

Problem A. Colorful Tree

Input file: `stdin`
Output file: `stdout`
Time limit: 1 second
Memory limit: 256 megabytes

It is an interesting problem in graph theory called “graph coloring”.

In graph theory, graph coloring is a special case of graph labeling; it is an assignment of labels traditionally called “colors” to elements of a graph subject to certain constraints. In its simplest form, it is a way of coloring the vertices of a graph such that no two adjacent vertices share the same color; this is called a vertex coloring. (from Wikipedia)

The smallest number of colors needed to color a graph is called its chromatic number.

THE STUDENT ASKS: I wonder if this problem is easy on trees.

THE TEACHER EXPLAINS: Well, two colors is enough for trees.

But the critical thinking student thought he should better check that. He thought it is too simple to be true.

Note: the tree is a connected graph with no cycles.

Input

The first line contains the number of test cases T ($0 \leq T \leq 20$). The first line of each test case contains the number of vertices in the tree, n ($1 \leq n \leq 10^5$). The following $n - 1$ lines contains numbers a and b ($1 \leq a < b \leq n$), meaning that vertices a and b are connected.

Output

For each test case output one line containing “Case #tc: num”, where tc is the number of the test case (starting from 1) and num is the chromatic number of the given tree.

Examples

stdin	stdout
1 2 1 2	Case #1: 2

Problem B. Farmer

Input file: `stdin`
Output file: `stdout`
Time limit: 1 second
Memory limit: 256 megabytes

John is a successful farmer and he would like to expand his business. For this reason he is going to buy a new plot of land to grow even more crops (and earn even more money). Currently there are T ($0 \leq T \leq 1000$) plots on sale and John wants to find the best deal. He considers deal the best if the price per area unit is the lowest. Can you help him by coding a solution that computes price per area unit?

Input

First line contains the number of plots T ($0 \leq T \leq 1000$). Each plot defined is as a quadrilateral by 4 integer points on the 2D plane (in clockwise or counterclockwise order), meaning there are 8 integers (32-bit) in total which describe geometry and location of the plot. Last number on the line represents the price of plot.

Output

For each plot output a line “**Case #tc: x**”, where **tc** is plot’s sequence number (starting from 1) and **x** is price per unit for the **tc**-th plot rounded to two decimal places.

Examples

stdin	stdout
1 0 0 0 10 10 10 10 0 100	Case #1: 1.00

Problem C. Letter Array

Input file: `stdin`
 Output file: `stdout`
 Time limit: 1 second
 Memory limit: 256 megabytes

Jonas is reading the book called “Algorithms and Data Structures (for extremely experienced computer scientists edition)”. He learned a new data structure - the array of lowercase letters. As Jonas is innovative and you are his good friend, he assigned you the task to implement and to use the array of uppercase letters. This data structure is mainly a string with only uppercase letters which can operate two types of queries:

- “*s l r*” - sorts letters in the interval $[l; r]$ ($0 \leq l \leq r < \text{length of string}$);
- “*g l r*” - prints frequencies of all letters from A to Z in the interval $[l; r]$.

Input

The first line contains the number of test cases T ($T \leq 5$). The first line of each test case contains the string of only uppercase letters ($1 \leq \text{thelength} \leq 10^5$). The second line of each test case contains the number of queries q ($0 \leq q \leq 6000$). The following q lines contain “*s l r*” or “*g l r*” (quotes for clarity only).

Note: the sum of all $r - l$ in the queries of “Sort” won’t exceed 10^5 in the whole input file.

Output

For each test case output one line containing “**Case #tc:**”, where **tc** is the number of the test case (starting from 1). Then for each query “*g l r*” output a line containing 26 numbers (frequencies of letters from A to Z in the given subinterval $[l; r]$).

Examples

stdin	stdout
1	Case #1:
BCA	0 1 0
11	0 0 0 0 0 0 0
g 0 0	0 0 1 0
g 1 1	0 0 0 0 0 0 0
g 2 2	1 0
s 1 2	0 0 0 0 0 0 0
g 1 1	1 0
g 2 2	0 0 0 0 0 0 0
s 0 2	0 0 1 0
g 1 2	0 0 0 0 0 0 0
g 0 0	0 1 1 0
g 1 1	0 0 0 0 0 0 0
g 2 2	1 0
	0 0 0 0 0 0 0
	0 1 0
	0 0 0 0 0 0 0
	0 0 1 0
	0 0 0 0 0 0 0

Problem D. Lightning

Input file: `stdin`
Output file: `stdout`
Time limit: 1 second
Memory limit: 256 megabytes

The lightning usually hits towers, tall buildings. Have you noticed that? The thing is that they are near the “source of power”.

We will use this lightning model (which may be not true in real world). Every direct path in which the lightning can strike is characterized by its non-resistance (the bigger non-resistance, the bigger power of the lightning) and its length.

There are n objects (k of them are the targets at which the lightning can strike). We say the path has power x if the minimum non-resistance along the path is x .

The lightning could be modelled as follows:

1. The maximum power P the lightning can strike is calculated. It is the minimum non-resistance of a path with the biggest power from the source of power to the one of the k targets.
2. Then the minimum distance is calculated which has the path with power P from the source of power to the one of the k targets. It's guaranteed that it will be possible to find at least one such path.
3. All targets which have the path with minimum found distance with power P are called dangerous (because they are open for the strike).

Input

The first line contains the number of test cases T ($0 \leq T \leq 20$). The first line of each test case contains the number of objects n ($2 \leq n \leq 10^4$) and the number of targets k ($1 \leq k < n$). Targets are objects numbered from 1 to k . The source of power is the object numbered n . The second line of each test case contains the number of direct paths m ($1 \leq m \leq 10^4$). The following m lines contains the numbers a, b ($1 \leq a < b \leq n$) and integers d, nr ($0 < d \leq 10^5, 0 < nr \leq 10^6$) meaning that objects numbered a and b are connected by the direct path (with the distance d , non-resistance nr). Each direct path can be used by the lightning bidirectionally.

Note: the same pair of objects can be connected by more than one direct path.

Output

For each test case output one line containing “**Case #tc:**”, where **tc** is the number of the test case (starting from 1). Then output all the targets which are in danger in the ascending order (separated by spaces).

Examples

stdin	stdout
1 3 2 2 3 1 2 3 2 3 1 2	Case #1: 1

Problem E. Magical Code

Input file: `stdin`
Output file: `stdout`
Time limit: 1 second
Memory limit: 256 megabytes

Do you know how magicians cast spells? The spell consists of its name and its magical code. The casting is done by saying its name while keeping in mind the magical code (only then the spell works). Only the most experienced, the smartest and the most talented magicians can create their own spells. They name their spell and determine its purpose. They make some rituals near the Tree of Power. And if the magician is worth the tree creates the magical code for this spell.

However, the tree itself is designed and created by the mathematician who told me the mechanism of creating spells. Here is the algorithm in “our language”:

```
const int MOD = 1013;
int code = 0;
for (int i = 0; i < spell_name.size(); i++)
    code = (code * 31 + spell_name[i]) % MOD;
printf("You are worth it! Your power is unleashed by the number %d" , code);
```

The creator of the tree wants to boost your knowledge. So he tells some of the most recent codes and parts of the spell's name. You need to tell what the maximum number of spells could have this magical code. If there is only one choice - print it!

Note: the spell's name consists only of lowercase English alphabet letters ('a'-'z').

Input

The first line contains the number of test cases T ($T \leq 20$). The first line of each test case contains the magical code, c ($0 \leq c < MOD$). The second line of each test case contains the spell's name (unknown letters are given as '?') (the name is no longer than 10^4 characters).

Note: the number of unknown letters in one spell's name won't exceed 10.

Output

For each test case output one line containing “Case #**tc**: **answer**”, where **tc** is the number of the test case (starting from 1). If there is one possible answer - print the name of the spell as the “**answer**”, otherwise, print the number of ways.

Examples

stdin	stdout
2	Case #1: foobar
31	Case #2: 16
fo?bar	
31	
f???ar	

Problem F. Magician Wars

Input file: `stdin`
Output file: `stdout`
Time limit: 1 second
Memory limit: 256 megabytes

Two powerful magicians “The king of fire” and “The hero of ice” are fighting each other to conquer the world. The winner will be the one who is the smarter and... a bit more successful.

The fight between two greatest magicians involves a lot of strategy because magicians can cast only limited amount of spells during each moment of time.

You support the king of fire just because the summer has just ended and you want to swim in the sea and to play volleyball in the beach... Ah, these good summer days! It may be possible again with the power of fire!

Each magician is characterized by *hp* (health points), *atk* (the strength of the attack), *def* (the defence abilities). Each spell is one of the following types:

- “Heal *x*”, which changes *hp* to $\min(\text{current_hp} + x; 10)$;
- “Atk *x*”, which changes *atk* to $\max(\text{current_atk}; x)$;
- “Def *x*”, which changes *def* to $\max(\text{current_def}; x)$;
- “HIT!”, which deals damage to the opponent which equals $\max(0; \text{attacking_atk} - \text{defending_def})$;

Opponent loses the dealt damage points from his *hp*. If player’s *hp* becomes less than or equal to 0 he is considered dead and his opponent is a winner.

Each player starts with *hp* = 10, *atk* = 0, *def* = 0. Luckily the king of fire moves first. It may increase our chances!

Each player has a sequence of 10 spells. He can reuse the same spell again. During the first turn a magician can use the first 5 spells, during the second - the first 6 spells and so on... During the turn a magician can cast only one spell.

Note: assume that both magicians are very wise and don’t make mistakes, i.e. they play optimally well. It means that a magician tries to minimize the number of turns if it is possible to win. If it is impossible to win, then a magician tries to maximize the number of turns.

Note: it is guaranteed that the winner exists and the game ends in at most 12 turns (that’s it, at most 6 turns for one player).

Input

The first line contains the number of test cases *T* ($0 \leq T \leq 20$). The first 10 lines of each test case contain the spells of the king of fire (“Heal *x*”, “Atk *x*”, “Def *x*”, “HIT!”), $0 \leq x \leq 5$. The other 10 lines of each test case contain the spells of the hero of ice (“Heal *x*”, “Atk *x*”, “Def *x*”, “HIT!”), $0 \leq x \leq 5$.

Output

For each test case output one line containing “Case #*tc*: **status** **turn**”, where *tc* is the number of the test case (starting from 1), **status** is “win” or “lose” (“win” if the king of fire wins, “lose” otherwise), **turn** is the turn in which the final hit was made.

Examples

stdin	stdout
1 Atk 5 HIT! HIT! HIT! HIT! HIT! HIT! HIT! HIT! HIT! Def 3 Def 2 Def 2 Def 2 Def 2 Def 2 Def 2 Def 2 Def 2 Def 2 Def 2	Case #1: win 11

Problem G. Pairs

Input file: `stdin`
Output file: `stdout`
Time limit: 1 second
Memory limit: 256 megabytes

The pair in the land of 1000000007 is the pair of numbers a and b with the following relation: $a \times b = 1 \pmod{1000000007}$. It is believed to be the sign of divine. Your task is to increase the number of happy marriages so you must find the maximum number of distinct pairs! Distinct pairs can not have the same number (with the same index). Pairs are different when the sets of their indices are different

Input

The first line contains the number of test cases T ($T \leq 5$). The first line of each test case contains the size of population, n ($1 \leq n \leq 30000$). The following n lines contain the numbers a_i ($1 \leq a_i < 1000000007$).

Output

For each test case output one line containing “Case #**tc**: **num**”, where **tc** is the number of the test case (starting from 1) and **num** is the maximum number of distinct pairs.

Examples

stdin	stdout
1 5 1 1 1 2 500000004	Case #1: 2

Problem H. Real Magic

Input file: `stdin`
Output file: `stdout`
Time limit: 1 second
Memory limit: 256 megabytes

People like magic. Magic is powerful, interesting and mysterious. However, the real magicians are not more than talented cheaters. One of the most famous magicians is David Foolfield.

For example, suppose there is a room with a tied magician. Then the doors are closed, opened again - there is no magician and he appears in some other room. The thing we don't know that there are secret doors that connect rooms with each other. Of course, he may use another cheap trick - to hide in the same room and his assistant that looks the same appears in the other room. However, we assume that there are no assistants in this problem.

We finally figured out the secret paths between the rooms. Now we want to calculate in how many rooms (other than the initial) the magician can appear. The magician can appear in any room which can be reached from the starting room directly or indirectly.

Input

The first line contains the number of test cases T ($T \leq 30$). The first line of each test case contains the number of rooms in the building n and the number of secret paths m ($1 \leq n, m \leq 10^4$). The following m lines contains numbers a and b ($1 \leq a \leq b \leq n$), meaning that rooms a and b are connected.

Output

For each test case output one line containing "**Case #tc:**", where **tc** is the number of the test case (starting from 1). Then output one additional line containing n numbers separated by spaces which show in how many rooms the magician can appear if he starts at the i -th room (for all i from 1 to n).

Examples

stdin	stdout
1 2 1 1 2	Case #1: 1 1

Problem I. Searching

Input file: `stdin`
Output file: `stdout`
Time limit: 1 second
Memory limit: 256 megabytes

You have a simple system in which you want to have a search function. You decided to implement it for searching email addresses. There is a bunch of email addresses in your program.

User enters an email address. And the function detects if there is one. If yes, then its answer is “yes”, otherwise it tries to change one symbol in the entered address (the error detection). Then, if it matches only one address, the answer is still “yes”. But if it detects more than one address or does not detect anything, the answer is “no”.

Input

The first line contains the number of test cases T ($T \leq 100$). The first line of each test case contains the number of email addresses in the program n and the number of queries m ($1 \leq n, m \leq 10$). The following n lines contain email addresses (only lowercase letters, digits and symbols “@”, “.”). The following m lines contain an email address, which is searched (only lowercase letters, digits and symbols “@”, “.”).

Note: the length of email addresses doesn't exceed 100.

Output

For each test case output one line containing “**Case #tc:**”, where **tc** is the number of the test case (starting from 1). Then output m lines containing answers to the queries (“yes” or “no”).

Examples

stdin	stdout
1	Case #1:
3 4	yes
terminator625@gmail.com	yes
catman@yahoo.co.uk	no
catmug@yahoo.co.uk	no
terminator625@gmail.com	
termonator625@gmail.com	
terminator625@gmale.com	
catmag@yahoo.co.uk	

Problem J. Triangles

Input file: `stdin`
Output file: `stdout`
Time limit: 1 second
Memory limit: 256 megabytes

Tom has just found out that he can't make a triangle using three line segments of lengths 1, 2 and 5. He has lots of different line segments and wants to know how many different triangles he can make. The triangles are different if the sets of their line segments' lengths are different.

Input

The first line contains the number of test cases T ($T \leq 50$). The first line of each test case contains the number of segments N ($N \leq 100$). The second line of each test case contains N integers a_i separated by spaces, a_i is the length of i -th line segment ($1 \leq a_i \leq 10^5$).

Output

For each test case output one line containing "Case #**tc**: **num**", where **tc** is the number of the test case (starting from 1) and **num** is the number of distinct triangles Tom can make.

Examples

stdin	stdout
2	Case #1: 1
3	Case #2: 2
2 2 3	
5	
5 5 5 5 7	