The 2017 National Collegiate Programming Contest

Hosted by CUET

10th March 2017

You get 21 Pages

11 Problems

&

300 Minutes

Rules for NCPC 2017 at CUET, Onsite Contest:

a) Solutions to problems submitted for judging are called runs. Each run is judged as accepted or rejected by the judge, and the team is notified of the results.

b) Notification of accepted runs will **NOT** be suspended at the last one hour of the contest time to keep the final results secret. Notification of rejected runs will also continue until the end of the contest. But the teams will not be given any balloon and the public rank list will not be updated as usual in the last one hour.

c) A contestant may submit a clarification request to judges only through the CodeMarshal clarification system. If the judges agree that an ambiguity or error exists, a clarification will be issued to all contestants. Judges may prefer not to answer a clarification at all in which case that particular clarification request will be marked as IGNORED in the CodeMarshal clarification page.

d) Contestants are not to converse with anyone except members of their team and personnel designated by the organizing committee while seated at the team desk. **They cannot even talk with their team members when they are walking around the contest floor to have food or any other purpose.** Systems support staff or judges may advise contestants on system-related problems such as explaining system error messages.

e) While the contest is scheduled for a particular time length (five hours), the chief judge or the judging director has the authority to alter the length of the contest in the event of unforeseen difficulties. Should the contest duration be altered, every attempt will be made to notify contestants in a timely and uniform manner.

f) **A team may be disqualified by the** chief judge or the judging director for any activity that jeopardizes the contest such as dislodging extension cords, unauthorized modification of contest materials, distracting behavior or communicating with other teams. The **judges on the contest floor** will report to the **Judging Director** about distracting behavior of any team. **The judges can also recommend penalizing a team with additional penalty minutes for their distracting behavior.**

g) Eleven problems will be posed. So far as possible, problems will avoid dependence on detailed knowledge of a particular applications area or particular contest language. Of these problems at least one will be solvable by a first year computer science student, another one will be solvable by a second year computer science student and rest will determine the winner.

h) Contestants will have foods available in their contest room during the contest. So they cannot leave the contest room during the contest without explicit permission from the judges. **The contestants are not allowed to communicate with any contestant (even contestants of his own team) or coaches when they are outside the contest arena.**

i) Teams can bring **printed materials** with them and they can also bring five additional books. But they are not allowed to bring calculators or any machine-readable devices like CD, DVD, Pen-drive, IPOD, MP3/MP4 players, floppy disks etc. **Mobile phone MUST be switched off at all times and stored inside a bag or any other place that is publicly non visible during the entire contest time. Failure to adherence to this clause under any condition will very likely lead to strict disciplinary retaliation and possible disqualification.**

j) With the help of the volunteers, the contestants can have printouts of their codes for debugging purposes. **Passing of printed codes to other teams is strictly prohibited.**

k) The decision of the judges is final.

l) Teams should inform the volunteers/judges if they don't get verdict from the codemarshal within 5 minutes of submission. Teams should also notify the volunteers if they cannot log in into the CodeMarshal system. This sort of complains will not be entertained after the contest.

A Independence Day

Input: Standard Input
Output: Standard Output

Ratul is my younger brother. He is the new decade's citizen of Bangladesh. March is the special month for Ratul because in each March he decorates his house with red and green colors. This month is not only important for Ratul but it is also a memorable month for all Bangladeshi. Ratul doesn't have enough knowledge about Independence day. He is a very Kenny boy, so he wants to know about the Independence day from his father. His father briefly described the Bangladesh Liberation War and the Independence day.

The Independence Day of Bangladesh is on March 26 . It is for the country's declaration of independence from Pakistan. Our undaunted leaders and freedom-loving people did not let the matter go unchallenged. They rose up with great determination and started a vigorous movement against them. A lot of blood was shed in different movements. At last, the war of liberation was waged on the midnight of March 25, 1971. The war continued for nine long months. More than 30 lacs of people sacrificed their valuable lives for the cause of freedom. This bloodshed of our heroic people did not go in vain. We won!

The **Bir Sreshtho, is the highest military award of Bangladesh. It was awarded to seven freedom fighters who showed utmost bravery and died in action for their nation.**

Now Ratul can't take his emotion and he makes a little dictionary and wanted to memorize it. Ratul first memorized the country name **"Bangladesh"** then the seven freedom fighter's name in the order given below:

"Bir Sreshtho Mohiuddin Jahangir", "Bir Sreshtho Hamidur Rahman", "Bir Sreshtho Mostafa Kamal", "Bir Sreshtho Mohammad Ruhul Amin", "Bir Sreshtho Matiur Rahman", "Bir Sreshtho Munshi Abdur Rouf" and "Bir Sreshtho Noor Mohammad Sheikh".

Now as a young programmer your task is to solve a simple problem related to the memorable dictionary. Given the memorized order number **N,** you need to print the N-th name in the memorized dictionary .

Input

Input starts with an integer $T (\leq 100)$ denoting the number of test cases.
Each case starts with an integer $N (1 \leq N \leq 8)$, where N is the memorize order number.

Output

For each case, print the case number first, then print the N-th memorized name in the dictionary. You have to follow the exact format as sample input and output.

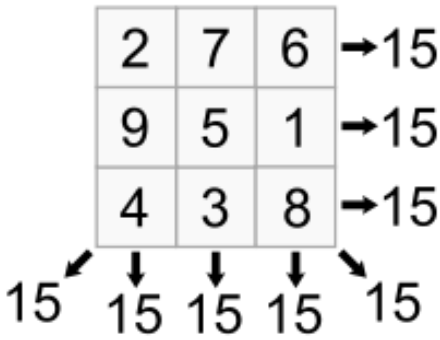
Sample Input	Output for Sample Input
2 3 4	Case 1: Bir Sreshtho Hamidur Rahman Case 2: Bir Sreshtho Mostafa Kamal

B

Magic Tree

Input: Standard Input
Output: Standard Output

All of you know about magic square. Your childhood is incomplete if you haven't played with magic squares!



A magic square is a $n \times n$ grid, each containing a distinct integer between 1 to $n \times n$. The sum of integers of each row, column and diagonal are equal(See picture).

But this problem is not about magic squares, it's about magic trees! (A tree is a connected graph with n nodes and n-1 edges). Each node of a magic tree contains a positive integer (not necessarily distinct). The sum of integers within a path from any leaf node to the root node are same. This sum is called the magic number for that magic tree.

You are given a tree. You want to decorate it as a magic tree. To decorate a tree:

- First, you must choose a node as the root.
- Write a positive integer in each node so that the tree becomes a magic tree.

As decorating a tree is expensive, you must make the magic number as small as possible. Find the minimum possible magic number for the magic tree and number of ways to decorate it. As the number of ways can be large, print the number of ways modulo 1000000007 (10^9+7).

Two decorations are different if,

- Their roots are different or
- There is a node u on which a different integer is written.

Input

The first line contains an integer **T (1 ≤ T ≤100)** denoting the number of test cases. First line of each test case contains a single integer **n (1 ≤ n ≤1000)**. Each of the next n-1 lines contains two integers **u v (1 ≤ u, v ≤ n & u ≠ v)** denoting that node **u** and **v** are connected by an edge..

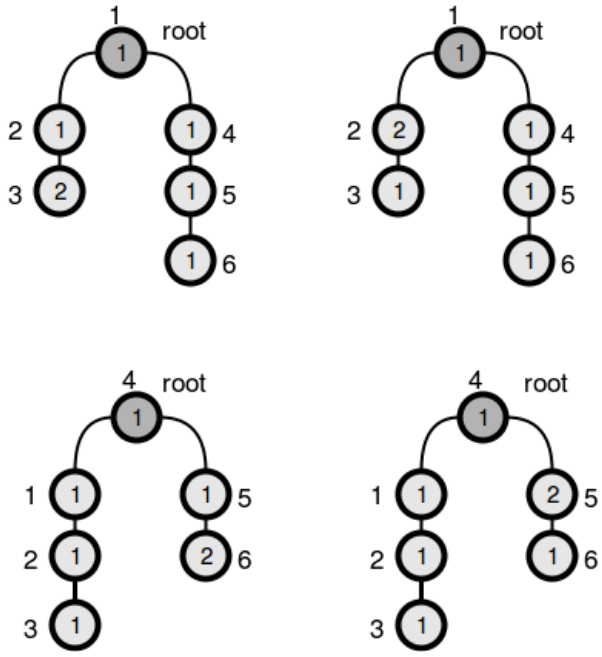
Output

For each case, print the minimum magic number and the number of ways modulo 1000000007 (10^9+7) to decorate the tree.

Sample Input	Output for Sample Input
2 6 1 2 2 3 1 4 4 5 5 6 5 1 2 1 3 3 4 3 5	Case 1: 4 4 Case 2: 3 2

Explanation of case 1:

The four ways for case 1 are shown below:



If you decorate the tree in any other way, you will get magic number greater than 4.

C Para La La Land

Input: Standard Input
Output: Standard Output

The jazz-loving people of Para La La Land are struggling with the optimal location of Seb's (a jazz gig). They want the location, that minimizes the overall travel cost of their people. As the name says, in 3D Para La La Land, people can only move in the axis-parallel direction. So, travel cost of going from point ***(x0, y0, z0)*** to ***(x1, y1, z1)*** is ***abs(x0 - x1) + abs(y0 - y1) + abs(z0 - z1)***.

In Para La La Land, people come and go frequently. So, you have to handle two kinds of query.

1. Insert query in format **i x y z**:
Means a jazz lover joins Para La La Land at coordinate x, y, z.

2. Remove query in format **r id**:
Means the person you inserted in idth insert operation leaves the Para La La Land.

After performing each query the optimal location of Seb's may change, so you have to find out the **minimum travel cost sum** of all people to meet at Seb's. For a single person, the cost is the travel cost of moving from his/ her joining point to the Seb's. You are minimizing the sum of all such costs.

Input

The first line of the input is an integer T, the number of test cases. Following lines contain T test cases. A case starts with a line containing an integers N, the number of queries. Each of following N lines represents a query. Query of format **i x y z** is an insert query and query of format **r id** is a remove query.

Constraints:
$1 \leq \mathbf{T} \leq 10$.
$0 \leq \mathbf{N} \leq 100005$.
$-1000000009 \leq \mathbf{x, y, z} \leq 1000000009$.
$1 \leq \mathbf{id} \leq$ Number of insert operations so far. And it's guaranteed to be valid.

Output

For each test case, the output contains the test case id in the first line. And each of the following N lines contains the optimal distance sum after performing the corresponding input query. Input will be such that the output value is always an integer.

Sample Input	Output for Sample Input
1 6 i 5 3 1 i 1 2 4 i 1 3 5 r 1 i 2 2 2 r 3	Case 1: 0 8 9 2 5 3

Explanation:
After 3rd insertion, you will have 3 people. One possible optimal location for Seb's is (1, 3, 4).
Minimum travel cost sum = (abs(5 - 1) + abs(3 - 3) + abs(1 - 4)) + (abs(1 - 1) + abs(2 - 3) + abs(4 - 4)) + (abs(1 - 1) + abs(3 - 3) + abs(5 - 4)) = 9.

D

Game of String

Input: Standard Input
Output: Standard Output

You already have heard the name of Captain Pirate Jack Dumb. Being chased by police he went deep into the forest and found a treasure box. He opened the box and found character set from 'a' to 'j'. There were unlimited amount of alphabets of each types. Being bored he started to play with those characters. He randomly picked N letters and put them maintaining a sequence.

Today he travelled from the starting position to the end of that sequence. He may perform some operations at his current position along his way. During his travel, he made **M** operations.

There are four types of operations:

I x y: Insert letter **y** at current position **x**.

R x y: Replace letter at current position **x** with **y**.

D x: Delete the character from current position **x**.

Q A B C: He Calculated the number of ways he could choose three positions **x, y and z** such that **x<y<z** and the $\mathbf{x^{th}}$**,** $\mathbf{y^{th}}$ and $\mathbf{z^{th}}$ letter of the sequence are **A, B** and **C** respectively, where **A**, **B** and **C** are letters between 'a' to 'j'.

Input

First line of the input starts with an integer **T** denoting the number of test cases.
Each test case starts with a nonempty string of length N and an integer **M**. The ith element of the string will denote the ith letter of the sequence.
The next M lines will contain information of the operations in the order he executed them. Each operation will be in the form given in the description.

Constrains:

$1 \leq \mathbf{T} \leq 12$
$1 \leq$ initial string length ≤ 100000
$1 \leq \mathbf{M} \leq 100000$
'a' $\leq$ **A, B, C** $\leq$ 'j'
String will never be empty and you may safely assume that all inputs are valid.

Output

For each "**Q**" operation, print desired answer on a single line. You have to follow the exact format as sample input and output.

Sample Input	Output for Sample Input
2 abc 13 Q a b c I 1 b I 3 a Q b a b Q a b c Q c c c I 4 b Q b a b R 4 a Q a b c D 6 I 6 b Q b a b aaaa 1 Q a a a	1 2 2 0 4 3 6 4

Explanation of sample test case 1:

Initial String was "abc".
After inserting 'b' at position 1, string became "babc".
After inserting 'a' at position 3, string became "baabc".
After inserting 'b' at position 4, string became "baabbc".
After replacing character at position 4 with 'a', string became "baaabc".
After deleting character at position 6, string became "baaab".
After inserting 'b' at position 6, string became "baaabb".

E Golden Light

Input: Standard Input
Output: Standard Output

Karas is a legendary gambler, famous for understanding which games can be won and which cannot. He is often referred to as "The Cowardly Sage" due to his guiding principle, "Never play a game that you are sure you cannot win."

Today Karas is battling Old Uyghur. Old Uyghur has an array **A** of **N** Golden Lights. Some of them are switched on and some are off. Old Uyghur has proposed to play a game using these **N** Golden Lights called "End of Light".

"End of Light" game is played by two players. Karas and Old Uyghur will be alternating turns. At each turn, the player can apply one of the following three moves.

1. Choose any switched on light with position **i** in the array and turn it off.
2. Choose any two switched on lights with positions **i** and **j**, where **i > j** and then turn both of them off.
3. Choose any switched on light at position **i** and any switched off light at position **j** where **i > j** and flip both of them, i.e, turn off light at **i** and turn on light at **j**.

Using these rules, all the lights will eventually turn off, hence "End of Light" will be reached. Now, whoever has no more moves to play, will lose. It is guaranteed that both players play optimally, i.e, both players will choose the move that gives him the optimal result.

Now, Old Uyghur is cunning and instead of playing just one round of match with Karas using the array **A**, he decided to play multiple games using any prefix of **A** and also change the array **A** time to time. Karas will play only if he is guaranteed to win, hence he came to you for help.

So basically, we are given the initial states of the **N** golden lights as a binary string, where **1** means that the light is switched on and **0** means that the light is switched off. Then **M** instructions containing queries and updates follow. There are two types of instructions.

1. **Q P** - Where **Q** is a character, which stands for query and **P** is an integer. For this query, you have to find whether Karas will win if Karas and Old Uyghur plays "End of Light" game using the first **P** lights of **A** from position **1** to **P** (inclusive).

   Karas will move first and if he doesn't win, print "Coward", else print the first move that will lead to victory. For printing a move, you need to print two integers **i**, **j** corresponding to the lights selected as mentioned in the rules above. If there are multiple moves that lead to victory, choose the one that has the **maximum** value of **i** (refer to the three moves of the

"End of Light" game above). If there are still multiple moves possible, choose the move that has the **minimum** value of **j** (assume that **j = 0** when Karas uses move **1**). Please see the sample I/O at the end of the description for more details.

2. **U P** - Where **U** is a character, which stands for update and **P** is an integer. When you receive this instruction, simply toggle the light at position **P**.

You can find more details down below at explanation of first sample case.

Input

Input contains multiple test cases. First line of the input contains a single integer **T** denoting number of test cases. Next follows the details of the **T** test cases.

First line of each case is a binary string representing the states of the lights in the array. Next line contains a single integer M. Then next M lines contains the instructions as above, either query ("**Q P**") or update ("**U P"**).

Constraints

T ≤ 5
N = Length of Binary String
1 ≤ N, M ≤ 100,000
1 ≤ P ≤ N

Output

For each case, print the case number (see the sample output). Then for each query of type "**Q**", print "Coward" if Karas will definitely lose if they play with the array prefix from **1** to **P** (inclusive) or print the first move that leads to victory of Karas. Details of printing the move can be found in the description.

Sample Input	Output for Sample Input
1 1001101000 5 Q 2 Q 9 Q 5 U 5 Q 5	Case 1: 1 0 7 0 Coward 4 1

Explanation

First, we have "**Q** 2**"**. This means we are playing with prefix "10". In this case, if we apply move **1** on switch of light at position 1, all lights get turned off. Hence, we print "1 0" where **i** = 1 and **j** = 0 (for move of type **1**, **j** is always 0)

Next, we have "**Q** 9". This means we are playing with prefix "100110100". In this case, if we apply move of type **1** again and switch off light at position 7, we end up with "1001100". Well, from this state, no matter what move player 2 makes, player 2 will always lose (see the next paragraph to understand why). So we make move "7 0".

For the next query, we have "**Q** 5". This means we are playing with prefix "10011". The first player can't win from here. If player 1 uses move **1** to turn off one light, then player 2 will use move **2** to turn off remaining lights. If player 1 uses move **2** to turn off two lights, then player 2 will use move **1** to turn off last light. Hence, player 1 can only use move three and he can use it to go to one of the following states: "11010" or "10110" or "11001" or "10101". Now, no matter where player 1 leads the game to, player 2 can use move of type three to go to this state: "11100". Now, from this state, player 1 cannot use move of type **3** and using any one of the other moves leads to defeat. Hence, Karas should chicken out on this game.

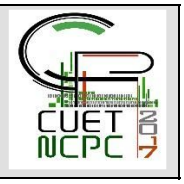
Next, we update the array **A** by flipping at position 5. Now we have the string "1001001000".

Next, we have "**Q** 5". This means we are playing with state "10010". From this state, Karas can use move of type **2** and turn off both lights at position 4 and 1. Hence, we print "4 1".

F

Finding Path

Input: Standard Input
Output: Standard Output



You will be given a rooted tree. And a lot of operations on the tree. The operations will be of three types:

1. Add a new node V with node U. Node U will be always valid, that means it belongs to the tree. Node V will be such that it doesn't belong to the tree so far. It will be a completely new node.
2. Relabel a node U with node V. Node U will be always valid, that means it belongs to the tree. Node V will be such that it doesn't belong to the tree so far. It will be a completely new node.
3. Query U, V. For this query, you need to find the path length between nodes U and V. U and V will always be valid.

The tree is unweighted, so the path length is denoted by the number of edges that needs to be traversed for going from U to V. Node 1 is the root. And it's always there at the beginning.

Input

First line, T ($0 < T < 10$), number of test cases.
Each case will start with Q ($0 < Q < 200{,}001$), then number of operations.
Each operation will be of the form C U V ($0 < C < 4$) ($0 \leq U, V \leq 1000{,}000{,}000$). The C denotes the type of operation. In case of operation type 1 and 2, U will be such that it's already in the tree, and V will be a new node. In case of operation type 3, both nodes are already in the tree.

Output

Print "Case X:" for each case, where X is the case number. Then for each type 3 query, print the path length in a separate line.

Sample Input	Output for Sample Input
1 8 1 1 5 1 1 2 1 5 6 3 2 6 1 2 7 3 7 6 2 7 9 3 9 6	Case 1: 3 4 4

G Garbled Array

Input: Standard Input
Output: Standard Output

Alice has a *sorted* array **A** of ***n*** *unique* numbers. She asked her little sister lucy to right shift the array some number of times.

For example of A=[1, 2, 3, 4, 5, 6] and it's right shifted 3 times, the array will become [4, 5, 6, 1, 2, 3].

As lucy is very young, she sometimes makes mistake while shifting and shuffling the array randomly. That results in a garbled array.

Given an array, you have to tell if the array can be produced by right shifting a sorted array some number of times. If No, print "x" without quotes. If Yes, print the minimum number of times the array was shifted.

Input

The first line contains **T (1 ≤ T ≤ 2000)**, number of test cases. Each of the test cases contains two lines. The first line contains **n (1 ≤ n ≤ 2000)**, the size of the array. The second line contains n distinct space-separated integers A_0, A_1, ... A_{n-1}. ($0 \le A_i \le 10^5$).

Output

For each test case, print the case number and the answer to the problem.

Sample Input	Output for Sample Input
3 6 4 5 6 1 2 3 6 1 2 3 6 5 4 4 4 1 2 3	Case 1: 3 Case 2: x Case 3: 1

H

A Gift of Strings

Input: Standard Input
Output: Standard Output

Alice's birthday is coming up, and her best friend Bob wants to surprise her with a gift on her special day. Bob knows that she likes nothing more than strings, which is why he wants to give her all possible unique strings he can find of length **N**, consisting of letters from the alphabet of size **M**.

Bob has hired you, one of the greatest programmers out there, to find the number of possible unique strings.

Two strings are considered the same, if after rotating them in a cyclic order and/or reversing them, they become the same.

For example, the strings **abc**, **bca**, **cab**, **cba**, **acb** and **bac** are all considered the same string.

Input

The first line of each input contains a single integer **T (1 ≤ T ≤ 200)**, which denotes the number of test cases.
The next **T** lines contain two integers, **N (1 ≤ N ≤ 10^5)** and **M (1 ≤ M ≤ 10^9)**, which denotes the length of the string, and the size of the alphabet respectively.

Output

For each test case, output the case number, followed by the number of unique strings Bob can give to Alice. Since the result can be large, print it modulo **1,000,000,007**.
See the sample input/output for more clarification.

Sample Input	Output for Sample Input
2 2 2 3 3	Case 1: 3 Case 2: 10

Explanation:

For the first case, the three unique strings are: **aa**, **ab**, and **bb**.

For the second case, the ten unique strings are: **aaa**, **aab**, **aac**, **abb**, **abc**, **acc**, **bbb**, **bbc**, **bcc**, and **ccc**.

I Repeater Again Probably

Input: Standard Input
Output: Standard Output

Hello soldier, welcome to the military base Ground Zero. The enemies are planning to attack our base again. Right now they are camping in the jungle near our camp for the night. We must destroy them before the sunrise.

Now there are **N** enemy vehicles standing in a straight line. The $\mathbf{i}^{th}$ of these vehicles is $\mathbf{a_i}$ distance away from the base ($1 \leq a_1 < a_2 < a_3 < ... < a_N \leq 10^9$; all $\mathbf{a_i}$ are distinct integers). For practical purpose, you can think that our base and all the vehicles are dots in a straight line.

Now the enemies don't know about our secret weapon the repeater. The repeater is a powerful weapon with which we can destroy the enemy vehicles. But to use the repeater, we first need to know the positions of the enemies. Unfortunately, our spy is very bad. He could not manage to acquire their exact positions. He could only manage to acquire **N** ranges each corresponding to a particular enemy but he lost the ordering. He says he is sure that for each range, there is an enemy in the range but he doesn't know which enemy it is. That is, for $\mathbf{j}^{th}$ range $[\mathbf{S_j}, \mathbf{E_j}]$, there is an enemy at one of the integer points between $\mathbf{S_j}$ to $\mathbf{E_j}$ (inclusive). Each integer point has the same probability of being the enemy's position. In other words, the distance of the $\mathbf{i}^{th}$ enemy $\mathbf{a_i}$ is an uniform random integer between some $\mathbf{j}^{th}$ range $[\mathbf{S_j}, \mathbf{E_j}]$ (**i** not necessarily equals to **j**). So if he gives us a range [2, 4], then there is an enemy with distance 2, 3 or 4 from the base; all with probability ⅓.

As the spy failed to give us the exact ordering of the enemy ranges, we don't know which range belongs to which enemy. So can you give us the ordering of the ranges so that in your ordering $\mathbf{i}^{th}$ range will belong to the $\mathbf{i}^{th}$ enemy? There can be multiple ordering as we only know the ranges without any particular order. But can you give us the ordering which is most likely going to happen? Also, can you give us the probability that our spy's data is valid? That is what is the probability that we can choose the $\mathbf{i}^{th}$ enemy's distance a_i from the $\mathbf{i}^{th}$ range in your new ordering while maintaining $a_1 < a_2 < a_3 < ... < a_N$?

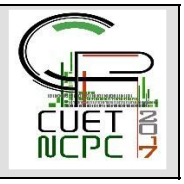
Input

First line of the input is **T**(≤500), then **T** test cases follow. First line of each test case consists of a single integer **N** ($1 \leq \mathbf{N} \leq 1000$), the number of enemy vehicles. Next **N** lines will contain two space separated positive integers $\mathbf{S_i}$ $\mathbf{E_i}$, denoting the start and end of $\mathbf{i}^{th}$ range ($1 \leq \mathbf{i} \leq \mathbf{N}$, $1 \leq \mathbf{S_i} \leq \mathbf{E_i} \leq 10^9$, $\mathbf{E_i} - \mathbf{S_i} \leq 1000$).

Output

For each test case print a line in "Case I:" format where I is the case number. In next line give us the expected ordering(ranges are numbered 1 to **N**). If there are multiple candidate orderings then output the one with the smallest first number. If the first number is same then the ordering with the second smallest number. If they are same then the third one and so on. In the next line output the probability, that this ordering is the correct ordering. For the probability, error lower than 1e-6 will be ignored. See samples for more info.

Sample Input	Output for Sample Input
3 1 2 3 2 3 4 1 2 3 2 4 6 7 1 3	Case 1: 1 1 Case 2: 2 1 1.0000 Case 3: 3 1 2 0.666666666667

J	Sniper Shot **Input:** Standard Input **Output:** Standard Output	

Consider a 2d zombieland. A sniper is sitting alertly at a point **S**. A zombie is about to run from a point **A** towards a point **B** along the connecting line-segment. The segment **AB** is parallel to x-axis. There is a circular mountain of radius **R** in the land, centered at **C**. The mountain might obstruct the sniper's vision.

You are provided with the values **Sx, Sy, Ax, Ay, Bx**, **By, Cx** and **Cy** denoting x and y coordinates of **S**, **A**, **B** and **C** respectively and radius **R** of the circular mountain.

For special security reasons, the authority needs to know the portions of **AB** guarded by the sniper. Help them by finding the portions of line-segment **AB** which are visible to the sniper.

To explain how to print the segments which are visible, let's consider two points **P** and **Q** (**P** and **Q** are distinct points) on **AB**. The points **P** and **Q** are in order of ascending distance from **A**. That means, if **Ax < Bx**, then **Ax ≤ Px < Qx ≤ Bx**, or if **Ax > Bx**, then **Ax ≥ Px > Qx ≥ Bx**. If all points between **P** and **Q** are visible, then you have to print the segment **PQ**. Furthermore, if **P** and **Q** are same points, that means only a single point is visible, then you don't need to print that **PQ** and it will be considered invisible. So basically you need to print segments that have positive length.

If there are multiple ordered visible segments you need to print, let's say, **PQ** and **P'Q'**, then they also should be printed in order of ascending distance from **A**. That means, print the segment which is nearest (from **A**) first.

Input

An integer **T ≤ 100**, denoting the number of test cases. Next **T** lines will contain 9 space separated integers denoting **Sx, Sy, Ax, Ay, Bx, By, Cx, Cy** and **R** respectively.

Constraint:

- Every value will be positive and less than 1000.
- The line-segment AB will not intersect the circle (it may touch), also the point **S** will neither be inside nor on the circle.
- **A**, **B** and **S** will be distinct points.

Output

Print the case number in a line, then print the coordinates of the endpoints of each visible segment in a single line, rounded to 3 decimal places. If there are no valid visible segments, then print "Invisible" without quotes.

Sample Input	Output for Sample Input
4 30 45 30 30 50 30 32 35 3 30 45 30 30 50 30 32 35 1 50 45 30 30 50 30 48 35 2 50 45 30 30 50 30 48 35 3	Case 1: 38.117 30.000 50.000 30.000 Case 2: 30.000 30.000 31.493 30.000 34.568 30.000 50.000 30.000 Case 3: 30.000 30.000 43.750 30.000 Case 4: 30.000 30.000 41.883 30.000

K

Implications

Input: Standard Input
Output: Standard Output

In the study of discrete mathematics, the material conditional (also known as material implication, or simply implication) is a logical connective (or a binary operator) that is often symbolized by a forward arrow "→". An implication is used to form statements of the form $p \rightarrow q$ which is read as "if p then q" or "p only if q" or "if p is true, then q is also true". The statement $p \rightarrow q$ is false only when p is true and q is false.

As one might have expected, more than one events can imply one single event and a single event can imply more than one events at the same time. For this problem, let's change the definition of implications a bit. For all implications of the form $p \rightarrow qi$, if p is true, each of qi must also be true. On the other hand, for all implications of the form $pi \rightarrow q$, if q is true, at least one of pi has to be true.

So, if there are two implications $p \rightarrow q$ and $w \rightarrow q$, while given that q is true, either of p or w can be true. Hence, we cannot tell exactly which one is true. On the other hand, for implications $p \rightarrow q$ and $q \rightarrow w$, if q is true, we know that only p can cause q to be true, hence we can conclude p is true without any confusion. In this case, w is also true by definition, since q is true.

As we can see in the above examples, several implications can form chains of direct or indirect implications which can be used to determine truth values of other events from a set of given events which are already known to be true.

For this problem, you will be given a list of events which are known to be true and several implications formed by those events. Your task is to find all the events which certainly can be concluded as true.

Input

First line of the input will contain the number of test cases T ($1 < T < 25$). Then T cases follow.

For each case, the first line will contain three integers E, N, M. Here, E ($1 \leq E \leq 1000$) is the total number of events, N ($1 \leq N \leq E$) is the number of events that are known to be true. And M ($1 \leq M \leq 100000$) is the number of implications. The next line will contain a list of N integers p_i ($1 \leq p_i \leq E$) separated by single space, which are the events that are known to be true. Each of the following M lines will contain a pair of integers p, q ($1 \leq p, q \leq E$) which will denote the relation $p \rightarrow q$.

Events are numbered from 1 to E and there will be no event outside of this domain. The implications will never be circular.

Output

For each test case, on the first line, print the case number starting from 1, and the number of different events that are certainly true, including the given ones. On the second line, print the list of events separated by single space in ascending order. Check sample input and output for details.

Sample Input	Output for Sample Input
3 3 1 2 2 1 2 2 3 3 1 2 3 1 3 2 3 4 1 4 4 1 2 1 3 2 4 3 4	Case 1: 3 1 2 3 Case 2: 1 3 Case 3: 4 1 2 3 4

Explanation:

For the third case, it is known that event 4 is true. This can be caused by any of the events 2 and 3. Now, for at least one of the events 2 or 3 to be true, event 1 has to be true. Because only event 1 implies 2 and 3. This, by definition, makes both events 2 and 3 to be true. Hence we can conclude that all four events are true.