of black pixels. A component may enclose *holes*. A *hole* consists of white pixels that are inside a closed polyline. A *compact component* encloses no holes.



Note that in Figure 1 the white pixel in the middle, marked with ***x***, is **not** inside the highlighted closed polyline.

Figure 1

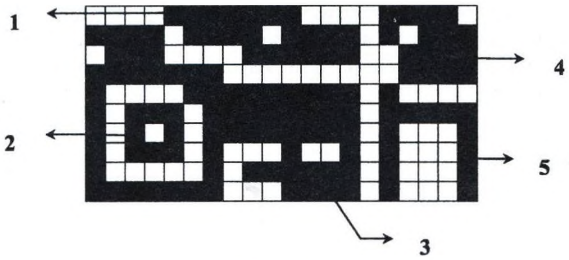


Figure 2

Figure 2 shows a bitmap with five components, of which two are compact ones.

You are to write a program that computes the total number of components and the number of compact components of a given coded bitmap.

Encoding. The bitmaps under investigation are coded (compressed) with the following method. Each row is coded with a sequence of integers $W_1, B_1, ..., W_k, B_k$, where W_i is the number of consecutive white pixels, and B_i is the number of consecutive black pixels, respectively.

For example,

the code of the first line of the bitmap in Figure 2 is the sequence 4 7 4 4 1 0. Components 4 and 5 are compact, while components 1, 2 and 3 are not compact.

Input Data

The first line of the file **`BITMAP.IN`** contains two positive integers, *N* and *M*. *N* ($2 \le N \le 10000$) is the number of rows and *M* ($2 \le M \le 1000$) is the number of columns of the bitmap. The next *N* lines contain the coded bitmap according to the paragraph on *encoding*. Each line terminates with ***-1***.

Output Data

The file **`BITMAP.OUT`** must contain two lines, with a single integer in each line. The first line contains the number of all components and the second line contains the number of compact components of the input bitmap.

Example input and output

BITMAP.IN

```
10 20
4 7 4 4 1 0 -1
0 4 1 4 1 4 1 1 1 3 -1
1 3 4 6 2 4 -1
0 7 9 4 -1
0 1 4 9 1 1 4 0 -1
0 1 1 3 1 8 1 5 -1
0 1 1 1 1 1 1 8 1 1 3 1 -1
0 1 1 3 1 1 3 1 2 1 1 1 3 1 -1
0 1 5 1 1 6 1 1 3 1 -1
0 7 3 4 1 1 3 1 -1
```

BITMAP.OUT

```
5
2
```

Grading

If the first line of your output file contains the correct number of all components, then you obtain 50% of the scores. If the second line of your output file contains the correct number of compact components, then you obtain an additional 50% of the scores.

Day 2 – Problem B

Wildcard Patterns

Problem

Wildcard patterns are frequently used to specify sets of names. For example, we can specify all file names which start with **h** and end with **bak** by the pattern **h*bak**.

A *wildcard pattern* is a string that may contain * characters as wildcards. A string W matches a pattern P if W can be obtained from P by substituting some (possibly empty) string for each * character in P. (Different strings may be substituted for different occurrences of *.) For a pair of patterns P_1 and P_2, Q is a *common pattern* of P_1 and P_2 if any string that matches Q also matches both P_1 and P_2.

A set $Q_1, Q_2, \ldots, Q_L$ of common patterns is called *complete* if any string that matches both P_1 and P_2 matches at least one Q_i ($1 \le i \le L$).

You are to write a program that for a given pair of patterns P_1 and P_2 computes

A. at least one common pattern (but no wrong patterns); or

B. a complete set of common patterns with no more than 6666 entry; or

C. a complete set of common patterns which is minimal in the number of elements (i.e. there is no smaller set of common patterns, which is complete) Note that solving this task you will earn **100% plus an additional 50% of the scores**.

Input Data

The first and the second lines of the file **PATTERN.IN** contain the patterns P_1 and P_2. The patterns are composed of the lowercase letters from 'a' to 'z' and the '*' character. Each pattern contains no more than 20 characters. The number of the * characters in each pattern is between 0 and 6.

Output Data

The first line of the file **PATTERN.OUT** must contain the integer ***K***, the number of common patterns computed as a solution. The next ***K*** lines contain one common pattern in each line; these constitute the solution.

The order of the common patterns is irrelevant. Both case B and C can be solved within the specified limit of 6666 entries. If there is no common pattern of P_1 and P_2, then the first and only line must contain the number 0.

Example input and output

PATTERN.IN	**PATTERN.OUT** for case C
`*ab*`	4
`ba*b`	`ba*ab*b`
	`bab*b`
	`ba*ab`
	`bab`

Grading

If your solution solves case A then you obtain 5 points. If your solution solves case B then you obtain 10 points. If your solution solves case C then you obtain 15 points.

Day 2 – Problem C

Selecting a Majority Member

Problem

Students in your school belong to two disjoint groups. It is known that one group, called the majority group, contains more students than the other group. We want to select a student who belongs to the majority group. Student's identifiers are integers from 1 to the number of students. The only operation that we can perform is to query whether two students belong to the same group.

You are to write a program that finds a student who belongs to the majority group, while it performs as few queries as possible.

LIBRARY

To perform queries, you are given a library `query` with three operations:

- `Size`, to be called once at the beginning, without arguments; it returns *N*, the number of all students. `Size` must be called before the first call to `Member`.
- `Member`, to be called with two student identifiers as arguments; it returns 1 if these two students belong to the same group, otherwise it returns 0.
- `Answer`, to be called once at the end, must be used to submit your solution. `Answer` has an integer argument, the label of a student belonging to the majority group.

The dialogue between your program and the library procedures is logged in the text file **SELECT.OUT**. This log file also contains your answer and an indication whether your answer was correct.

Your answer is accepted only if for any two disjoint groups of all students ***A*** and ***B***, for which all your queries give the same result, your answer identifies a member of the largest group of ***A*** and ***B***. The library forces your program to issue all the queries that are necessary to identify a member of the majority group. That is, there are no pre-defined test cases but the library produces them "on the fly". If you give an answer before you are 100% certain that the answer is correct, you will get 0 points. So don't guess! ☺

Instruction for Pascal programmers: include the import statement

```
uses query;
```

in the source code.

Instructions for C/C++ programmers: use the instruction

```
#include "query.h"
```

in the source code, create a project file `select.gpr` in the task directory, add the files `select.c` (`select.cpp`) and `query.o` into this project, and then *compile* and/or *make* your program.

Experimentation

You can experiment with the library by creating a text file `select.in`. The first and only line must contain the integer ***N***, the number of all students. ***N*** must be an *odd* number!

Example input and output

SELECT.IN

```
7
```

SELECT.OUT

```
Size=7
Member(1,2)=0
Member(3,4)=1
Member(5,6)=1
Member(4,6)=0
Your Answer: 7, Correct
Majority Group:
2 5..7
Non-majority Group:
1 3 4
Number of Queries: 4
Full Possible Score: 3
Your Score: 3
```

For example,

the answer 1 would not be accepted because, while all your queries give the same result for the groups {2, 5, 6, 7} and {1, 3, 4} 1 is not member of the majority group {2, 5, 6, 7}. (*a..b* denotes the set of integers from *a* to *b*.)

Constraints

- For the number of students ***N***, we have 5≤***N***<30000, and ***N*** is odd.
- Your program must not read or write any files.
- For student identifiers ***i***, we have 1≤***i***≤***N***.
- FreePascal library file names: `query.ppw` and `query.ow`.
- Pascal function declarations:

```
function Size: integer;
function Member(x, y: integer): integer;
procedure Answer(x: integer);
```

- C/C++ library file names: `query.h`, and `query.o`
- C/C++ function declarations:

```
int Size(void);
int Member(int x, int y);
void Answer(int x);
```

GRADING

If your answer is correct and the number of queries that your program has performed is ***K*** then you obtain Max(0, ***N-K***) points, where ***N*** is the number of students.

Final results ***were not known when this Chronicle was going to press…***

Contents

CENTRAL-EUROPEAN OLYMPIAD IN INFORMATICS (CEOI) 2

REGULATIONS 4

1ST CENTRAL-EUROPEAN OLYMPIAD IN INFORMATICS
CLUJ – ROMANIA, MAY 27-31., 1994 8

2ND CENTRAL-EUROPEAN OLYMPIAD IN INFORMATICS
SZEGED – HUNGARY, MAY 29 - JUNE 3, 1995 14

3RD CENTRAL-EUROPEAN OLYMPIAD IN INFORMATICS
BRATISLAVA – SLOVAKIA, OCTOBER 9-13., 1996 21

4TH CENTRAL-EUROPEAN OLYMPIAD IN INFORMATICS
NOWY SACZ – POLAND, JULY 17-24., 1997 28

5TH CENTRAL-EUROPEAN OLYMPIAD IN INFORMATICS
ZADAR – CROATIA, MAY 20-27., 1998 37

6TH CENTRAL-EUROPEAN OLYMPIAD IN INFORMATICS
BRNO – CZECH REPUBLIC, SEPTEMBER 2-9., 1999 46

7TH CENTRAL-EUROPEAN OLYMPIAD IN INFORMATICS
CLUJ – ROMANIA, AUGUST 24-31., 2000 55

8TH CENTRAL-EUROPEAN OLYMPIAD IN INFORMATICS
ZALAEGERSZEG – HUNGARY, AUGUST 10-17., 2001 65

CEOI countries