The 2003 ACM ICPC Northwestern Europe Regional Contest
Lund Institute of Technology, Lund, Sweden
November 16, 2003

The Problem Set

A - Bridging signals
B - Vase collection
C - Gladiators
D - Who's the boss?
E - Subway tree systems
F - Prison rearrangement
G - Sightseeing tour
H - A number game

acm International Collegiate Programming Contest

IBM event sponsor

Problem A

Bridging signals

*Source code: signals.**

'Oh no, they've done it again', cries the chief designer at the Waferland chip factory. Once more the routing designers have screwed up completely, making the signals on the chip connecting the ports of two functional blocks cross each other all over the place. At this late stage of the process, it is too expensive to redo the routing. Instead, the engineers have to bridge the signals, using the third dimension, so that no two signals cross. However, bridging is a complicated operation, and thus it is desirable to bridge as few signals as possible. The call for a computer program that finds the maximum number of signals which may be connected on the silicon surface without crossing each other, is imminent. Bearing in mind that there may be thousands of signal ports at the boundary of a functional block, the problem asks quite a lot of the programmer. Are you up to the task?

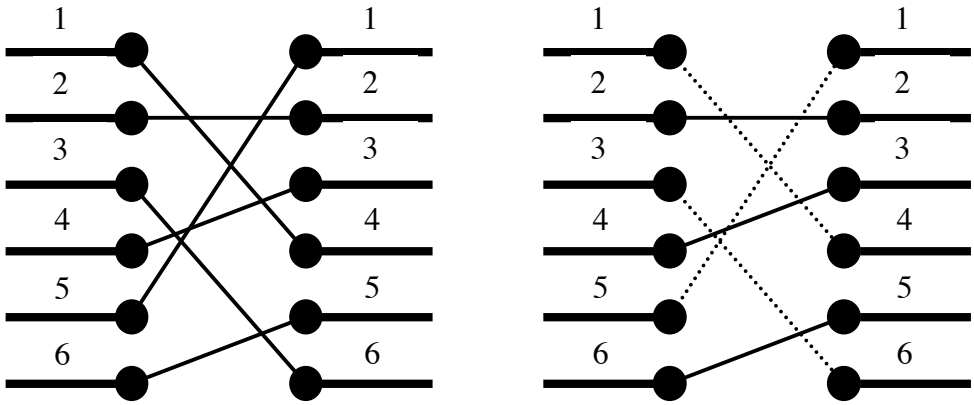


Figure 1. To the left: The two blocks' ports and their signal mapping (4,2,6,3,1,5). To the right: At most three signals may be routed on the silicon surface without crossing each other. The dashed signals must be bridged.

A typical situation is schematically depicted in figure 1. The ports of the two functional blocks are numbered from 1 to p, from top to bottom. The signal mapping is described by a permutation of the numbers 1 to p in the form of a list of p unique numbers in the range 1 to p, in which the i:th number specifies which port on the right side should be connected to the i:th port on the left side. Two signals cross if and only if the straight lines connecting the two ports of each pair do.

Input

On the first line of the input, there is a single positive integer n, telling the number of test scenarios to follow. Each test scenario begins with a line containing a single positive integer $p<40000$, the number of ports on the two functional blocks. Then follow p lines, describing the signal mapping: On the i:th line is the port number of the block on the right side which should be connected to the i:th port of the block on the left side.

Output

For each test scenario, output one line containing the maximum number of signals which may be routed on the silicon surface without crossing each other.

Example input:

```
4
6
4
2
```

```
6
3
1
5
10
2
3
4
5
6
7
8
9
10
1
8
8
7
6
5
4
3
2
1
9
5
8
9
2
3
1
7
4
6
```

Example output:

```
3
9
1
4
```

Problem B

Vase collection

*Source code: vases.**

Mr Cheng is a collector of old Chinese porcelain, more specifically late 15th century Feng dynasty vases. The art of vase-making at this time followed very strict artistic rules. There was a limited number of accepted styles, each defined by its shape and decoration. More specifically, there were 36 vase shapes and 36 different patterns of decoration – in all 1296 different styles.

For a collector, the obvious goal is to own a sample of each of the 1296 styles. Mr Cheng however, like so many other collectors, could never afford a complete collection, and instead concentrates on some shapes and some decorations. As symmetry between shape and decoration was one of the main aestheathical paradigms of the Feng dynasty, Mr Cheng wants to have a full collection of all combinations of *k* shapes and *k* decorations, for as large a *k* as possible. However, he has discovered that determining this *k* for a given collection is not always trivial. This means that his collection might actually be better than he thinks. Can you help him?

Input

On the first line of the input, there is a single positive integer *n*, telling the number of test scenarios to follow. Each test scenario begins with a line containing a single positive integer $m \leq 100$, the number of vases in the collection. Then follow *m* lines, one per vase, each with a pair of numbers, s_i and d_i, separated by a single space, where s_i ($0 < s_i \leq 36$) indicates the shape of Mr Cheng's *i*:th vase, and d_i ($0 < d_i \leq 36$) indicates its decoration.

Output

For each test scenario, output one line containing the maximum *k*, such that there are *k* shapes and *k* decorations for which Mr Cheng's collection contains all *k*k* combined styles.

Example input:

```
2
5
11 13
23 5
17 36
11 5
23 13
2
23 15
15 23
```

Example output:

```
2
1
```

Problem C

Gladiators

*Source code: gladiator.**

In the TV game show Gladiators, the final competition is to run through a steeplechase course. To get some variation, the producer has decided to change the course each week. The course is always built out of *m* obstacles, all of different heights, placed along a straight line. An obstacle consists of two initially connected platforms which may be separated. Between the two platforms of an obstacle, other higher obstacles may be put. Also, obstacles may be put after one another.

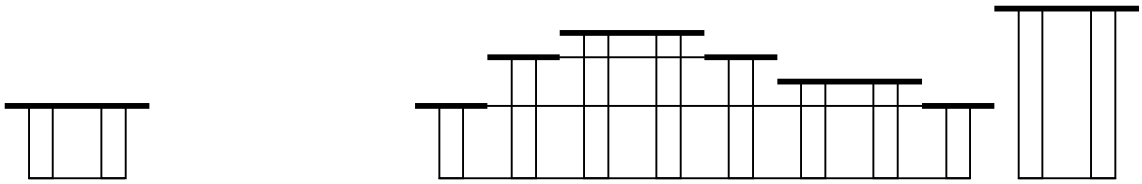


Figure 1. To the left: An obstacle seen from the side. To the right: A steeplechase course consisting of 5 obstacles. The contestants run from left to right.

The producer thinks it is most desirable that the results from different weeks may be compared to each other. Therefore, he wants to build courses of similar degree of difficulty.
A proposed measure of difficulty is the number of platforms that are higher than their immediately preceeding platform along the course. Moreover, the leftmost (first) platform of the course always give rise to one point since it is located above the floor. E.g. the course to the right in figure 1 has four points of difficulty.
Your mission is to find out how many ways there are to build a course of a given point of difficulty, from a given number of obstacles.

Input

On the first line of input is a single positive integer *n*, telling the number of test scenarios to follow. Each test scenario consists of one line containing two non negative integers *m* and *k*, where $m \leq 50$ is the number of obstacles, and *k* is the point of difficulty asked for.

Output

For each test scenario, output one line containing a single positive integer equal to the number of different courses constructable from the *m* obstacles, amounting to a point of difficulty of exactly *k*. You may safely assume that the answer is less than 10^{100}.

Example input:

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

Example output:

```
0
1
1
2
1
8
```

Problem D

Who's the boss?

*Source code: boss.**

Several surveys indicate that the taller you are, the higher you can climb the corporate ladder. At TALL Enterprises Inc. this "de facto standard" has been properly formalized: your boss is always at least as tall as you are. Furthermore, you can safely assume that your boss earns a bit more than you do. In fact, you can be absolutely sure that your immediate boss is the person who earns the least among all the employees that earn more than you and are at least as tall as you are. Furthermore, if you are the immediate boss of someone, that person is your subordinate, and all his subordinates are your subordinates as well. If you are nobody's boss, then you have no subordinates. As simple as these rules are, many people working for TALL are unsure of to whom they should be turning in their weekly progress report and how many subordinates they have. Write a program that will help in determining for any employee who the immediate boss of that employee is and how many subordinates they have. Quality Assurance at TALL have devised a series of tests to ensure that your program is correct. These test are described below.

Input

On the first line of the input is a single positive integer *n*, telling the number of test scenarios to follow. Each scenario begins with a line containing two positive integers *m* and *q*, where *m* (at most 30000) is the number of employees and *q* (at most 200) is the number of queries. The following *m* lines each list an employee by three integers on the same line: employee ID number (six decimal digits, the first one of which is not zero), yearly salary in Euros and finally height in µm (1 µm = 10^{-6} meters – accuracy is important at TALL). The chairperson is the employee that earns more than anyone else and is also the tallest person in the company. Then there are *q* lines listing queries. Each query is a single legal employee ID.

The salary is a positive integer which is at most 10 000 000. No two employees have the same ID, and no two employees have the same salary. The height of an employee is at least 1 000 000 µm and at most 2 500 000 µm.

Output

For each employee ID *x* in a query output a single line with two integers *y k*, separated by one space character, where *y* is the ID of *x*'s boss, and *k* is the number of subordinates of *x*. If the query is the ID of the chairperson, then you should output 0 as the ID of his or her boss (since the chairperson has no immediate boss except, possibly, God).

Example input:

```
2
3 3
123456 14323 1700000
123458 41412 1900000
123457 15221 1800000
123456
123458
123457
4 4
200002 12234 1832001
200003 15002 1745201
200004 18745 1883410
200001 24834 1921313
```

Example output:

```
123457 0
0 2
123458 1
200001 2
200004 0
200004 0
0 3
```

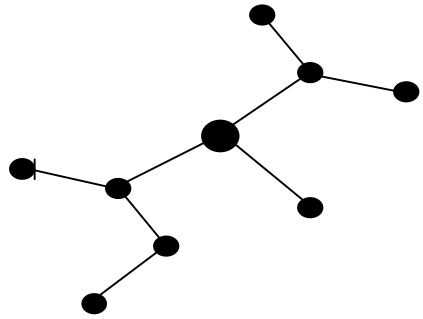
```
200004
200002
200003
200001
```

Problem E

Subway tree systems

*Source code: subway.**

Some major cities have subway systems in the form of a tree, i.e. between any pair of stations, there is one and only one way of going by subway. Moreover, most of these cities have a unique central station. Imagine you are a tourist in one of these cities and you want to explore all of the subway system. You start at the central station and pick a subway line at random and jump aboard the subway car. Every time you arrive at a station, you pick one of the subway lines you have not yet travelled on. If there is none left to explore at your current station, you take the subway line back on which you first came to the station, until you eventually have travelled along all of the lines twice, once for each direction. At that point you are back at the central station. Afterwards, all you remember of the order of your exploration is whether you went further away from the central station or back towards it at any given time, i.e. you could encode your tour as a binary string, where 0 encodes taking a subway line getting you one station further away from the central station, and 1 encodes getting you one station closer to the central station.



```
0010011101001011
0100011011001011
0100101100100111
...
```

Figure 1. To the left: A subway tree system. The larger dot is the central station. To the right: Three out of several possible encodings of exploration tours for the subway system.

Input

On the first line of input is a single positive integer *n*, telling the number of test scenarios to follow. Each test scenario consists of two lines, each containing a string of the characters '0' and '1' of length at most 3000, both describing a correct exploration tour of a subway tree system.

Output

For each test scenario, output one line containing the text “same” if the two strings may encode exploration tours of the same subway tree system, or the text “different” if the two strings cannot be exploration tours of the same subway tree system.

Example input:

```
2
0010011101001011
0100011011001011
0100101100100111
0011000111010101
```

Example output:

```
same
different
```

Problem F

Prison rearrangement

*Source code: prison.**

In order to lower the risk of riots and escape attempts, the boards of two nearby prisons of equal prisoner capacity, have decided to rearrange their prisoners among themselves. They want to exchange half of the prisoners of one prison, for half of the prisoners of the other. However, from the archived information of the prisoners' crime history, they know that some pairs of prisoners are dangerous to keep in the same prison, and that is why they are separated today, i.e. for every such pair of prisoners, one prisoners serves time in the first prison, and the other in the second one. The boards agree on the importance of keeping these pairs split between the prisons, which makes their rearrangement task a bit tricky. In fact, they soon find out that sometimes it is impossible to fulfil their wish of swapping half of the prisoners. Whenever this is the case, they have to settle for exchanging as close to one half of the prisoners as possible.

Input

On the first line of the input is a single positive integer n, telling the number of test scenarios to follow. Each scenario begins with a line containing two non-negative integers m and r, $1<m<200$ being the number of prisoners in each of the two prisons, and r the number of dangerous pairs among the prisoners. Then follow r lines each containing a pair x_i y_i of integers in the range 1 to m, which means that prisoner x_i of the first prison must not be placed in the same prison as prisoner y_i of the second prison.

Output

For each test scenario, output one line containing the largest integer $k \leq m/2$, such that it is possible to exchange k prisoners of the first prison for k prisoners of the second prison without getting two prisoners of any dangerous pair in the same prison.

Example input:

```
3
101 0
3 3
1 2
1 3
1 1
8 12
1 1
1 2
1 3
1 4
2 5
3 5
4 5
5 5
6 6
7 6
8 7
8 8
```

Example output:

```
50
0
3
```

Problem G

Sightseeing tour

*Source code: tour.**

The city executive board in Lund wants to construct a sightseeing tour by bus in Lund, so that tourists can see every corner of the beautiful city. They want to construct the tour so that every street in the city is visited exactly once. The bus should also start and end at the same junction. As in any city, the streets are either one-way or two-way, traffic rules that must be obeyed by the tour bus. Help the executive board and determine if it's possible to construct a sightseeing tour under these constraints.

Input

On the first line of the input is a single positive integer n, telling the number of test scenarios to follow. Each scenario begins with a line containing two positive integers m and s, $1 \leq m \leq 200$, $1 \leq s \leq 1000$ being the number of junctions and streets, respectively. The following s lines contain the streets. Each street is described with three integers, x_i, y_i, and d_i, $1 \leq x_i, y_i \leq m$, $0 \leq d_i \leq 1$, where x_i and y_i are the junctions connected by a street. If $d_i=1$, then the street is a one-way street (going from x_i to y_i), otherwise it's a two-way street. You may assume that there exists a junction from where all other junctions can be reached.

Output

For each scenario, output one line containing the text "possible" or "impossible", whether or not it's possible to construct a sightseeing tour.

Example input:

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

Example output:

```
possible
impossible
impossible
possible
```

Problem H

A number game

*Source code: numbers.**

The wide dissemination of calculators and computers is not without disadvantages. Teachers all over the world find out that even students in technical disciplines tend to have a surprising lack of calculating ability. Accustomed as they are to the use of calculators and computers, many of them are unable to make calculations like 7*8 mentally, or to factor 91 by heart.
We all know, but who cares?
Professor Bartjens cares. Professor Bartjens is a bit old-fashioned. He decided to give his students some training in calculating without electronic equipment - even without a slide rule. He invented a two-person game involving mental calculations.

Professor Bartjens would write a positive number on the blackboard. During the game more positive numbers may appear on the blackboard. The two players will then make moves in turn. A player on move is obliged to make a move, unless the blackboard is empty, in which case the game is over. A move is one of the following:

- If you see the number 1 on the blackboard, you may take it. That means: you gain one point, and the number disappears from the blackboard.
- If you see a prime number p on the blackboard, you may subtract one. That is: you gain one point, and the p on the blackboard is replaced by $p - 1$.
- If you see a composite number c on the blackboard, you may replace it by two smaller (positive) numbers, a and b, such that $a * b = c$. You do not gain any points.

The goal is of course to obtain as many points as you can.

Professor Bartjens was hoping that his students would find the game so interesting that they would spend all day playing, thereby improving their skills in calculation. Indeed his students did find the game interesting, and spent many hours, not so much playing the game as discussing optimal strategies.
The students came to two conclusions. First, the sum of the two players' points after any given game are the same regardless of the actual scheme played. Thus, a player maximising his own points also minimises his opponent's! Second, it is always best to take a point when you have the chance. Thus, whenever prime numbers or ones are written on the blackboard, the player on move takes one of them.

Here is your problem: given a starting number, and assuming both players play to maximise their own points, what will be the outcome?

Input

On the first line of the input is a single positive integer n, telling the number of test scenarios to follow. Each scenario consists of a single line containing the positive integer $m<1000000$, the number initially written on the blackboard.

Output

For each test scenario, output one line containing two numbers separated by one space character, equal to the points gained by the two players, both playing to maximise their own points. The first number is the number of points gained by the first player.

Example input:

```
6
1
2
3
4
5
6
```

Example output:

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