MIUP'13

# Maratona Inter-Universitária de Programação 2013

U LISBOA | UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DE LISBOA

http://miup.di.fc.ul.pt

# Problems

## Scientific Committee

- Alexandre Francisco (Instituto Superior Técnico)
- André Restivo (Universidade do Porto)
- Carlos Grilo (Instituto Politécnico de Leiria)
- Cristina Vieira (Universidade do Algarve)
- Fábio Marques (Universidade de Aveiro)
- Filipe Araújo (Universidade de Coimbra)
- Hugo Torres Vieira (Universidade de Lisboa)
- Margarida Mamede (Universidade Nova de Lisboa)
- Paul Crocker (Universidade da Beira Interior)
- Pedro Guerreiro (Universidade do Algarve)
- Pedro Mariano (Universidade de Lisboa)
- Pedro Ribeiro (Universidade do Porto)
- Rui Mendes (Universidade do Minho)
- Simão Sousa (Universidade da Beira Interior)
- Vasco Pedro (Universidade de Évora)
- Vítor Nogueira (Universidade de Évora)

## Local Organizers

- Fernando Ramos
- Hugo Miranda
- Hugo Torres Vieira
- João Carregosa
- Pedro M. Ferreira
- Pedro Mariano

## Compilation

| Language | Extension | Compiler | Version | Compilation command |
|---|---|---|---|---|
| **C** | .c | gcc | 4.7.2 | gcc -Wall -lm $file |
| **C++** | .cpp | g++ | 4.7.2 | g++ -Wall $file |
| **Java** | .java | OpenJDK | 1.7.0_25 | javac $file |

## Compilation Constraints

- **Maximum compilation time:** 60 seconds
- **Maximum source code size:** 100 KB
- Every source code must be submitted in a single file
- In case of Java submissions, the `.java` file has to have the same name as the class that contains the main method. There is no limit for the number of classes contained in that file.

## Execution

| Language | Execution command |
|---|---|
| **C** | ./a.out |
| **C++** | ./a.out |
| **Java** | java -Xmx256M -Xss256M -classpath . $name |

## Runtime constraints

Unless **explicitly stated otherwise in the problem** the following limits apply to all problems.

- **Maximum CPU time:** 1 second
- **Maximum Memory:** 256MB

## Formatting

- All lines (both in the input and output) should be ended by the newline character (`'\n'`)
- Except when explicitly stated, single spaces are used as a separator.
- No line starts or ends with any kind of whitespace.

## Documentation

Language documentation is available, namely:

- man pages for C/C++ *(using the command line)*
- C++ STL documentation *(bookmarked on your browser)*
- Java SE 7 documentation *(bookmarked on your browser)*

## Problem A: Speed Leaks

This is the time of big data. It is also the time of big leaks. Recently millions of electronic toll records of government vehicles were leaked from the Via Verde[1] computers. Each of those toll records holds the registration plate of the vehicle, the speed recorded when the vehicle passed under the toll bridge, and other data that does not concern us at this moment.

Government vehicles are known drive to very fast, usually for no good reason. With this wealth of data on our hands, we want to disclose to the general public which government vehicles really abuse the speed limits.

### Task

Write a program that reads a file containing toll records and then writes the list of vehicles sorted by decreasing *speed history*. By definition, the speed history of a vehicle is the list of all its recorded speeds, sorted in decreasing order. Given two non-empty speed histories, `s1` and `s2`, we say that `s1` is greater than `s2` if the max value in `s1` is larger than the max value in `s2` or if the max values are equal and the second largest value in `s1` is larger than the second largest values in `s2` or if the max values are equal and the second max values are also equal and the third max value in `s1` is larger than the third max value in `s2`, etc, with the standard understanding that if when comparing speeds two by two as explained one of the speed histories reaches the end and the other not, the latter is the greatest of the two. Note that, if the speed history is sorted decreasingly, the max value is the first, the second max is the second, etc.

### Input

The input file has an undetermined number of lines. Each line contains two items: the first is a string containing only numeric digits, basic uppercase letters and hyphens, representing the registration plate; the second is a positive integer, representing the recorded speed.

### Output

The output file contains a line for each vehicle. Each line contains the registration plate followed by the speed history of the vehicle. Each item on the line is separated from the next by a single space. The lines are sorted decreasingly by speed history (as explained) and then by lexicographical order of the registration plates (i.e., by means of the usual string comparison).

[1]Via Verde (literally "Green Lane") is the electronic toll collection system used in Portugal, covering all motorways and bridges.

## Constraints

- The number of lines in the input file is not zero and is not larger than 20 000.
- The length of the registration plate is not zero and is not larger than 11.
- The recorded speed is a positive integer not larger than 300.
- No other constraints apply.

## Input example

```
12-NL-98 160
81-DU-51 145
88-NP-99 190
33-HZ-60 180
12-NL-98 185
81-DU-51 179
33-HZ-60 235
81-DU-51 160
45-FU-91 190
45-FU-91 130
12-NL-98 230
81-DU-51 178
```

## Output example

```
33-HZ-60 235 180
12-NL-98 230 185 160
45-FU-91 190 130
88-NP-99 190
81-DU-51 179 178 160 145
```

## Problem B: Say Geese

The subway company “Switched Tracks” has a large budget to dig the entire city of Geese (the largest city in Svizzerland). They want a brand new subway system in five years! In a previous market study the company decided where to place the subway stations. The company also knows which pairs of stations may have tunnels and which ones may not. However, they cannot decide on what to do next.

### Task

For now, they only need a small help from you. How many different ways do they have to connect stations, such that the subway system is connected? I.e., passengers should be able to move between arbitrary pairs of stations without leaving the underground. Note that the system may have redundant tunnels, or even cycles among stations.

### Input

The first line of the input is the number of stations, say $n$. You can be sure that $n \geq 2$. Each station has a number between 0 and $n-1$. The following line gives the number of different tunnels that may exist, say $t$. Each of the following $t$ lines has a pair of different numbers separated by white spaces. The numbers represent subway stations that might be connected by a tunnel. For example, the line

```
0 8
```

means that the company may create a tunnel between subway stations 0 and 8. “0 8” means the same as “8 0”. No two stations are ever isolated from each other. At least one path exists among any two.

### Output

The output is a single line with the number of different ways of connecting the stations. Note that this number may not fit into 32 bits.

### Constraints

There are at most 11 stations and 45 possible tunnels.

### Runtime constraints

Maximum CPU time: 3 seconds

## Input example 1

```
3
3
0 1
1 2
2 0
```

## Output example 1

```
4
```

## Input example 2

```
4
6
0 1
1 2
2 3
3 0
0 2
1 3
```

## Output example 2

```
38
```

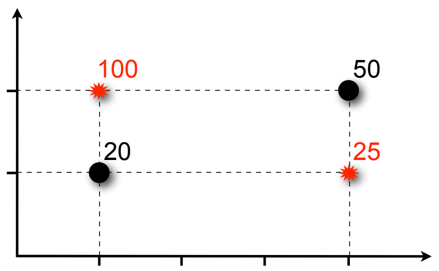
## Problem C: Roman Warfare

Romans planned their invasions carefully and this was probably one of the reasons for their military success. They were eager to conquer wealthy enemy populations, and since Roman armies were heavily armed they were always victorious in battle (except for battles near a small Gaul village?), but their equipment made them hard to move and costly to maintain. When planning an attack, Roman generals had to consider the wealth of the enemy targets and the distances armies had to travel as well as their maintenance costs, which was not easy once the number of armies and potential enemy populations grew. The problem was to decide which armies would attack which cities, considering that one army could at most attack one city and one city could at most be attacked by one army, with the aim of conquering as much wealth as possible, minimizing the distance covered by all armies, and choosing the armies with smaller maintenance costs.

One military directive that prevailed at the time was that any two armies with increasing maintenance costs could only attack populations of increasing wealth, as otherwise the maintenance costs might prove to be hard to justify to the emperor. For example, if an army with maintenance value 5 attacks a population of wealth 20 then an army with maintenance value greater than 5 can only attack populations wealthier than 20, and cannot attack populations poorer than 20. Luckily, no two populations had the same wealth value and no two armies had the same maintenance value. Another directive that was consensual was that often the best military choice would be simply not to use an army in an attack, which implied for that army in particular no traveling at all, no maintenance costs and, naturally, no conquered wealth.

Roman armies travelled only in the four cardinal directions (north, east, south and west) so distances were measured in a manhattan like manner. Generals were also smart enough to plan traveling so as to avoid any obstacles along the way, nevertheless guaranteeing that minimal distances were covered. For example, an army placed at location (1,1) can travel 0.2 units eastbound, then 1 unit northbound and finally 0.8 units eastbound so as to arrive at location (2,2), in such way avoiding locations (1,2) and (2,1).

Generals were aware that the decision to attack a population with the army nearest to it could possibly not be the best one. Consider for instance an army with maintenance cost 100 located at (1,2) and a population of wealth 20 located at (1,1), for which, even if nearby, an attack would imply that no other army of lesser maintenance value could attack populations of greater wealth. In such case, some other population of wealth 50 located at (4,2) would not be attacked if there was only another army left with maintenance cost 25 regardless of its position, i.e., even if it was located at (4,1) as shown in the figure.



For the configuration shown in the figure the optimal planning should indicate 70 $(= 20 + 50)$ as the maximum conquered wealth, with a minimum distance covered of 6 $(= 3 + 3)$ and overall maintenance cost of 125 $(= 100 + 25)$ of the used armies.

### Task

Write a program that given a set of two-dimensional coordinates for the armies and enemy populations together with their respective maintenance costs and wealth determines after the invasion the total wealth conquered, the total distance covered by the armies and the total maintenance costs incurred. These values should be calculated in such a way that in the first place maximizes the conquered wealth, in the second place minimizes the total distance travelled and lastly minimizes the total maintenance value.

## Input

The first line of the input contains two integer values $A$ and $P$ separated by a single white space, that specify the number of armies and populations, respectively. Each of the following $A$ lines describes an army and contains three integers, separated by a single white space, the first two specifying the $x$, $y$ coordinates and the third the maintenance cost. Any two armies have distinct locations and maintenance values. Each of the following $P$ lines describes a population and contains three integers, separated by a single white space, the first two specifying the $x$, $y$ coordinates and the third the wealth. Any two populations have distinct locations and distinct wealth values. No army and population have the same location.

## Output

The output consists of a single line containing three integers, separated by a single white space, corresponding to the wealth conquered, the distance covered and the maintenance value of the used armies, respectively.

## Constraints

| | |
|---|---|
| $1 \leq A \leq 4\,000$ | Number of armies |
| $1 \leq P \leq 4\,000$ | Number of populations |
| $1 \leq x \leq 10\,000$ | $x$ coordinate |
| $1 \leq y \leq 10\,000$ | $y$ coordinate |
| $1 \leq v \leq 10\,000$ | Maintenance/wealth value |

## Input example 1

```
2 2
1 2 100
4 1 25
1 1 20
4 2 50
```

## Output example 1

```
70 6 125
```

## Input example 2

```
3 2
1 2 100
1 1 10
2 4 25
2 2 20
1 4 50
```

## Output example 2

```
70 3 35
```

## Input example 3

```
2 1
1 2 15
1 1 10
2 2 10
```

## Output example 3

```
10 1 15
```

## Problem D: Base Arithmetic

Arithmetic is tricky as it depends on the base. For instance the statement:

$$11 + 101 = 100 \times 10$$

is false in base 10 but true in base 2.

### Task

Your task is to determine the first base where a given statement of the form above is true.

### Input

The input containts four strings sepparated by spaces. These strings represent four numbers, $n_1, \ldots, n_4$ in a given base and represent the statement

$$n_1 + n_2 = n_3 \times n_4$$

The input only contains digits and uppercase letters. The numbers will be represented using the standard used in hexadecimal notation, uppercase letters will substitute numbers with $A = 10, B = 11, \ldots, Z = 36$.

### Output

The output containts a single number that is the first base $B$ where the equality is true. All tests given have a solution.

### Constraints

$n_1 + n_2 = n_3 \times n_4$ must fit in 32 bits.

$2 \leq B \leq 36$

### Input example 1

```
11 101 100 10
```

### Output example 1

```
2
```

### Input example 2

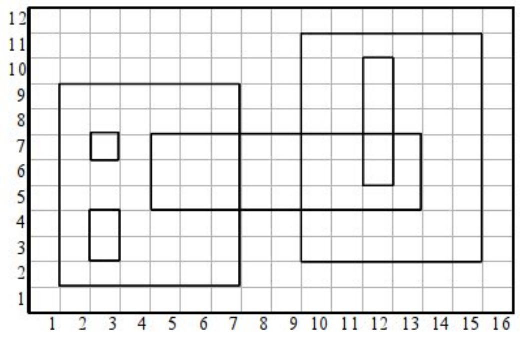
```
101011 110010 7773P 9
```

### Output example 2

```
32
```

## Problem E: Boundaries

Kyle is a child that just loves geometry and games. His current favorite game involves the usage of a graph paper, that is, a writing paper that is printed with fine lines making up a regular grid. The game starts by defining a drawing canvas with width **W** and height **H**. Inside the canvas, a bunch of arbitrary rectangles are drawn, eventually intersecting each other. The following picture illustrates a possible starting position for the game, with 6 rectangles drawn inside a $16 \times 12$ canvas.

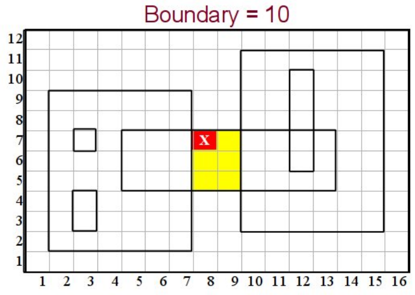


The rectangles define a series of different geometrical shapes completely enclosed by lines. Each geometrical shape is defined by the set of adjacent cells (vertically and horizontally) that have no rectangle lines between them.

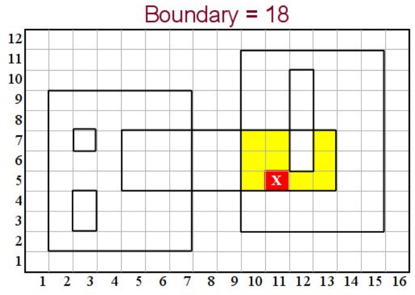
The game proceeds with a series of queries. In each query, a grid cell inside the canvas is given. This cell belongs to only one geometrical shape and the goal is to find out the total length of the boundary of that shape, that is, the length of the rectangle lines that totally enclose the shape and that define it. Note that this includes not only the outside perimeter, but also the boundaries inside the shape, in the cases where other geometrical shapes are contained inside it.

The following figure illustrates 6 different queries. Each query cell is indicated by an 'X' character and the shape that contains it is given in yellow. The respective boundary length is indicated for each case.

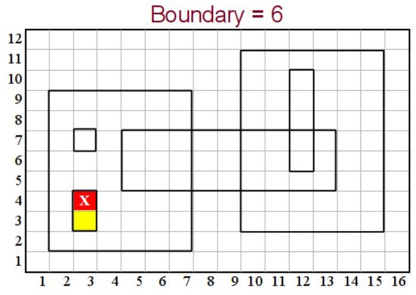
Boundary = 10



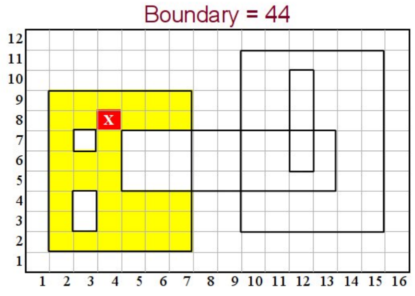
Boundary = 18



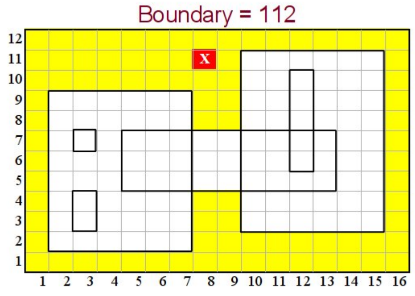
Boundary = 6



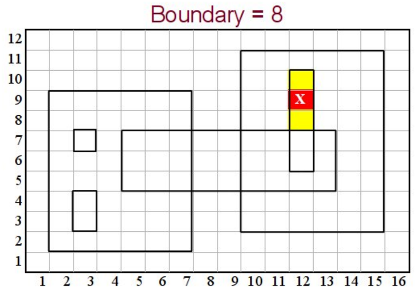
Boundary = 44



Boundary = 112



Boundary = 8



Can you help Kyle play this game?

## Task

Given the canvas size, a series of rectangles and a number of query cells, your task is, for each query, to compute the length of the boundary that defines the geometrical shape containing the given cell, as defined above.

## Input

The first line of input contains two space-separated integers **W** and **H**, the width and weight of the drawing canvas.

The second line contains an integer **R**, the number of rectangles. The following **R** lines describe the rectangles (one per line) and each contains four space-separated integers $\mathbf{X}_1$ $\mathbf{Y}_1$ $\mathbf{X}_2$ $\mathbf{Y}_2$ where $(\mathbf{X}_1, \mathbf{Y}_1)$ is the bottom-left cell inside the rectangle and $(\mathbf{X}_2, \mathbf{Y}_2)$ is the top-right cell also inside the rectangle. The rectangles can come in any order and they may overlap in any way.

After comes a line containing a single integer **Q**, the number of queries to be answered. Then come **Q** lines, each one with two space-separated integers $\mathbf{QX}_i$ $\mathbf{QY}_i$ indicating the cell that should be considered for the query. The queries can come in any order and there may be repeated queries.

Note also that sample input 1 corresponds to the figures given above.

## Output

The output should consist of exactly **Q** lines, each one with a single integer indicating the length of the boundary of the shape that contains the respective query cell (in the same order as the queries appear in the input).

Note that the given definition of boundary includes any outer and inner line that helps define the shape. Also note that if two different rectangles have overlapping sides that help define the shape, you should count those sides only once towards the calculation of the boundary (see sample input 2 for an example).

## Constraints

| | |
|---|---|
| $1 \leq \mathbf{W}, \mathbf{H} \leq 1\,000\,000$ | width and height of the drawing canvas |
| $0 \leq \mathbf{R} \leq 500$ | number of rectangles |
| $1 \leq \mathbf{X_1} \leq \mathbf{X_2} \leq W$ | horizontal coordinates of rectangles |
| $1 \leq \mathbf{Y_1} \leq \mathbf{Y_2} \leq H$ | vertical coordinates of rectangles |
| $1 \leq \mathbf{Q} \leq 500$ | number of queries |
| $1 \leq \mathbf{QX_i} \leq W$ | horizontal coordinates of queries |
| $1 \leq \mathbf{QY_i} \leq H$ | vertical coordinates of queries |

## Input example 1

```
16 12
6
10 3 15 11
2 2 7 9
3 7 3 7
3 3 3 4
5 5 13 7
12 6 12 10
6
8 7
11 5
3 4
4 8
8 11
12 9
```

## Output example 1

```
10
18
6
44
112
8
```

## Input example 2

```
13 9
4
12 7 12 7
2 3 5 8
6 3 10 8
12 7 12 7
3
2 3
12 7
8 6
```

## Output example 2

```
20
4
22
```

## Problem F: Twenty Thirteen

2013 is the first year since 1987 having all digits different from one another. 2013 is also composed of four sequential digits (0, 1, 2 and 3). The last such year was nearly 600 years ago, back in 1432. But the next such year is only 18 years away (2031).

### Task

Given a set of years, for each one of them, calculate the next year composed solely of different and sequential digits.

### Input

The first line of the input will contain a single integer $n$ representing the number of different years. The next $n$ lines will each contain a year ($y_1$ to $y_n$).

### Output

The output should contain $n$ lines. Each one of them should contain a single integer representing the next year composed only by different and sequential digits. Years should not have any leading zeros.

### Constraints

$1 \leq n \leq 10\,000$
$1 \leq y_i < 9\,876\,543\,210$

### Input example

```
4
1430
1987
2014
11111
```

### Output example

```
1432
2013
2031
12034
```

## Problem G: Polygon Phobia

Bob is a young artist, whose current works are paintings with coloured line segments (like the one depicted in the figure below). But Bob is phobic about polygons, he simply cannot draw any closed chain of line segments.

In his next painting workshop Bob has decided to try a new approach, "the luck of the draw". He will start by carefully choosing the possible line segments and by writing the two different endpoints of each segment on a small piece of paper, just as if the painting canvas was a Cartesian coordinate system.

Then, all those pieces of paper will be put into a large jar, which will be very well shaken. At this point, the painting process will begin, using his "luck of the draw" approach. Whilst the jar is not empty, Bob will pick a piece of paper out of it, one at a time, and will then paint the corresponding line segment unless it *closes an existing chain* (because of his phobia).

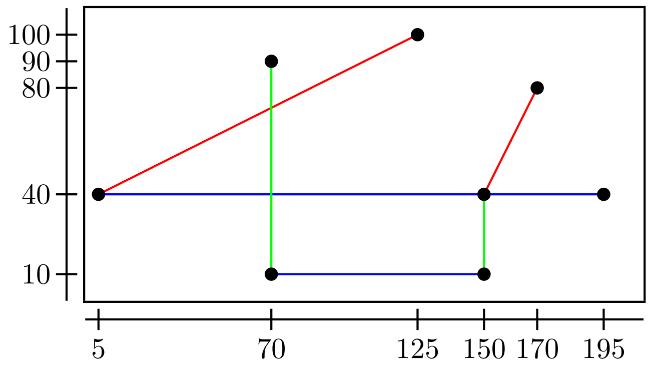
Papers taken out of the jar will be put directly into the recycling bin.

A segment $\overline{p_1\,p_n}$ *closes an existing chain* if the painting already has segments of the form:

$$\overline{p_1\,p_2},\ \overline{p_2\,p_3},\ \overline{p_3\,p_4},\ \ldots,\ \overline{p_{n-1}\,p_n} \quad \text{(for some } n \geq 3\text{)},$$

where $\overline{p\,q}$ denotes the line segment whose endpoints are $p$ and $q$. Notice that $\overline{p\,q} = \overline{q\,p}$. Notice also that chains are defined by segment endpoints and not by line intersections. For instance:

- line segment $\overline{(1,1)\ (5,1)}$ closes the chain $\overline{(1,1)\ (5,5)}$, $\overline{(5,5)\ (5,1)}$, but
- line segment $\overline{(1,1)\ (6,1)}$ does not close the chain $\overline{(1,1)\ (5,5)}$, $\overline{(5,5)\ (5,1)}$.



The figure shows the line segments Bob would paint if the pieces of paper picked from the jar contained the line segments presented in the table, in the specified order.

| Line Segment | | Comment |
|---|---|---|
| **(1)** | $\overline{(5,40)\ (195,40)}$ | Painted (in blue) |
| **(2)** | $\overline{(5,40)\ (125,100)}$ | Painted (in red) |
| **(3)** | $\overline{(70,90)\ (70,10)}$ | Painted (in green) |
| **(4)** | $\overline{(150,10)\ (70,10)}$ | Painted (in blue) |
| **(5)** | $\overline{(150,10)\ (70,90)}$ | Not painted due to **(4)** and **(3)** |
| **(6)** | $\overline{(125,100)\ (195,40)}$ | Not painted due to **(2)** and **(1)** |
| **(7)** | $\overline{(170,80)\ (150,40)}$ | Painted (in red) |
| **(8)** | $\overline{(150,40)\ (150,10)}$ | Painted (in green) |
| **(9)** | $\overline{(70,90)\ (170,80)}$ | Not painted due to **(3)**, **(4)**, **(8)** and **(7)** |
| **(10)** | $\overline{(70,90)\ (150,40)}$ | Not painted due to **(3)**, **(4)** and **(8)** |

## Task

Given a sequence of distinct line segments, each one defined by two different endpoints, the goal is to find out how many line segments Bob would actually paint.

## Input

The first line of the input contains one integer, $S$, which is the number of (distinct) line segments. Each of the following $S$ lines contains four integers, $x_1$, $y_1$, $x_2$, and $y_2$, which indicate that $(x_1, y_1)$ and $(x_2, y_2)$ are the endpoints of a line segment, defined in Cartesian coordinates. Integers in the same line are separated by a single space.

## Output

The output has a single line with the number of line segments that Bob would paint.

## Constraints

| | |
|---|---|
| $1 \leq S \leq 100\,000$ | Number of line segments. |
| $0 \leq x_i < 1\,000$ | Abscissa of a segment endpoint. |
| $0 \leq y_i < 1\,000$ | Ordinate of a segment endpoint. |

## Input example 1

```
10
5 40 195 40
5 40 125 100
70 90 70 10
150 10 70 10
150 10 70 90
125 100 195 40
170 80 150 40
150 40 150 10
70 90 170 80
70 90 150 40
```

## Output example 1

```
6
```

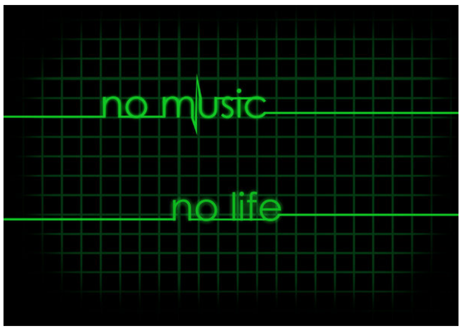
## Input example 2

```
3
5 7 15 7
5 7 20 7
10 7 20 7
```

## Output example 2

```
3
```

## Problem H: No Music, no Life



As Plato's advocated "Music gives a soul to the universe, wings to the mind, flight to the imagination, and life to everything". Drawing inspiration from this, the organizers of the World Music Day intend to produce a galactic ode to the universe. For that, a mega live concert including orchestras of all over the world will be broadcasting across the globe. One of the goals of the celebration is to put together a mega orchestra featuring all known instruments in the world. Such a mega orchestra should be formed by merging some candidate orchestras. More precisely all the musicians from each participating orchestra play their own instrument (and the orchestra features all the instruments, as stated). Furthermore, all players of a given instrument must come from the same orchestra.

Let us see a small example. Considered the instrument list {piano, violin, sitar, tuba, flute} and four orchestras $O_1$={piano, flute}, $O_2$={violin, tuba}, $O_3$={violin, sitar}, $O_4$={piano, tuba, flute}, whose musicians play the correspondent musical instruments. In this case, orchestra $O_3$ supplies the violins and the sitars and orchestra $O_4$ supplies the pianos, the tubas and the flutes.

### Task

Your task is to write a program to find all sets of orchestras that can be used to put together the mega orchestra, as explained.

### Input

The first line of the input will contain two integers representing the number of instruments, $N_I$, and the number of orchestras, $N_O$. Each of the next $N_O$ lines will contain a binary string of length $N_I$ representing the existence of each instrument in each orchestra.

### Output

The output is a single line with the number of sets of orchestras.

### Constraints

$1 \leq N_I, N_O \leq 25$

## Input example 1

```
5 4
10001
01010
01100
10011
```

## Output example 1

```
1
```

## Input example 2

```
5 6
10001
01010
01100
00100
10011
00010
```

## Output example 2

```
3
```

## Problem I: The Magic Potion

The year is `IIIIIOIIIOI` AC. Statix Unix is entirely occupied by the Obliquous. Well, not entirely ... One small village (Manhanttanix) of indomitable Orthogonals still holds out against the invaders.

The reasons for this resistance are twofold. On one hand Manhanttanix was designed according to the Quadrangular Tactic, i.e., the entire village resembles a Quadrangular Grid. On the other hand, they produce a magic potion that gives them superhuman strength. In order to make the magic potion useless for the invaders, they decided to brew a specialized version for each person. But to avoid contamination, each version has to be produced in an empty place of the village.

Since the Orthogonals don't like to be always moving around the village to get their potion, the chief of the tribe decided to build an elaborated aqueduct (or *potionduct*) that connects each source of magic potion to the home of the intended inhabitant. Moreover, it must be built according to the ancestral tradition of the Orthogonals: it has to follow the village grid and there cannot be any intersection.

In order to fulfill this task the tribe leader designated two of the residents. One, Brightix, to conceive the structure of the *potionduct*. The other, Musclix, to build it.

The village musician, Noisix, decided to pay tribute to the two heroes. Since everybody likes him as long as he doesn't speak, let alone sing, he decided to draw a schematic view of a village with a *potionduct*:

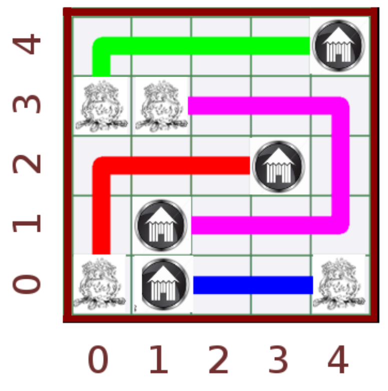


Figura 1: Orthogonals village

### Task

Given the size of the village, the locations of potion sources and of the home of the intended inhabitant, the goal is to find out if a *potionduct* can be built

### Input

The first line of the input contains one integer, $N$, that indicates the size, $N \times N$, of the quadrangular village.

The second line contains another integer, $P$, which indicates the number of inhabitants.

Each of the following $P$ lines contains four integers, $p_{long}$, $p_{lat}$, $h_{long}$ and $h_{lat}$: the first two indicate the position (latitude and longitude) of the magic potion source and the other two indicate the home of the inhabitant for who that magic potion is suitable.

## Output

The output has a single line with: either the word `Toutatis`, if the *potionduct* can be built; or the word `Alesia`, otherwise.

## Constraints

- $2 \leq N \leq 7$
- $1 \leq P \leq N$
- $0 \leq p_{long}, p_{lat}, h_{long}, h_{lat} \leq N-1$

## Input example 1

```
5
4
0 0 3 2
4 0 1 0
1 3 1 1
0 3 4 4
```

## Output example 1

```
Toutatis
```

## Input example 2

```
6
2
0 2 5 5
3 0 2 5
```

## Output example 2

```
Alesia
```