A Recurrence

Input: Standard Input
Output: Standard Output

Consider a tuple P_1, P_2, P_3, …, P_n. Now consider the following recurrence function.

- $F(P_1, P_2, P_3, \dots, P_n) = 0$ if any of the P_i is negative or the tuple P is not sorted in non-increasing order.
- $F(P_1, P_2, P_3, \dots, P_n) = 1$ if all of the P_is is zero.
- $F(P_1, P_2, P_3, \dots, P_n) = F(P_1 - 1, P_2, P_3, \dots, P_n) + F(P_1, P_2 - 1, P_3, \dots, P_n) + F(P_1, P_2, P_3 - 1, \dots, P_n) + \cdots . + F(P_1, P_2, P_3, \dots, P_n - 1)\ otherwise$

For example if n is 4 then the value
F(4,3,2,-1) is 0 because the last parameter is negative.
F(4,3,2,5) is 0 because the tuple is not sorted from the largest to smallest.
F(3,3,2,1) = F(3,3,2,1)+F(4,2,2,1)+F(4,3,1,1)+F(4,3,2,0)
F(1, 1, 0, 0) = F(0,1,0,0)+F(1,0,0,0)+F(1,1,-1,0)+F(1,1,0,-1) = 2
Given the tuple P your task is to calculate the value of $F(P_1, P_2, P_3, \dots, P_n)$. The result can be very big so output the result mod 1,000,000,009 (this is a prime number).

Input

Input starts with an integer ***T* (≤ 50)**, denoting the number of test cases.

Each test case consists of two lines. First line contains **n**. Second line contains **n** integers separated by a single space. These are the tuple **P**. n is between 1 and 1000 inclusive. Each of the numbers in tuple P is between 1 and 1000 inclusive. P will be sorted in non-increasing order.

Output

For each test case output contains a line in the format Case x: R where x is the case number (starting from 1) and R is the value of $F(P_1, P_2, P_3, \dots, P_n)\ mod\ 1{,}000{,}000{,}009$.

Sample Input	Output for Sample Input
8 3 7 5 4 6 7 7 5 3 2 1 2 4 2 3 7 4 4 4 8 7 5 5 5 7 7 6 5 5 2 8 7 3 6 3 1	Case 1: 100100 Case 2: 398009117 Case 3: 9 Case 4: 25025 Case 5: 923714728 Case 6: 311516464 Case 7: 1430 Case 8: 315

B Emoogle Grid

Input: Standard Input
Output: Standard Output

You have to color an **M x N (1 ≤ M, N ≤ 10^8)** two dimensional grid. You will be provided **K (2 ≤ K ≤ 10^8)** different colors to do so. You will also be provided a list of **B (0 ≤ B ≤ 500)** blocked cells of this grid. You cannot color those blocked cells. A cell can be described as **(x, y)**, which points to the **y^{th}** cell from the left of the **x^{th}** row from the top.

While coloring the grid, you have to follow these rules –

1. You have to color each cell which is not blocked.
2. You cannot color a blocked cell.
3. You can choose exactly one color from **K** given colors to color a cell.
4. No two vertically adjacent cells can have the same color, i.e. cell **(x, y)** and cell **(x + 1, y)** cannot contain the same color.

Now the great problem setter smiled with emotion and thought that he would ask the contestants to find how many ways the board can be colored. Since the number can be very large and he doesn't want the contestants to be in trouble dealing with big integers; he decided to ask them to find the result modulo **100,000,007**. So he prepared the judge data for the problem using a random generator and saved this problem for a future contest as a giveaway (easiest) problem.

But unfortunately he got married and forgot the problem completely. After some days he rediscovered his problem and became very excited. But after a while, he saw that, in the judge data, he forgot to add the integer which supposed to be the 'number of rows'. He didn't find the input generator and his codes, but luckily he has the input file and the correct answer file. So, he asks your help to regenerate the data. Yes, you are given the input file which contains all the information except the 'number of rows' and the answer file; you have to find the number of rows he might have used for this problem.

Input

Input starts with an integer **T (≤ 150)**, denoting the number of test cases.

Each test case starts with a line containing four integers **N, K, B** and **R (0 ≤ R < 100000007)** which denotes the result for this case. Each of the next **B** lines will contains two integers **x** and **y (1 ≤ x ≤ M, 1 ≤ y ≤ N)**, denoting the row and column number of a blocked cell. All the cells will be distinct.

Output

For each case, print the case number and the minimum possible value of **M**. You can assume that solution exists for each case.

Sample Input

```
4
3 3 0 1728
4 4 2 186624
3 1
3 3
2 5 2 20
1 2
2 2
2 3 0 989323
```

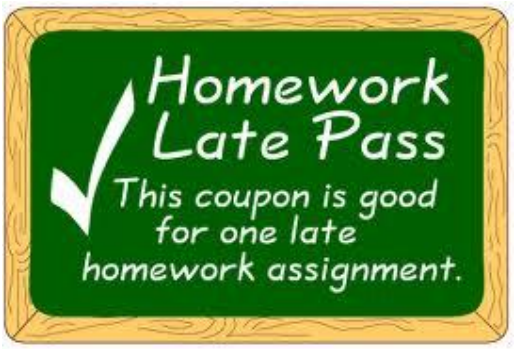
Output for Sample Input

```
Case 1: 3
Case 2: 3
Case 3: 2
Case 4: 20
```

C Do Your Own Homework!

Input: Standard Input
Output: Standard Output

These days Soha is so busy that he doesn't have time to do his own homework. But this is not a big problem since he has got many friends who are willing to help him out. One of his friend's name is Sparrow. Whenever Soha is assigned any homework, he turns to Sparrow for her help.



Sparrow has given a list of subjects that she is comfortable with along with the number of days it will take her to complete an assignment for each subject. Soha has got only **D** days to complete his next assignment. However, the professor of this subject is a little flexible and allows late submissions up to 5 days. That means he will not accept any submission that is after **D+5** days from now. Will Sparrow be able to do it for Soha this time?

Input

First line of input is a positive integer **T (T≤100)** that determines the number of test cases. Each case starts with a line containing an integer **N** that represents the number of subjects Sparrow is comfortable with. Each of the next **N** lines contain the name of a subject followed by the number of days it will take Sparrow to complete an assignment of that subject. All these subject names will be distinct. The next line contains an integer **D**. The meaning of **D** is described above. The following line contains the name of the subject whose homework is due. All the subjects' names consist of lowercase letters and the length of each is at least 1 and at most 20. All the integer inputs are positive in the range **[1,100]**.

Output

For each case, first output the case number first starting from 1. If Sparrow doesn't take more than **D** days to complete the assignment, output "**Yesss**"; if she takes more than **D** days but not more than **D+5**, output "**Late**"; if she takes more than **D+5** days or if she isn't comfortable with the subject, output "**Do your own homework!**". Quotes are for clarity only and don't need to be part of the output. Look at the samples for more details. Be careful about the spelling.

Sample Input	Output for Sample Input
3 3 compiler 4 cplusplus 1 java 8 5 compiler 2 algorithm 3 math 9 4 math 2 java 8 ai 3 6 calculus	Case 1: Yesss Case 2: Late Case 3: Do your own homework!

D Traveler of Gridland

Input: Standard Input
Output: Standard Output

Girdland is a square shaped country of size **2000000000 × 2000000000** units. We will identify any point in this land using Cartesian coordinate system. Using this system the coordinate of the center of Gridland is **(0, 0)**, the coordinate of the lower-left corner is **(-1000000000, -1000000000)** and upper-right corner is **(1000000000, 1000000000)**. So the x-axis is a horizontal line which divides the land into two equal rectangles and y-axis is a vertical line which divides the land into two equal rectangles. There are roads in the country which goes through the grid lines only i.e. along **x = -1000000000, -999999999, … , -1, 0, 1, …, 999999999, 1000000000** and **y = -1000000000, -999999999, … , -1, 0, 1, …, 999999999, 1000000000**. So all roads are either a horizontal line or a vertical line having integer distance from axis.

Travelers like this Gridland because of the simplicity of its road network. Each of the roads is so straight and axis parallel.

From any position **(a, b)**, in unit time any traveler can move to

i. **(a + 1, b)**
ii. **(a – 1, b)**
iii. **(a, b + 1)**
iv. **(a, b – 1)**

A traveler cannot move outside the Gridland. She also cannot move to any position which is occupied by a monster. There may be some monsters in Gridland. There are three types of monsters

i. Point monster
ii. Line monster
iii. Rectangle monster

A point monster can occupy only a single point, a line monster can occupy a straight line, and a rectangle monster can occupy a rectangular region.

The position of the point monster can be specified by a pair of integers **(u, v)**, which indicates that the monster occupies the coordinate position **(u, v)**.

The position of a line monster can be specified by two pair of integers **(u1, v1)** and **(u2, v2)**. **(u1, v1)** is the one end of the line monster and **(u2, v2)** is the other end of the monster. It is guaranteed that this line will always be axis parallel. The monster occupies the whole line region inclusively.

The position of rectangle monster can be specified by two pairs of integers **(u1, v1)** and **(u2, v2)**. **(u1, v1)** is the coordinate of the lower left corner of monster and **(u2, v2)** is the upper right corner of the monster. It is guaranteed that the edges of monster will always be axis parallel. The monster occupies the whole rectangular region inclusively.

Initially a traveler is in a position of coordinate **(sourceX, sourceY)** in Gridland. She needs to reach the position of coordinate **(destinationX, destinationY)**. You have to calculate the minimum unit time required for her to reach the destination. If it is not possible to reach the destination, you have to output **"Impossible"** (quotes for clarity).

Input

Input starts with an integer **T (≤ 100)**, denoting the number of test cases.

Each test case starts with four integer **sourceX**, **sourceY**, **destinationX**, **destinationY (-1000000000 ≤ sourceX, sourceY, destinationX, destinationY ≤ 1000000000)**. Next line contains three integers **M**, **N** and **O**, the number of point monsters, the number of line monsters and the number of rectangle monsters. Total number of monsters can be at most **100**, i.e. **(0 ≤ M + N + O ≤ 100)**. Each of the next M lines describe a point monster by two integers **u** and **v (-1000000000 ≤ u, v ≤ 1000000000)**. Each of the next N lines will describes a line monster by four integers **u1**, **v1**, **u2** and **v2 (-1000000000 ≤ u1, v1, u2, v2 ≤ 1000000000)**. Each of the next **O** lines will describes a rectangle monster by four integers **u1**, **v1**, **u2** and **v2 (-1000000000 ≤ u1, v1, u2, v2 ≤ 1000000000)**. Note that one monster can overlap with another. You can safely assume that source and destination is not occupied by any monster.

Output

For each test case, output a single line in the format **"Case C: N"**, where **C** will be replaced by the case number and **N** will be replaced by the shortest path distance from source to destination. If it is not possible to reach source to destination replace **"N"** by **"Impossible"** without quotes.

Sample Input	Output for Sample Input
1 1 2 4 3 1 1 1 4 6 4 1 7 1 2 1 3 5	Case 1: 14

E Hybrid Salientia

Input: Standard Input
Output: Standard Output

It's believed that frogs jump due to lack of natural physical defense against predators. However, there are some types of frogs that do not leap. In this problem, we will consider a hybrid version of a frog that can both leap and walk.

Consider a magical creek with **N** stones. The shape of each stone is either a circle or a square. Our frog is currently standing on stone 1 and it is going to make (**N-1)** leaps so that it can land on every stone. It is believed that after making **N-1** jumps, the frog will grow wings and fly away. After every jump, it loses 10% of its 'leaping energy'. That means in the $\mathbf{K}^{th}$ leap it can jump to a maximum distance of $L*0.9^{k-1}$, where **L** is the initial *maximum jump distance*. The frog, however, can walk from any point to any other point within a stone without loss of any energy.

In this problem, you have to find the minimum value of **L** that will enable the frog to visit all the stones starting from stone 1. Obviously, the visiting order of the stones will be such that the value of **L** is minimized. When calculating the distances, assume the frog is a point and the stones are circles and squares on a 2D Cartesian coordinate.

Input

The first line of input is an integer **T (≤ 200)** that indicates the number of test cases. Each case starts with a line containing an integer **N (2 ≤ N ≤ 15)** that represents the number of stones. The next **N** lines contain the descriptions of the stones starting from stone **1**. Each stone will be given in the format **type X Y R. type** can be **'*C*'** or **'*S*'** and represents circle and square respectively. If **type** is equal to **'*C*'**, then (**X**, **Y**) will give you the center of the circle and **R** will give you the radius. If **type** is **'*S*'**, then (**X**, **Y**) will give you the lower left corner of the square and **R** will give the length of the sides. The sides of the squares are axis parallel. **0 ≤ X, Y ≤ 1000000, 0 < R ≤ 1000** and stones will be non-overlapping.

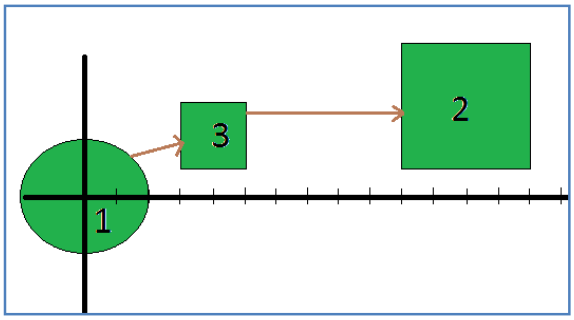
Output

For each case, output the minimum value of **L**. Errors less than 10^{-6} will be ignored.

Sample Input	Output for Sample Input
2 2 C 0 0 5 C 10 0 2 3 C 0 0 2 S 10 1 4 S 3 1 2	3.000000 5.555556

Note

For the second sample we have the picture on the right. Initial value of L is 5.555556. First, the frog makes a leap to stone 3. It loses 10% of energy and that means the next leap distance can be at most 5.555556 * 0.9 = 5.000000. Since the shortest distance between stone 3 and stone 2 is 5.000000, the next leap will enable the frog to land safely on stone 2.



F 0 s, 1 s and ? Marks

Input: Standard Input
Output: Standard Output

Given a string consisting of **0**, **1** and **?** only, change all the **?** to **0/1,** so that the size of the largest group is minimized. A group is a substring that contains either all zeros or all ones.
Consider the following example:

0 1 1 ? 0 1 0 ? ? ?

We can replace the question marks (?) to get

0 1 1 0 0 1 0 1 0 0

The groups are (0) (1 1) (0 0) (1) (0) (1) (0 0) and the corresponding sizes are 1, 2, 2, 1, 1, 1, 2. That means the above replacement would give us a maximum group size of 2. In fact, of all the 2^4 possible replacements, we won't get any maximum group size that is smaller than 2.

Input

The first line of input is an integer **T (T ≤ 5000)** that indicates the number of test cases. Each case is a line consisting of a string that contains **'0'**, **'1'**, and **'?'** only. The length of the string will be in the range **[1,1000]**.

Output

For each case, output the case number first followed by the size of the minimized largest group.

Sample Input	Output for Sample Input
4 011?010??? ??? 000111 00000000000000	Case 1: 2 Case 2: 1 Case 3: 3 Case 4: 14

G Save the Princess

Input: Standard Input
Output: Standard Output

The news has just arrived: The princess has been kidnapped by the monster. The prince became worried. But the chief scientist informed that the princess has a tracking device hidden in her locket and from that it is known that the princess is kept in a tower in a very big lake at Bermuda Triangle. The prince went out to save his princess and reached the lake. At the area it is not safe to fly so the prince needs to use a boat to reach the tower. The only problem is the big circular rocks in the lake which the prince cannot pass through. There are several of them in the lake. The prince has a satellite image of the area where the rocks are marked. The prince wondered what the shortest distance is between his current position and the tower. Can you help him?

Input

The first line of input will be the number of test cases **T (T < 51)**. Then T cases will follow from the next line. Each case starts with 4 integers **xp, yp, xt, yt**. Here **(xp, yp)** is the position of the prince and **(xt, yt)** is the position of the tower. The next line will contain an integer **n**, number of rocks in the lake. Each of the following **n** lines will contain 3 integers $\mathbf{x_i}$, $\mathbf{y_i}$, $\mathbf{r_i}$. $\mathbf{(x_i, y_i)}$ is the center and $\mathbf{r_i}$ is the radius of the i-th rock. The positions **(xp, yp)** and **(xt, yt)** will be different and both have positive distance from every rock.

Output

For each case first output one line in the format "Case **k**: **d**", where **k** Is the case number starting from 1 and **d** is the minimum distance the prince has to cover to reach the tower. See sample input and output for details.

Note

- All positions are given in Cartesian coordinates system and all the distances are Euclidian distances.
- You may consider prince, his boat and the tower each as a point object.
- The rocks will not overlap, but may touch each other. In this case, the prince cannot go through the touching point.
- **n** will be between 0 and 50 inclusive.
- All the coordinate will have an absolute value less than 1001. All the radii will be positive and less than 1001.
- It is confirmed that the prince can reach the tower.
- The output will be considered correct, if it has an absolute error less than 1e-5.

Sample Input	Output for Sample Input
2 -3 2 2 -2 2 0 0 2 0 -5 3 -3 0 4 0 2 0 5 4 0 -3 3	Case 1: 8.14159265 Case 2: 7.00000000

H Permutation Transformer

Input: Standard Input
Output: Standard Output

Write a program to transform the permutation 1, 2, 3, ..., n according to m instructions. Each instruction (a, b) means to take out the subsequence from the a-th to the b-th element, reverse it, then append it to the end.

Input

There is only one case for this problem. The first line contains two integers **n** and **m (1 ≤ n,m ≤ 100,000)**. Each of the next **m** lines contains an instruction consisting of two integers **a** and **b (1 ≤ a ≤ b ≤ n)**.

Output

Print n lines, one for each integer, the final permutation.

Sample Input

```
10 2
2 5
4 8
```

Output for Sample Input

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

Explanation

Instruction (2,5): Take out the subsequence {2,3,4,5}, reverse it to {5,4,3,2}, append it to the remaining permutation {1,6,7,8,9,10}

Instruction (4,8): The subsequence from the 4-th to the 8-th element of {1,6,7,8,9,10,5,4,3,2} is {8,9,10,5,4}. Take it out, reverse it, and you'll get the sample output.

Warning: Don't use ***cin***, ***cout*** for this problem, use faster i/o methods e.g ***scanf***, ***printf***.

I Roundabout

Input: Standard Input
Output: Standard Output

Roundabout is a game show where two players compete against each other. There are 16 spheres arranged in four rows and four columns. Initially the spheres are all colored in white. As game progresses, these white spheres will be converted to one of the two colors of red and blue. Here red color is said to be the favorable color of first player and blue that of the second player. The game proceeds as follows:

i. Each player is asked a question in turn.
ii. If the first player answers a question correctly, he is able to turn one of the white spheres to red. Alternatively, the second player will be able to convert a white sphere to blue if he answers a question correctly.
iii. If there is no white sphere left, a player can choose to convert any of the spheres colored in opposing players color.
iv. If a player is not able to turn the color of any sphere because all the spheres are already turned into his favorable color, then the player is awarded a special score (see description below).

The scoring of the game is as follows:

1. Whenever a player converts a sphere into his favorable color, all the contiguous horizontal, vertical and diagonal matches with 3 or more favorable color containing the turned sphere will contribute to score for that turn according the following rule:
    a. All the matches containing 3 contiguous sphere of favorable color containing the turned sphere will contribute 3 points each.
    b. All the matches containing 4 contiguous sphere of favorable color containing the turned sphere will contribute 4 points each.
    c. A 4 contiguous match may also fulfill the requirement of one or more 3 contiguous match, in that case, all should be considered separately.
2. If all the spheres are already turned into the favorable color of the player, then 50 points will be added to the score for that turn.

When selecting a sphere to turn, the players use the following strategy:

1. They will turn a sphere that will earn them maximum point involving that turn, as described earlier.
2. If there is more than one such sphere, they will choose the sphere with lowest row value.
3. If there is more than one sphere to choose in (ii), they will choose the sphere with the lowest column value.

In this problem, you will be simulating different sessions of the game.

Input

The first line of input contains a positive integer **T (≤ 1000)**, denoting the number of test cases. This line will be followed by **T** lines, where each line describes one test case. It will consist of strings whose maximum length will be **50**. These strings will consist of the characters '**0**' and '**1**' only. Here **1** implies a correct response and **0** implies an incorrect response. Starting with the first character, every character in the odd position corresponds to a response from the first player. Similarly, starting with the second character, every character in the even position corresponds to a response from the second player.

Output

For each case of input, there will be one line of input. It will first contain the case number followed by two integers. The first integer corresponds to the total score of the first player and the second integer corresponds to that of the second player. Look at sample output for exact formatting.

Sample Input	Output for Sample Input
4 10101010 011011001111 11111111 010111	Case 1: 10 0 Case 2: 0 3 Case 3: 3 3 Case 4: 0 0

Note

For the first case, only the first player gets the option to turn. In first two turns he converts first two spheres from the first row but gets no point. In 3rd turn he converts the 3rd ball from the first row and gets 3 point as it is a part of 3 contiguous horizontal spheres of his favorable color. In the 4th turn he converts the 4th ball in the first row and gets 7 points because the turned ball is a part of 4 contiguous horizontal spheres starting from 1, (4 points) and also a part of 3 contiguous horizontal spheres starting from 2 (3 points).

J

Unlock the Winning Pot

Input: Standard Input
Output: Standard Output

You are about to finish your favorite game … (put the name of your favorite game here). And now you are on the last level. You have almost finished it but there is one hard quest that you want to avoid. So again you are going for shortcut. The shortcut is again to solve a puzzle. The puzzle is given as an **m x n** grid where each cell colored in red, blue or green. You are also given a target grid. You have to change given grid the to the target grid.

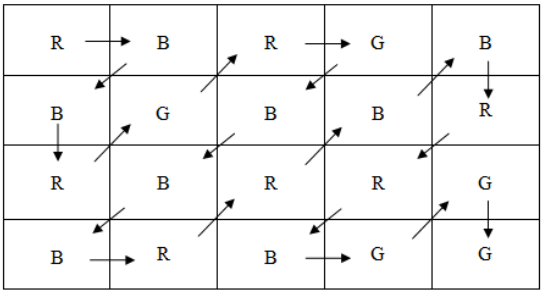
The only allowed operation is pressing a switch. After some trial and error you figured out how the switch works. It rearranges the element by reading them by one diagonal after another. And put them back row wise. See he tables below for details.

Then each cell is recolored using following rule
A cell (i,j) is recolored only if (i+j) is even. (The cells are numbered (1, 1) for the left top point and (m, n) for the right bottom point). However, no cells in bottommost row or rightmost column are recolored, even if (i+j) is even.

For recoloring 3 cells are considered. The cell right to it, the cell below it and the cell itself.
The recoloring rule is color(current, below, right)=f(current, f(below, right)). Where the f function is defined by following table.

	B	R	G	B
A				
R		R	G	B
G		G	B	R
B		B	R	G

For example the following grid will be read as RBBRGRGBBBRRBBRRBGGG and it will be transformed into the grid shown in below.



Initial grid

R	B	B	R	G
R	G	B	B	B
R	R	B	B	R
R	B	G	G	G

After rearrangement

B	B	G	R	G
R	R	B	R	B
R	R	B	B	R
R	B	G	G	G

After recoloring

Now you are wondering given the initial configuration how long it will take to solve the puzzle or whether it is impossible to solve in fewer than 2^{24} steps.

Input

Input starts with an integer **T ≤ 100**. **T** test cases follow.

Each test case starts with two positive integers **m, n (3 ≤ m, n and m * n ≤ 25)** .Then follows **m** lines, each containing **n** space separated characters, representing the initial grid. Then follows another **m** lines, each containing **n** space separated characters, representing the target grid. Both grids are followed by an empty line. See Sample IO for detail.

Output

For each case print one line containing number of steps needed to reach the solution or -1 if solution can not be reached in less than 2^{24} steps.

Sample Input

```
3
4 5
R B R G B
B G B B R
R B R R G
B R B G G

B B G R G
R R B R B
R R B B R
R B G G G

3 3
R R R
R R R
R R R

R R R
R R R
R R B

3 3
R R R
R R R
R R R

R R R
R R R
R R R
```

Output for Sample Input

```
Case 1: 1
Case 2: -1
Case 3: 0
```