Problem A. Kick Start

Input file:	`standard input`
Output file:	`standard output`

Kick Start is a global online coding competition brought by Google. Participants can compete in online rounds held throughout the entire year, and will have the opportunity to develop and grow their programming abilities while getting a glimpse into the technical skills needed for a career at Google.

Mr. Panda is a Kick Start enthusiast who doesn't miss any Kick Start round of any year. He is now reading the 2019 schedule of rounds and wonders what's the date of the next Kick Start round (**excluding today**). Can you help Mr. Panda determine that date?

Any given date in this problem will be in the format of a month abbreviation followed by a date ordinal number. Recall that month abbreviations are 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sept', 'Oct', 'Nov', and 'Dec'; and ordinal numbers are '1st', '2nd', '3rd', '4th', '5th', etc.

Input

The first line of the input gives the number of test cases, T $(1 \le T \le 100)$. T test cases follow.

For each test case, the first line contains an integer n $(1 \le n \le 20)$, the number of scheduled Kick Start rounds for this year.

In the next n lines, each line contains a date in the year 2019, in the format of a month abbreviation followed by a date ordinal number. Note that scheduled dates may **not** be arranged in chronological order, and all scheduled dates are distinct.

The last line contains the date of today, also in the format of a month abbreviation followed by a date ordinal number.

Output

For each test case, output one line containing "`Case #x: y`", where `x` is the test case number (starting from 1), and `y` is the date of the next Kick Start round or "See you next year" (quotes for clarity) if there is no following Kick Start round of this year.

Example

standard input	standard output

```
2
3
Jan 1st
Feb 2nd
Mar 3rd
Jan 2nd
8
Mar 24th
Apr 20th
May 26th
Jul 28th
Aug 25th
Sept 29th
Oct 19th
Nov 17th
Nov 17th
```
	```
	Case #1: Feb 2nd
	Case #2: See you next year
	```

Problem B. Infimum of Paths

Input file:	`standard input`
Output file:	`standard output`

On a directed graph, we use $lex(p)$ to denote the lexical weight of a path p, where the path p can be regarded as a sequence of consecutive edges. The lexical weight is defined by the recurrence relation

$$lex([]) = 0, lex([e_1, e_2, \ldots, e_n]) = \frac{w(e_1) + lex([e_2, e_3, \ldots, e_n])}{10},$$

where $w(e_1)$ is the weight of edge e_1, which is an integer between 0 and 9, inclusive.

Given a directed graph, find the infimum of the lexical weights of all paths from node 0 to node 1. The infimum of a set of rational numbers is the greatest rational number that, if exists, is less than or equal to all elements in this set.

Input

The first line of the input gives the number of test cases, T ($1 \le T \le 100$). T test cases follow.

For each case, the first line contains two integers, n ($2 \le n \le 2000$, $\sum n \le 20000$) and m ($1 \le m \le 4000$, $\sum m \le 40000$), where n is the number of nodes and m is the number of edges.

Then m lines follow, each of which contains three integers u, v, w ($0 \le u, v < n$, $0 \le w \le 9$), indicating an edge from u to v of weight w.

It is guaranteed that there exists at least one path from node 0 to node 1 for each test case.

Output

For each test case, output one line containing "`Case #x: y`", where `x` is the test case number (starting from 1), and `y` is the answer modulo $(10^9 + 7)$. More specifically, if the answer can be formed as an irreducible fraction $\frac{A}{B}$, then `y` will be $(A \cdot B^{-1}) \bmod (10^9 + 7)$.

Example

<table>
<tr><th>standard input</th><th>standard output</th></tr>
<tr><td><pre>2
5 5
0 2 3
2 3 4
2 4 1
3 1 2
4 1 3
5 6
0 1 9
2 0 6
3 0 1
0 3 3
4 0 3
4 2 7</pre></td><td><pre>Case #1: 241000002
Case #2: 40404041</pre></td></tr>
</table>

Note

For the first sample, the path corresponding to the infimum is $0 \to 2 \to 4 \to 1$, so the answer is 0.313.

Problem C. Mr. Panda and Typewriter

Input file:	`standard input`
Output file:	`standard output`

Mr. Panda recently got a brand-new typewriter as a birthday gift from Mr. Champion. Mr. Panda likes the typewriter so much. He wants to use it to type a thank you letter S and mail it to Mr. Champion.

To type the thank you letter, Mr. Panda starts with an empty string on a white-paper, and the following operations are allowed to perform by using the typewriter:

- Spend X units of time to add any single character to the end of Mr. Panda's string.
- Spend Y units of time to copy any substring of Mr. Panda's string (that is, all of the sequential characters between some start point and some end point in Mr. Panda's string) to the clipboard. Doing this overwrites whatever was in the clipboard before. The clipboard starts off empty.
- Spend Z units of time to add the entire contents of the clipboard to the end of Mr. Panda's string. (The contents of the clipboard do not change.)

Mr. Panda needs to make his string exactly the same as the contents in the thank you letter S. Note that Mr. Panda must create exactly the thank you letter with no additional character.

Mr. Panda wants to find a way to type the thank you letter with the minimum amount of spent time. Because Mr. Panda is too lazy, he asks for your help.

Could you please help Mr. Panda find an optimized way to type the thank you letter so that the amount of time spent is minimized? Note that you just need to tell Mr. Panda the minimum number of time units that are needed.

Input

The first line of the input gives the number of test cases T ($1 \le T \le 100$). T test cases follow.

Each test case starts with a line consisting of four integers n ($1 \le n \le 5000$), the length of Mr. Panda's thank you letter, X, Y and Z ($1 \le X, Y, Z \le 10^9$). X, Y and Z are the time cost of operations that can be performed by the typewriter.

Then, a line consisting of n integers S_0, S_1, …, S_{n-1} follows, denoting the contents of Mr. Panda's thank you letter. Each integer S_i ($1 \le S_i \le 10^9$) represents a single character in the letter.

It is guaranteed that $n \le 1000$ in at least 80% of test cases.

Output

For each test case, output one line containing "`Case #x: y`", where `x` is the test case number (starting from 1) and `y` is the minimum number of time units that are needed to type the thank you letter.

Example

standard input	standard output

2
6 1 1 1
1 2 3 1 2 3
6 10 8 8
1 2 2 1 2 3
	Case #1: 5
	Case #2: 56

Problem D. Pulse Nova

Input file:	`standard input`
Output file:	`standard output`

Mr.Panda is playing a game named Watcher of Samsara, a famous tower defense game developed by a Chinese studio. At the beginning of the game, every player is able to choose their first hero arbitrarily from the hero pool which is randomly generated by the game system. Among all the heroes, Mr.Panda always chooses the hero named Leshrac, because of the character's ultimate skill, Pulse Nova.

Pulse Nova creates waves of damaging energy around Leshrac, one per second, to damage all nearby enemy units. This powerful skill inspires Mr.Panda to come up with a geometry problem.

There are n straight lines on the 2D plane, where the i^{th} line passes two integer points P_i and Q_i. You need to place a circle with radius of exactly R on the plane. For each given line, there might be a segment part of this line that will be inside the circle. Please find a position for this circle to maximize the total length of all segment parts which will be inside the circle.

Input

The first line of the input gives the number of test cases, T $(1 \le T \le 100)$. T test cases follow.

The first line of each test case contains two integers n $(1 \le n \le 50)$ and R $(1 \le R \le 3000)$, the number of straight lines, and the radius of the circle, respectively.

In the next n lines, each contains four integers. In the i^{th} line, the first two integers indicate the 2D coordinates of point P_i, and the last two integers indicate the 2D coordinate of point Q_i. We ensure that $P_i \neq Q_i$ and the absolute value of every coordinate is not greater than 1000.

We ensure the sum of n in all test cases is not greater than 100.

Output

For each test case, output one line containing "`Case #x: y`", where `x` is the test case number (starting from 1) and `y` is the maximum length which will be inside the circle. Your answer will be considered correct if it is within an absolute or relative error of 10^{-6} when compared with the correct answer.

Example

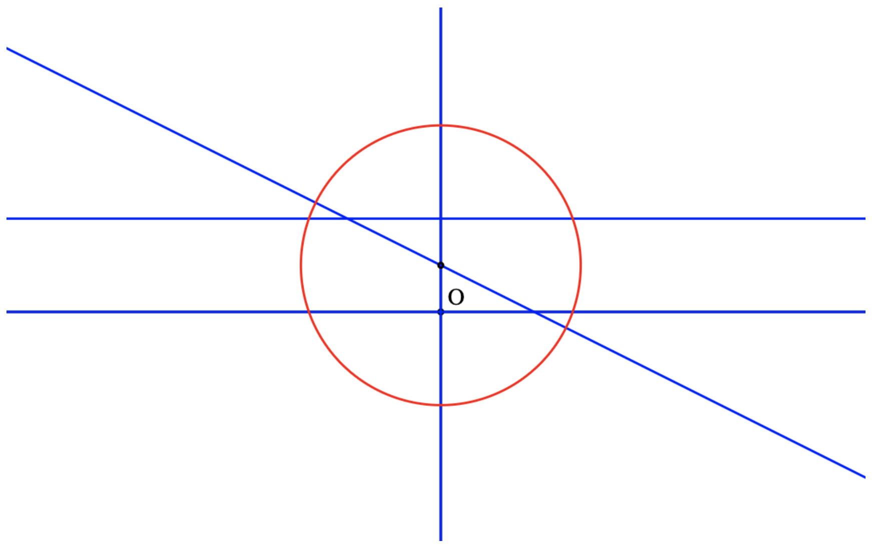
standard input	standard output

3
2 2
1 1 1 2
1 1 2 1
4 3
0 0 0 1
2 0 0 1
0 0 1 0
0 2 1 2
5 4
1 3 -2 3
0 0 4 0
0 1 -1 2
-3 1 2 -1
1 3 2 -3
	Case #1: 8.0000000000
	Case #2: 23.3137084990
	Case #3: 38.0402955628

Note

For the first sample, you can place the center of the circle at $(1, 1)$, so the total length inside the circle will be 8.

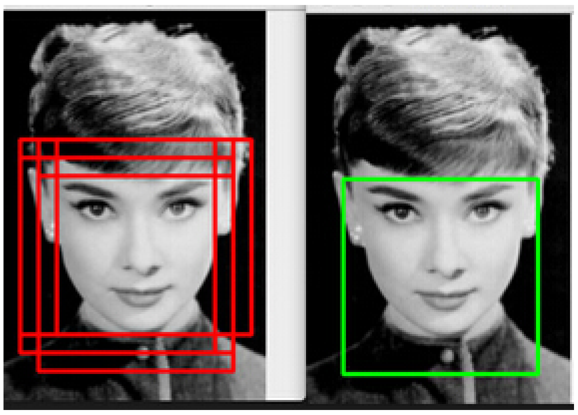
For the second sample, you can place the center of the circle at $(0, 1)$, so the total length inside the circle will be $12 + 8\sqrt{2}$.



Problem E. Non-Maximum Suppression

Input file:	`standard input`
Output file:	`standard output`

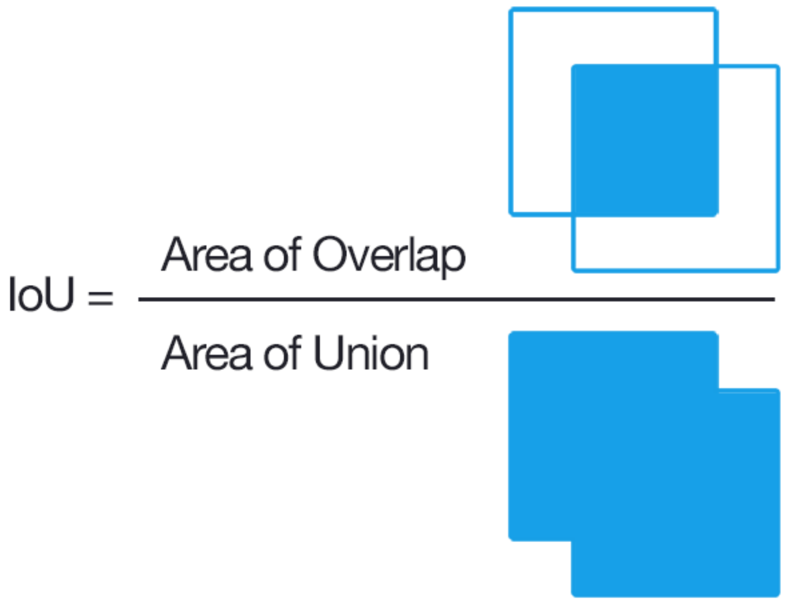
Non-maximum suppression (NMS) has been widely used in several aspects of computer vision and is an integral part of many object detection algorithms. One of the most common problems in object detection is that an object might be detected multiple times. NMS technique ensures we get only a single detection per object.



For each class, a list of bounding boxes is proposed by the algorithm. Each box is associated with a score, indicating the confidence the algorithm thinks there is an object of the given class inside the box. Now let's see how the NMS procedure works. In the beginning, all the boxes are unselected and non-suppressed.

1. First, select the box with the highest confidence score among boxes that are unselected and non-suppressed.

2. Then, look at all the unselected boxes in the image. If the **IoU** of an unselected box and a selected box is **strictly** larger than a given threshold, then the unselected box will be suppressed (discarded). We measure how two boxes are overlapped by the concept of **Intersection over Union (IoU)**.



3. Repeat the above procedure until all non-suppressed boxes are selected.

In this problem, we assume there is only one class, and simplify the object detection algorithm by only proposing **square shaped** boxes of the **same size**. Given the IoU threshold and a list of congruent square boxes proposed by the algorithm, can you report all selected boxes after going through the NMS procedure?

Input

The first line of the input gives the number of test cases, T ($1 \le T \le 10$). T test cases follow.

For each test case, the first line contains two integers n, S ($1 \le n \le 10^5$, $1 \le S \le 10^7$) and a floating-point number *threshold* ($0.300 \le \mathit{threshold} \le 0.700$), indicating the number of square bounding boxes proposed by the detection algorithm, the size of the squares, and the IoU threshold. The *threshold* is exact and is given to the third digit after the decimal point.

In the next n lines, each line contains two integers x, y ($0 \le x < x + S \le 10^7$, $0 \le y < y + S \le 10^7$) — the coordinates of the bottom left corner of a square box, followed by a floating point number *score* ($0.0 \le \mathit{score} \le 1.0$) — the score of this box. Scores are exact and have at most six digits after the decimal point. Also, scores are distinct.

Output

For each test case, the output starts with a line containing "`Case #x: y`", where `x` is the test case number (starting from 1), and `y` is the number of selected bounding boxes after going through the NMS procedure. The next line contains `y` integers indicating the indices of the boxes (starting from 1) in **ascending** order.

Example

standard input	standard output

1
3 4 0.390
0 0 0.9
1 1 0.8
2 2 0.7
	Case #1: 2
	1 3

Note

There are 3 boxes in the example: $[0, 4] \times [0, 4]$, $[1, 5] \times [1, 5]$ and $[2, 6] \times [2, 6]$, which are already in descending order of score. The second box is suppressed because the IoU of the first and the second box is $\frac{9}{23}$, which is strictly larger than 0.390, the IoU threshold. The third box is selected because the IoU of the first and the third box is $\frac{1}{7}$, which is smaller than the IoU threshold.

Problem F. Ferry

Input file:	`standard input`
Output file:	`standard output`

There are three islands named A, B, and C which are located at the corners of an equilateral triangle. There are n visitors initially on island A. Each of the visitors has a destination island w_i which is either island B or island C. There is one ferry boat currently docking at island A. The ferry boat has a fixed route: $A \to B \to C \to A \to B \to C \to A \dots$

Each visitor has an attribute t_i, representing the minimal time to ferry them between any two islands without causing seasick. The ferry boat can carry no more than three people at the same time. To ensure that all people on the boat won't be seasick, the time it takes voyaging between any two islands is determined by the largest t_i of the people on the ferry boat.

Once a visitor arrives at the destination island, the visitor will stay on the island and will not embark on the ferry boat again. Besides, a visitor will only disembark from the ferry boat when arriving at his or her destination. The ferry boat can not set out from the dock if there is nobody on the boat, but luckily we have countless sailors on island A at the beginning. The sailors are so well trained that their attributes are all 1, and unlike the visitors, they have no destination and can visit each island multiple times.

All sailors and the ferry boat should be back to island A after sending all visitors to their destination islands. You need to find out the shortest time to achieve this goal.

Input

The first line of the input gives the number of test cases, T $(1 \le T \le 10)$. T test cases follow.

For each test case, the first line contains an integer n $(1 \le n \le 50000)$, representing the number of visitors.

In the following n lines, each line contains two integers describing a visitor. The first integer w_i represents the visitor's destination island, where w_i is either 1 representing island B, or 2 representing island C. The second integer t_i $(1 \le t_i \le 1000)$ represents the minimal time to ferry the visitor between any two islands.

Output

For each test case, output one line containing "`Case #x: y`", where `x` is the test case number (starting from 1) and `y` is the shortest time to send every visitors to the island they want to reach.

Example

standard input	standard output

2
6
1 1
1 2
1 3
1 2
2 3
2 1
1
1 5
	Case #1: 14
	Case #2: 7

Problem G. Game on the Tree

Input file: `standard input`
Output file: `standard output`

Mr. Panda and Mr. Sheep are playing a game on a tree with n vertices. Initially, there is a token on the vertex **1**. Mr. Panda and Mr. Sheep take turns and Mr.Panda moves first.

In each turn, a player must move the token to another vertex. There is a restriction that except for the first step, the distance the token moved must be strictly greater than the distance it moved in the previous turn by the opponent. If there is no valid move for the turn, the player loses.

Mr. Sheep finds this game might be unfair to him. So Mr. Panda allows Mr. Sheep to choose a connected subgraph of the tree which also contains the vertex 1 along with the token on the vertex. If both Mr. Panda and Mr. Sheep play **optimally**, in how many different ways can Mr. Sheep choose a subgraph for them to play the game on so that he can win? Two ways are considered different if the subgraphs in these two ways differ. As the answer can be very large, you only need to output it modulo $(10^9 + 7)$.

Input

The first line of the input gives the number of test cases, T $(1 \le T \le 10)$. T test cases follow.

For each test case, the first line contains an integer n $(1 \le n \le 2 \times 10^5)$, representing the number of vertices of the tree.

The next $n-1$ lines each contains two integers x and y $(1 \le x, y \le n)$, indicating there is an edge between vertex x and vertex y. It is guaranteed that the given edges form a tree.

Output

For each test case, output one line containing "`Case #x: y`", where `x` is the test case number (starting from 1) and `y` is the number of different winning ways modulo $(10^9 + 7)$.

Example

standard input	standard output

2
2
1 2
6
1 2
2 3
1 4
4 5
4 6
	Case #1: 1
	Case #2: 5

Problem H. Mr. Panda and SAD

Input file:	`standard input`
Output file:	`standard output`

Mr. Panda owns many strings with only uppercase letters. He defines the happiness of a string to be the number of occurrence of "`SAD`" (SAD stands for Shanghai Algorithm Discussion) in the string.

One day Mr. Panda broke one of his precious string inadvertently into n pieces. The i^{th} piece had the string s_i on it. He collected all the pieces and turned to Mr. Sheep for help. As Mr. Panda was very sad about the broken string, Mr. Sheep comforted him that if they concatenated all the pieces into a new string, the happiness of the string could be even greater than its original happiness value! But they really didn't know how to concatenate pieces to get the maximum happiness. It is your turn to help them again.

Input

The first line of the input gives the number of test cases, T $(1 \leq T \leq 100)$. T test cases follow.

Each test case begins with a line containing one integer n $(1 \leq n \leq 2 \times 10^5)$, the number of pieces Mr. Panda's precious string was broken into.

In the next n lines, each line contains a string s_i $(1 \leq |s_i| \leq 20)$ with only uppercase letters.

It is guaranteed that the sum of n in all cases is not greater than 10^6, and the sum of $|s_i|$ in all cases is not greater than 2×10^6.

Output

For each test case, output one line containing "`Case #x: y`", where `x` is the test case number (starting from 1) and `y` is the maximum happiness of the string that can be achieved by concatenating the pieces.

Example

standard input	standard output
3 3 SAD D SA 3 SS A DD 4 DS SA ADSA D	Case #1: 2 Case #2: 1 Case #3: 3

Note

For the first sample, the new string can be concatenated as `SADSAD`, hence the happiness is 2.

For the second sample, the new string can be concatenated as `SSADD`, hence the happiness is 1.

For the third sample, the new string can be concatenated as `SA+DS+ADSA+D` = `SADSADSAD`, hence the happiness is 3.

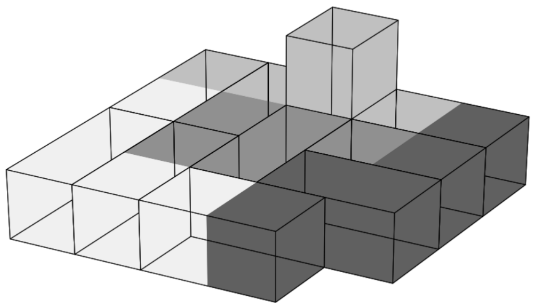
Problem I. Mr. Panda and Blocks

Input file:	`standard input`
Output file:	`standard output`

Mr. Panda recently received a bucket of toy blocks as his birthday gift. Each block is a $1 \times 1 \times 2$ cuboid, which is constructed by a pair of face-to-face $1 \times 1 \times 1$ colored cubes. There are n types of colors, labeled as $1, 2, \ldots, n$.

Mr. Panda checked all of the blocks, and he found that he had just $\frac{n \times (n+1)}{2}$ blocks and each of these blocks was painted with a unique pair of colors. That is, for each pair of colors (i, j) $(1 \le i \le j \le n)$, he had exactly one block with one cube colored i, and the other colored j.

Mr. Panda plans to build a fantastic castle with these blocks today.



Firstly, he defines an attribute called *connected*:

1. If cube A shares a face with cube B, they are connected.
2. If cube A and cube B are connected, and cube B and cube C are connected, then cube A and cube C are connected.
3. If all pairs of cubes in the castle are connected, the castle is connected.

Then he comes up with the following requirements:

1. The whole castle should be connected.
2. For any color i, if only consider the cubes with that color, the sub-castle formed by these cubes should also be connected.

However, after many attempts, Mr. Panda still cannot build such a castle. So he turns to you for help. Could you please help Mr. Panda to build a castle which meets all his requirements?

Input

The first line of the input gives the number of test cases, T $(1 \le T \le 10)$. T test cases follow.

For each test case, one line contains an integer n $(1 \le n \le 200)$, representing the number of colors.

Output

For each test case, first output one line containing "`Case #x:`", where `x` is the test case number (starting from 1).

If it's impossible to build a castle that satisfies Mr. Panda's requirements, output a single line containing "NO" (quotes for clarity).

If it's possible to build the castle, first output a single line containing "YES" (quotes for clarity).

Then, output $\frac{n\times(n+1)}{2}$ lines describing the coordinates of all the blocks. Each of these lines should be outputted in the form of $i, j, x_i, y_i, z_i, x_j, y_j, z_j$ ($1 \le i \le j \le n, 0 \le x_i, y_i, z_i, x_j, y_j, z_j \le 10^9$), which means for the block (i, j), the cube with color i is located at (x_i, y_i, z_i) and the other cube with color j is located at (x_j, y_j, z_j). You should make sure that each pair of (i, j) occurs exactly once in your answer.

In case there is more than one solution, any of them will be accepted.

Example

standard input	standard output
2	Case #1:
3	YES
4	1 1 1 1 0 1 2 0
	1 2 1 3 0 1 4 0
	1 3 2 1 0 3 1 0
	2 2 2 2 0 2 3 0
	2 3 2 4 0 3 4 0
	3 3 3 2 0 3 3 0
	Case #2:
	YES
	1 2 1 3 0 1 4 0
	1 1 1 2 0 1 1 0
	1 3 2 1 0 2 2 0
	2 3 2 4 0 2 3 0
	1 4 3 1 0 4 1 0
	3 3 3 2 0 3 3 0
	2 2 3 4 0 3 4 1
	4 4 4 2 0 5 2 0
	3 4 4 3 0 5 3 0
	2 4 4 4 0 5 4 0

Problem J. Wire-compatible Protocol buffer

Input file: `standard input`
Output file: `standard output`

Protocol buffers (or protobuf) are a flexible, efficient, automated mechanism for serializing structured data – think XML, but smaller, faster and simpler. In this problem, we will discuss the wile-compatible problem on a simplified protocol buffer.

Protobuf descriptor

Unlike XML or JSON, which is schema-less, protobuf has a schema. Before actually serializing data, you need to define the structure of the data you'd like to serialize in the descriptor file. On the left side of the following is a basic example of a protobuf descriptor that defines a message containing information about a Car.

```
message Car {
    required string brand = 1 ;
    required string model = 2 ;
    required double max_mileage = 4 ;
    optional double max_speed = 3 ;

    repeated Accessory accessories = 7 ;
}

message Accessory {
    required string name = 1 ;
    optional string description = 2 ;
    required double price = 3 ;
}
```

```
brand: "Tesla"
model: "Model 3"
max_mileage: 595
max_speed: 261
accessories {
    name: "Body painting"
    desctiption: "(Blue)"
    price: 10000.0
}
accessories {
    name: "Autopilot model"
    price: 56000.0
}
```

Given the protobuf descriptor, we can define protobuf instances, for example, an instance of a Car protobuf is on the right side of the above.

As you can see, the message format is simple – each message type has one or more uniquely numbered fields, and each field has a name and a value type, where value types can be integers float numbers, booleans, strings, raw bytes, or even other protocol buffer message types (to simplify, in this problem we only consider doubles, strings and embedded protobuf messages like `Accessory` in the example), allowing you to structure your data hierarchically.

The descriptor of protobuf contains a set of messages. Each message contains a set of numbered fields. Each field has its own field rules (required, optional or repeated), type, and a field name (e.g. `brand`, `model`, `max_mileage` in the example).

Field rules

Each field has one of the following rules:

- required: A well-formed message must have exactly one of this field.
- optional: A well-formed message can have zero or one of this field (but no more than one).
- repeated: This field can be repeated any number of times (including zero) in a well-formed message. The order of the repeated values will be preserved.

Field types

In this problem, we only consider the following types:

- double: 64 bit IEEE 754 double-precision Float Point Number.
- string: UTF-8 encoded string of any length.
- message: other protobuf message types defined in the descriptor.

Field numbers

Each field is assigned with a unique number, which is an integer between 1 and $2^{29} - 1$, or 536,870,911.

Syntax

Formally, the syntax of the simplified protobuf can be specified using Extended Backus-Naur Form (EBNF):

```
|   alternation
()  grouping
[]  option (zero or one time)
{}  repetition (any number of times)

letter = "A" ... "Z" | "a" ... "z"
digit = "0" ... "9"

ident = letter { letter | digit | "_" }
messageName = ident
fieldName = ident
messageType = messageName

decimal = ("1" ... "9") { digit }
fieldNumber = decimal

label = "required" | "optional" | "repeated"
type = "double" | "string" | messageType
field = label type fieldName "=" fieldNumber ";"
messageBody = "{" { field } "}"
message = "message" messageName messageBody

descriptor = { message }
```

Note: message names and field names are case sensitive and cannot be reserved words ("message", "required", "optional", "repeated", "double", "string").

Protobuf encoding

In this section, we will discuss how to serialize protobuf instances into bytes.

Here is a simple example. Given a simple protobuf descriptor:

```
message Test1 {
    optional double a = 1 ;
    repeated string b = 2 ;
}
```

And an instance:

```
a: 1.0
b: "abc"
b: "ABC"
```

Its wire-format encoding is:

```
09 3f f0 00 00 00 00 00 00 12 03 61 62 63 12 03 41 42 43
```

Base 128 Varints

In this section, we will take a look at how an instance is encoded into a byte stream.

To understand a simple protocol buffer encoding, you first need to understand varints. Varints are a method of serializing integers using one or more bytes. Smaller numbers take a smaller number of bytes.

Each byte in a varint, except the last byte, has the most significant bit (msb) set – this indicates there are further bytes to come. The lower 7 bits of each byte are used to store the two's complement representation of the number in groups of 7 bits, least significant group first.

So, for example, here is the number 1 – it's a single byte, so the msb isn't set:

```
0000 0001
```

And here is 300 – this is a bit more complicated:

```
1010 1100 0000 0010
```

How do you figure out this is 300? First, you drop the msb from each byte, as this is just there to tell us whether we've reached the end of the number (as you can see, it's set in the first byte as there is more than one byte in the varint):

```
    1010 1100 0000 0010
->   010 1100  000 0010
```

You reverse the two groups of 7 bits because, as you remember, varints store numbers with the least significant group first. Then you concatenate them to get your final value:

```
000 0010  010 1100
->  000 0010 ++ 010 1100
->  100101100
->  256 + 32 + 8 + 4 = 300
```

Message structure

As you know, a protocol buffer message is a series of key-value pairs. The binary version of a message just uses the field's number as the key – the name and declared type for each field can only be determined on the decoding end by referencing the message type's descriptor.

When a message is encoded, the keys and values are concatenated into a byte stream. The "key" for each pair in a wire-format message is actually two values – the field number from your protobuf descriptor file, plus a wire type that provides just enough information to find the length of the following value. In most language implementations this key is referred to as a tag.

The available wire types are as follows:

Type	Meaning	Used For
0	Varint	int32, etc. (Not related to this problem)
1	64-bit	double, etc.
2	Length-delimited	strings, embedded messages, etc.
3	Start group	groups. (Not related to this problem)
4	End group	groups. (Not related to this problem)
5	32-bit	float, etc. (Not related to this problem)

Each key in the streamed message is a varint with the value

```
 (field_number << 3) | wire_type
```

Now let's look at our simple example again. You now know the first number in the stream is always a varint key, and here it's 09, or (dropping the msb):

```
000 1001
```

You take the last three bits to get the wire type (1) and then right-shift by three to get the field number (1). So you now know the field number is 1 and the following value is a 64-bit. Then, according to the descriptor, we know the following 8 bytes:

```
3f f0 00 00 00 00 00 00
```

is a double which is the representation of 1.0 in IEEE 754.

String

A wire type of 2 (length-delimited) means the value is a varint encoded length followed by the specified number of bytes of data.

Let's look at simple examples again:

```
12 03 61 62 63
12 03 41 42 43
```

The key here is 0x12, where field number = 2, wire type = 2. The length varint is 3, and then 0x61 0x62 0x63 is the ascii code for "abc", and 0x41 0x42 0x43 is the ascii code for "ABC".

Embedded messages

Here's a message descriptor with an embedded message of our example type, Test2:

```
message Test2 {
  optional Test1 c = 1;
}
```

And here's the encoded version, with Test1's a field set to 1.0:

```
0a 09 09 3f f0 00 00 00 00 00 00
```

The key is 0x0a, field number = 1, wire type = 2. The length varint is 9, and then the following 9 bytes is a Test1 instance, with a = 1.0.

Optional and repeated elements

If the protobuf descriptor has repeated elements, the encoded message can have zero or more key-value pairs with the same field number. These repeated values do not have to appear consecutively; they may be interleaved with other fields. The order of the elements with respect to each other is preserved when parsing.

For any optional fields, the encoded message may or may not have a key-value pair with that field number.

Field Order

Field numbers may be used in any order in a descriptor. The order chosen has no effect on how the messages are serialized.

Wire-format compatible problem

If message A is wire-format compatible with message B, it means the serialized bytes of any instance of message A can be parsed by message B without breaking the protobuf encoding assumption (and no unknown fields), and vice versa.

Input

The first line contains an integer n indicates the number of lines of a protobuf descriptor.

The following n lines are the content of the protobuf descriptor which contains a set of messages. Each line will not contain more than 120 characters.

Then follows a line with an integer m ($1 \le m \le 50000$), indicating there are m wire-format compatibility queries.

Then follow m lines, each of which is a wire-format compatibility query with two message names.

It is guaranteed that the descriptor is valid, in other words, no two messages have the same name, no two fields in a message have the same field name or tag number, and each message type in a field must be one of the existing message names.

There will be at most 1000 messages in the descriptor, and each message will have at most 16 fields.

All tokens of the input are separated by spaces.

Output

For each query, output one line containing "Wire-format compatible." (quotes for clarity) if the two messages in this query are wire-format compatible, or otherwise, "Wire-format incompatible." (quotes for clarity)

The sample input/output will be on the next page.

Examples

standard input	standard output

`18`
`message Test1 {`
`  optional string field = 1 ;`
`}`
`message Test2 {`
`  optional string field_string = 1 ;`
`}`
`message Test3 {`
`  optional string field = 2 ;`
`}`
`message Test4 {`
`  required string field = 1 ;`
`}`
`message StringMessage {`
`  optional string field = 1 ;`
`}`
`message Test5 {`
`  optional StringMessage field = 1 ;`
`}`
`4`
`Test1 Test2`
`Test1 Test3`
`Test1 Test4`
`Test1 Test5`
	`Wire-format compatible.`
	`Wire-format incompatible.`
	`Wire-format incompatible.`
	`Wire-format incompatible.`

`5`
`message A { optional B nest = 1 ; }`
`message B { optional C nest = 1 ; }`
`message C { }`
`message D { optional E nest = 1 ; }`
`message E { }`
`2`
`B D`
`A D`
	`Wire-format compatible.`
	`Wire-format incompatible.`

`3`
`message A { optional A nest = 1 ; }`
`message B { optional C nest = 1 ; }`
`message C { optional B nest = 1 ; }`
`3`
`A B`
`A C`
`B C`
	`Wire-format compatible.`
	`Wire-format compatible.`
	`Wire-format compatible.`

Note

In the first example:
For Test1 and Test2, though the two messages have different field names, the serialized message just cares about their field numbers.
For Test1 and Test3, the same field name as the two messages have, their field numbers do not match.
For Test1 and Test4, obviously required is incompatible with optional.
For Test1 and Test5, not all valid UTF-8 strings are valid messages, and vice versa.

Problem K. Russian Dolls on the Christmas Tree

Input file:	`standard input`
Output file:	`standard output`

Christmas is still one month away, but Mr. Panda already starts the Christmas preparation. Mr. Panda is decorating a Christmas tree with a set of Russian dolls. There are n Russian dolls numbered $1, 2, \ldots, n$. The i^{th} doll is designed to be perfectly nested inside the $(i+1)^{th}$ doll for all $1 \le i \le n-1$. Nesting dolls are stable only if they have neighboring ordinal numbers, otherwise the smaller one will slide out from the larger one. Dolls can be nested recursively. For example, the n dolls can be nested all the way up from smallest to largest until there is only one doll left.

The Christmas tree happens to have n nodes with one Russian doll dangling on each node, where the doll numbered 1 is put at the tree root. Mr. Panda will invite his friend Mr. Sheep to collect some dolls from the Christmas tree as gifts on Christmas Eve. Mr. Sheep will pick a tree node and collect all the dolls in the sub-tree with the selected node as the sub-tree root.

As there could be a lot of dolls, Mr. Sheep want to nest the dolls he collects for easy carrying. He wonders for each tree node, how many dolls will be ended up if he nests them as many as possible. Note that the dolls should be stably nested.

Input

The first line of input gives the number of test cases T $(1 \le T \le 10)$. T test cases follow.

Each test case starts with a line containing an integer n $(1 \le n \le 2 \times 10^5)$, the number of dolls and also the number of tree nodes.

The next $(n-1)$ lines each contains two integers x and y $(1 \le x, y \le n)$, denoting the dolls numbered x and y are neighbors in the Christmas tree.

It is guaranteed that the sum of n in all cases is not greater than 10^6.

Output

For each test case, the output consists one line starts with "`Case #x:`", where `x` is the test case number (starting from 1), followed by next n integers. The i^{th} $(1 \le i \le n)$ integer indicates the number of dolls will be ended up if Mr. Sheep selects the tree node that contains the doll numbered i, collects all the dolls in the sub-tree, and nests the dolls stably as many as possible.

Example

standard input	standard output

1
7
1 2
2 4
2 6
1 3
3 5
3 7
	Case #1: 1 3 3 1 1 1 1

Problem L. Spiral Matrix

Input file:	`standard input`
Output file:	`standard output`

Lee got a ticket to the Google Developer Day. When he came to the exhibition hall, he found that the booths were located in a matrix of size $n \times m$.

Unfortunately, Lee had badly sprained his left ankle a week before the GDD when playing basketball. As a result, he couldn't wander freely as he wanted. His ankle would get hurt if he tried to turn left. Lee also didn't want to make a big right turn by turning right multiple times, which would make him look stupid.

Lee wanted to visit each booth exactly once. Given his ankle situation, he made some rules visiting the booths. He could start with any booth in the exhibition hall, and choose an initial direction. Then each move would have to follow one of the possible moves:

1. Go straight to visit the next booth.
2. Turn right once and then go straight to visit the next booth.

You are a friend of Lee and you have the good habit of helping him. Can you find out the number of different ways to visit each booth exactly once? Two ways are considered different if and only if the visiting orders in these two ways vary.

Input

The first line of the input gives the number of test cases, T $(1 \le T \le 100)$. T test cases follow.

For each test case, the first and only line contains two integers n and m $(1 \le n, m \le 100)$, the number of rows and the number of columns of the matrix, respectively.

Output

For each test case, output one line containing "`Case #x: y`", where `x` is the test case number (starting from 1), and `y` is the number of different ways modulo $(10^9 + 7)$.

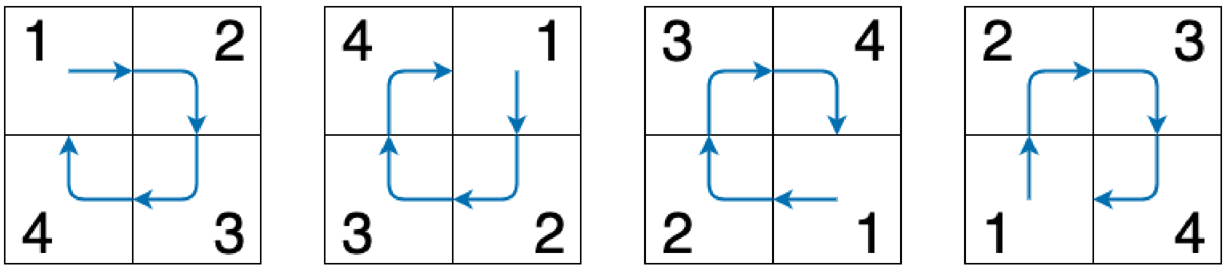
Example

standard input	standard output

`1`
`2 2`
	`Case #1: 4`

Note

Here are the 4 different ways for $n = 2, m = 2$.


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