

2015/06/10 2015年6月 PTC 線上程式設計競賽

競賽試題

共 5 題，14 頁

注意事項

一、本比賽系統採用 PC²，所使用的 I/O 是標準輸出輸入裝置，所以可以使用 C 語言的 `scanf()`、`printf()`，或是 C++ 語言上的 `cin`、`cout` 來讀入及輸出資料，比較要注意的是：本系統並不是用人工方式來 keyin 資料，所以不必在意使用者界面的問題，也就是說不用印出像是 "Please enter a number" 或 "The answer is" . . . 之類的文字；此外，有些題目是以讀到 EOF 為 input 結束，有些是讀入 0 結束等等的，必需善用 I/O 函式。上傳檔案的檔名請勿使用中文以免發生不必要的錯誤。

二、比賽用的編譯器版本：

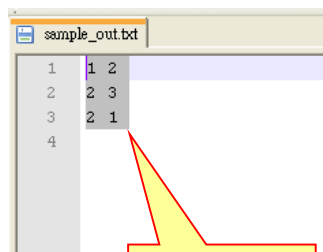
- C/C++ :
 - ◆ gcc : 4.9.2 (Cygwin 1.7.35_1)
 - ◆ g++ : 4.9.2 (Cygwin 1.7.35_1)
- Java :
 - ◆ Version 1.8.0_45
 - ◆ Java(TM) SE Runtime Environment (build 1.8.0_45-b14)
 - ◆ Java HotSpot(TM) 64-Bit Server VM(build 25.45-b02, mixed mode)
- Microsoft C# :
 - ◆ Microsoft (R) Visual C# Compiler version 4.0.30319.34209
- Microsoft C++ :
 - ◆ Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 16.00.30319.01
- Compiler Command line :
 - C : gcc -O2 -std=gnu99 -static \$* -lm
 - C++ : g++ -O2 -std=gnu++0x -static \$*
 - C++11 : g++ -O2 -std=c++11 -static \$*
 - C# : csc
 - Microsoft C++ : cl
 - Java : javac
 - For Java, the compiled code will be executed using the following command:
java -Xmx1024m -Xss8m 。

※"本次比賽提供兩種 C++ 選擇，上傳時請留意語言選項"

若出現 Compilation Error，可能是某些函式不支援。

三、PC² 系統判定錯誤可能原因：

正確答案



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

假設題目
要求結尾有
換行字元

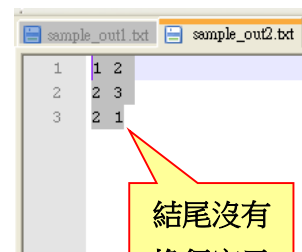
錯誤答案



```
1 1 2
2 2 3
3 2 1
4
5
sample_out2.txt
1 1 2
2 2 3
3 2 1
4
5
6
7
```

多空白

多換行字元



```
sample_out1.txt
1 1 2
2 2 3
3 2 1
sample_out2.txt
1 1 2
2 2 3
3 2 1
```

結尾沒有
換行字元

特別注意題目範例是否有換行字元。

四、PC² 系統判定結果說明：

結果

Yes

No - Compilation Error

No - Run-time Error

No - Time-limit Exceeded

No - Wrong Answer

No - Excessive Output

No - Output Format Error

No - Other - Contact Staff

說明

解題正確

錯誤：編譯錯誤

錯誤：程序運行錯誤

錯誤：運行超時（每道題都有運行時間限制）

錯誤：運行結果與標準答案不一致

錯誤：程序運行佔用內存空間超出要求

錯誤：輸出格式錯誤

未知錯誤

Problem 1. Magic Box

(Time Limit: 10 seconds)

Problem Description

Karen likes to explore ancient wonders. One day, he found a magic box with a magic number k on it, and n magic balls beside the magic box. Each ball also has a number written on it, where the i -th ball has the number a_i .

After a long time, Karen finally deciphered the text on that magic box: “Choose some (at least one) balls, such that the product of the numbers on the chosen balls equal to k , and you will get my treasure. Oh, this magic box doesn't support **BigInteger** yet, so I will calculate that product modulo 1000000007.”

We all know that Karen is very smart. But unfortunately, he is allergic to the number 1000000007. He cannot stop snoozing when he tries to do any arithmetic relevant to 1000000007. Can you help him to calculate how many ways to choose balls such that he can get the ancient treasure?

Technical Specification

- $1 \leq n \leq 30$
- $0 \leq k \leq 10^9$
- $0 \leq a_i \leq 10^9$
- The number of test case $T \leq 100$

Input Format

The first line contains an integer T indicating the number of the test cases. For each test case, there are two integers n, k in the first line. Then there are n integers a_1, a_2, \dots, a_n in the following line.

Output Format

For each test case, output the number of ways to choose balls such that he can get the ancient treasure.

Example

Sample Input:	Sample Output:
3 3 4 2 2 2 8 24 1 2 3 4 5 6 7 8 2 3 10 100000001	3 6 1

Problem 2. Hidden Lines

(Time Limit: 3 seconds)

Problem Description

3D computer graphics are graphics that use a three-dimensional representation of geometric data that is stored in the computer for the purposes of performing calculations. An important topic in 3D computer graphics is to determine which parts of surfaces are not visible from a certain viewpoint. The analogue for lines in 2D is to determine which parts of lines are hidden.

Consider that there are n line segments on the plane and the viewpoint is from the top ($y = +\infty$). The segments are regarded as opaque obstacles. A line segment is hidden if it is invisible from $y = +\infty$. For example, consider the three segments a_1, a_2, a_3 in Figure 1. For simplicity, let $[p_i, p_j]$ denote the segment defined by the two endpoints p_i and p_j . Then, $a_1 = [(1, 2), (4, 8)]$, $a_2 = [(1, 6), (7, 2)]$, $a_3 = [(8, 5), (10, 2)]$. In this example, the line segment $[(1, 2), (2.5, 5)]$ is hidden (since there is a line segment $[(1, 6), (2.5, 5)]$ above it). In contrast, the line segment $[(4, 4), (7, 2)]$ is not a hidden line segment.

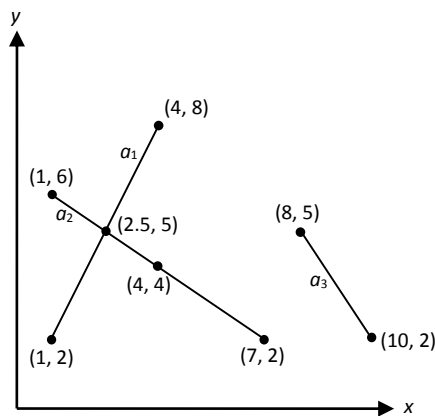


Figure 1.

We are interested in the total length of the hidden parts of n given line segments. For example, in Figure 1, we are interested in the total length of the two segments $[(1, 2), (2.5, 5)]$ and $[(2.5, 5), (4, 4)]$. Please write a program to determine the length.

Technical Specification

- The number n of line segments is at most 10^5 .
- The x, y coordinates of the endpoints of each line segment are integers between 0 and 10^5 .
- There is no vertical line segment.
- It is guaranteed that no two line segments induce a single line segment. We say that two line segments induce a single line segment if the union of their points represents a line segment. For example, $[(0, 0), (2, 2)]$ and $[(2, 2), (3, 3)]$ induce a line segment $[(0, 0), (3, 3)]$.

Input Format

The first line contains an integer $t \leq 10$ indicating the number of test cases. The first line of each test case contains an integer n , which is the number of line segments. The following n lines describe line segments a_1, a_2, \dots, a_n . The i -th line contains four integers x_i, y_i, x'_i, y'_i ($x_i < x'_i$), where $a_i = [(x_i, y_i), (x'_i, y'_i)]$.

Output Format

For each test case, output the total length of the hidden parts of the given line segments. The length should be rounded to three decimal places.

Example

Sample Input:	Sample Output:
2 3 1 2 4 8 1 6 7 2 8 5 10 2 3 0 1 100000 0 0 0 2 100000 1 100000 3 0	5.157 50005.000

Problem 3. Bombs

(Time Limit: 3 seconds)

Problem Description

Emergency! Someone has placed some bombs on the moving train! As a professional team leader of bomb disposal, you are assigned to defuse the bombs. Fortunately, the criminal has been arrested, and you get the blueprint of these bombs. However, you suddenly find out that these bombs are interconnected by many lines which are difficult to figure out the way to stop them. What's worse, there's a timer! If you cannot stop the bombs in time, they will explode and cause lots of injuries. As a result, your job is to decide which lines to be cut and the rest of them remain intact.

All bombs are made up of cores. There are totally two kinds of bombs, with single-core and dual-core. For each core, there are exactly two sensors on it. That is to say, the single-core bombs have one core and two sensors, while the dual-core bombs have two cores and four sensors, as shown in Figure 1. Note that the squares and the circles are the sensors.

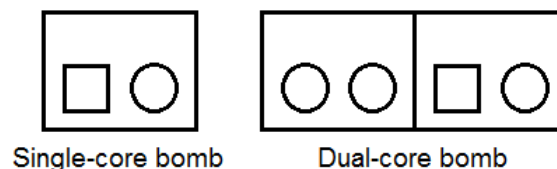


Figure 1. Two kinds of bombs

According to the blueprint, for each sensor, there will be **exactly one line** passing through it, though one line may pass through many sensors, and one line may also pass through only one sensor. In Figure 2, line 1 passes through bomb 1; line 2 passes through bomb 1, 2 and 3; line 3 passes through bomb 2 and 3. Note that the numbers beside the sensors denote the line passing through it.

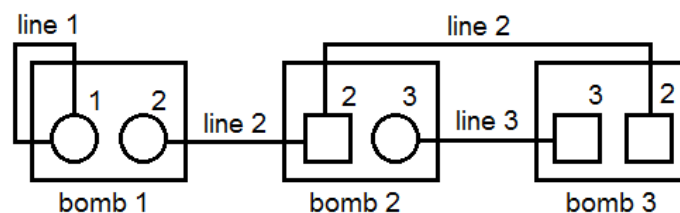


Figure 2. Lines passing through single-core bombs

You must stop **at least one core** for each bomb in order to stop the bomb. However, in order to stop a core, you need to make both of two sensors on it malfunctioned. **Cutting or not cutting one line will effect on all sensors passed by this line.** For the circle sensor, you must cut off the line passing through it; conversely, for the square sensor, you must ensure the line passing through it remains intact. In Figure 3, cutting off line 1, 4 and 6 will make all bombs stop. Note that the sensor with stripes means that it is malfunctioned.

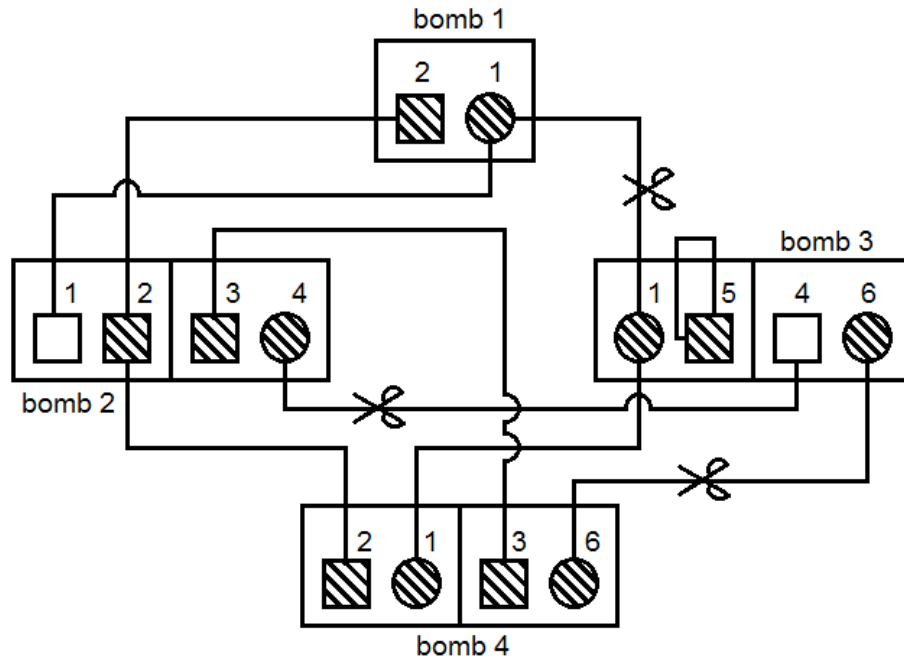


Figure 3. Bomb map with 1 single-core bomb and 3 dual-core bombs.

As shown in Figure 3, because all bombs fulfill the goal that at least one core is stopped, none of them will explode anymore. You immediately observe that there may be more than one solution which can stop the bombs. In above example, even if line 6 remains intact, all bombs will stop as well. You also find out that you can stop all bombs without cutting off any lines in some situations, such as all sensors are square. Similarly, there may exist some situations that you cannot stop all bombs no matter what lines you cut. Your job is to determine whether or not there exists a solution which can stop all bombs, given the description of blueprint.

Technical Specification

- The number of lines **n** will be in the range: $2 \leq n \leq 1000$.
- The identifier of lines **id** will be in the range: $1 \leq id \leq n$.
- The number of single-core bombs **s** will be in the range: $0 \leq s \leq n$.
- The number of dual-core bombs **d** will be in the range: $0 \leq d \leq n$.

- There will be at least one bomb in each test case: $1 \leq s+d$.

Input Format

The first line is an integer **t** which indicates the number of test cases. Then there will be exactly **t** sets of test cases following

Each test case will begin with three integers, **n**, **s**, and **d**, respectively. The next **s** lines denote the description of single-core bombs. Each of these **s** lines contain one 4-tuple, with four integers separated by spaces, denoting the core description. The first two integers are the line **ids** passing through the sensors; the last two integers represent the type of sensors, where 0 means square and 1 means circle sensor.

The next **d** lines denote the description of dual-core bombs. Each of these **d** lines contain two 4-tuples (eight integers) a line, denoting two core descriptions respectively. The format of 4-tuples are the same as described above.

Output Format

For each test case, output “Yes” in one line if there exists a solution which can stop all the bombs. Otherwise, output “No” in one line.

Example

Sample Input:	Sample Output:
2	No
3 3 0	Yes
1 2 1 1	
2 3 0 1	
3 2 0 0	
6 1 3	
2 1 0 1	
1 2 0 0 3 4 0 1	
1 5 1 0 4 6 0 1	
2 1 0 1 3 6 0 1	

Hint: These are the test cases shown in Figure 2 and 3, respectively.

Problem 4. Quick Reversal

(Time Limit: 3 seconds)

Problem Description

DNA sequences are genetic materials for most of the lives. There are four types of nucleotides, namely the adenine, guanine, cytosine and thymine, in a DNA sequence. The most common evolutionary events are reversals, in which a contiguous interval of a DNA sequence is put into the reverse order. For simplicity, we ignore the actual content of a DNA sequence but use its positions to specify a reversal event. Accordingly, a DNA sequence of length n can be consider as an integer sequence $1, 2, \dots, n$. Suppose we have the sequence $1, 2, 3, 4, 5, 6, 7$. After reversing the fragment $[3, 5]$, which denotes the fragment starting from position 3 to position 5, we get $1, 2, 5, 4, 3, 6, 7$. Subsequently, after one more reversal event $[1, 3]$, we would have $5, 2, 1, 4, 3, 6, 7$. You can see that the idea of a reversal event is quite simple. However, since a DNA sequence is usually very long, you need to devise a fast way to finish the required task.

Technical Specification

- The length of a DNA sequence can be as long as 10^7 .
- Initially, position i of the sequence is labeled by the number i .
- The reversal event can be applied to any fragment of the target sequence.
- The number of test cases is at most 10. The number of reversal events in a test case is at most 2000.

Input Format

The first line is an integer which indicates the number of test cases. Each test case starts with two numbers n and m separated by a space in one line. The first number n specifies the length of the sequence. The second number m is the number of reversal events that will be subsequently applied to this sequence. The next m lines are these events, in which two integers a and b separated by a space in the same line define a fragment from position a to position b that has to be reversed. You can assume that a is always less than or equal to b .

Output Format

For each test case, output the sum of labels in odd positions of the final sequence in one line.

Example

Sample Input:	Sample Output:
2 5 2 1 4 2 3 2910960 2 6 201506 8 888888	12 2118422030400

Problem 5. Milk Delivery

(Time Limit: 5 seconds)

Problem Description

Melky is a milkman, and he is very proud of his job. He always delivers as many bottles of milk as he can. In Melky's country, there are n cities numbered from 1 to n and m routes connecting cities. The i^{th} route R_i is from city s_i to city t_i . Along route R_i , Melky can deliver w_i bottles of milk.

Melky is planning a k -day domestic trip for his vacation. The start s and the end t of this trip can be arbitrary cities in his country. On each day of the trip, Melky will travel along exactly one route. The route should start at the end of the route travelled on the previous day. Moreover, the first travelled route should start at s , and the last travelled route should end at t . Since Melky loves his job, he will deliver milk along the route even he is on vacation.

Melky have listed q candidate pairs of start and end of his trip. For each pair of start s and end t , he wonders what the maximum number of bottles of milk such that he can deliver during his k -day trip from s to t is. Please write a program to help Melky.

Technical Specification

- There are at most 20 test cases.
- The routes are directed, and there might be more than one route from one city to another.
- During Melky's trip, he can deliver milk along the same route on different days.
- $0 < n \leq 200, 0 < m \leq 10000, 0 < k \leq 1000, 0 < q \leq 1000$.
- $0 < w_i \leq 10000$ for every $0 < i \leq m$.

Input Format

The first line is an integer which indicates the number of test cases. Each test case consists of three parts. The first part describes the basic settings. It has 4 integers n, m, k, q separated by blanks on one line. n is the number of cities. m is the number of routes. k is the number of days of Melky's trip. q is the number of

candidate pairs of start and end of Melky's trip. The second part of each test case describes the routes. This part consists of m lines, and each of these lines contains 3 integers u, v, w separated by blanks. The corresponding route is from city u to city v , and Melky can deliver w bottles of milk along it. The last part of each test case consists of q lines, and it describes the q candidate pairs of start and end of Melky's trip. Each line contains two integers s and t where s and t are the start and the end of the corresponding candidate pair, respectively.

Output Format

For each test case, output q lines. Each of these lines contains the maximum number of bottles of milk such that Melky can deliver during the corresponding k -day trip. If it is impossible to travel from the start to the end in exactly k days, output 0.

Example

Sample Input:	Sample Output:
2	0
3 3 2 3	6
1 2 3	7
2 3 4	2242
3 1 2	3241
1 2	
2 1	
1 3	
3 4 10 2	
1 2 1000	
2 3 100	
3 1 10	
1 3 1	
2 2	
3 2	