A. Rudolf and the Ticket

1 second, 256 megabytes

Rudolf is going to visit Bernard, and he decided to take the metro to get to him. The ticket can be purchased at a machine that accepts exactly two coins, the sum of which does not exceed k.

Rudolf has two pockets with coins. In the left pocket, there are n coins with denominations $b_1, b_2, \ldots, b_n$. In the right pocket, there are m coins with denominations $c_1, c_2, \ldots, c_m$. He wants to choose exactly one coin from the left pocket and exactly one coin from the right pocket (two coins in total).

Help Rudolf determine how many ways there are to select indices f and s such that $b_f + c_s \leq k$.

Input

The first line contains an integer t ($1 \leq t \leq 100$) — the number of test cases. Then follows the description of each test case.

The first line of each test case contains three natural numbers n, m, and k ($1 \leq n, m \leq 100, 1 \leq k \leq 2000$) — the number of coins in the left and right pockets, and the maximum sum of two coins for the ticket payment at the counter, respectively.

The second line of each test case contains n integers b_i ($1 \leq b_i \leq 1000$) — the denominations of coins in the left pocket.

The third line of each test case contains m integers c_i ($1 \leq c_i \leq 1000$) — the denominations of coins in the right pocket.

Output

For each testcase, output a single integer — the number of ways Rudolf can select two coins, taking one from each pocket, so that the sum of the coins does not exceed k.

input

```
4
4 4 8
1 5 10 14
2 1 8 1
2 3 4
4 8
1 2 3
4 2 7
1 1 1 1
2 7
3 4 2000
1 1 1
1 1 1 1
```

output

```
6
0
4
12
```

Note that the pairs indicate the **indices** of the coins in the array, not their denominations.

In the first test case, Rudolf can choose the following pairs of coins: $[1, 1], [1, 2], [1, 4], [2, 1], [2, 2], [2, 4]$.

In the second test case, Rudolf cannot choose one coin from each pocket in any way, as the sum of any two elements from the first and second arrays will exceed the value of $k = 4$.

In the third test case, Rudolf can choose: $[1, 1], [2, 1], [3, 1], [4, 1]$.

In the fourth test case, Rudolf can choose any coin from the left pocket and any coin from the right pocket.

B. Rudolf and 121

2 seconds, 256 megabytes

Rudolf has an array a of n integers, the elements are numbered from 1 to n.

In one operation, he can choose an index i ($2 \le i \le n - 1$) and assign:

- $a_{i-1} = a_{i-1} - 1$
- $a_i = a_i - 2$
- $a_{i+1} = a_{i+1} - 1$

Rudolf can apply this operation any number of times. Any index i can be used zero or more times.

Can he make all the elements of the array equal to zero using this operation?

Input

The first line of the input contains a single integer t ($1 \le t \le 10^4$) — the number of test cases in the test.

The first line of each case contains a single integer n ($3 \le n \le 2 \cdot 10^5$) — the number of elements in the array.

The second line of each case contains n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_j \le 10^9$) — the elements of the array.

It is guaranteed that the sum of the values of n over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, output "`YES`" if it is possible to make all the elements of the array zero using the described operations. Otherwise, output "`NO`".

You can output each letter in any case (lowercase or uppercase). For example, the strings "`yEs`", "`yes`", "`Yes`", and "`YES`" will be accepted as a positive answer.

input

```
7
5
1 3 5 5 2
5
2 4 4 5 1
5
0 1 3 3 1
6
5 6 0 2 3 0
4
1 2 7 2
3
7 1 0
4
1 1 1 1
```

output

```
YES
NO
YES
NO
NO
NO
NO
```

In the first example, the original array is $[1, 3, 5, 5, 2]$, to make all its elements zero, Rudolf can act as follows:

- apply the operation at $i = 4$ and get the array $[1, 3, 4, 3, 1]$;
- apply the operation at $i = 3$ and get the array $[1, 2, 2, 2, 1]$;
- apply the operation at $i = 2$ and get the array $[0, 0, 1, 2, 1]$;
- apply the operation at $i = 4$ and get the array $[0, 0, 0, 0, 0]$.

C. Rudolf and the Ugly String

2 seconds, 256 megabytes

Rudolf has a string s of length n. Rudolf considers the string s to be ugly if it contains the substring† "pie" or the substring "map", otherwise the string s will be considered beautiful.

For example, "ppiee", "mmap", "dfpiefghmap" are ugly strings, while "mathp", "ppiiee" are beautiful strings.

Rudolf wants to shorten the string s by removing some characters to make it beautiful.

The main character doesn't like to strain, so he asks you to make the string beautiful by removing the minimum number of characters. He can remove characters from **any** positions in the string (not just from the beginning or end of the string).

† String a is a substring of b if there exists a **consecutive** segment of characters in string b equal to a.

Input

The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases. The descriptions of the test cases follow.

The first line of each test case contains a single integer n ($1 \le n \le 10^6$) — the length of the string s.

The next line of each test case contains the string s of length n. The string s consists of lowercase Latin letters.

The sum of n over all test cases does not exceed 10^6.

Output

For each test case, output a single integer — the minimum number of characters that need to be deleted to make the string s beautiful. If the string is initially beautiful, then output 0.

input

```
6
9
mmapnapie
9
azabazapi
8
mappppie
18
mapmapmapmapmapmap
1
p
11
pppiepieeee
```

output

```
2
0
2
6
0
2
```

In the first test case, for example, you can delete the 4th and 9th characters to make the string beautiful.

In the second test case, the string is already beautiful.

D. Rudolf and the Ball Game

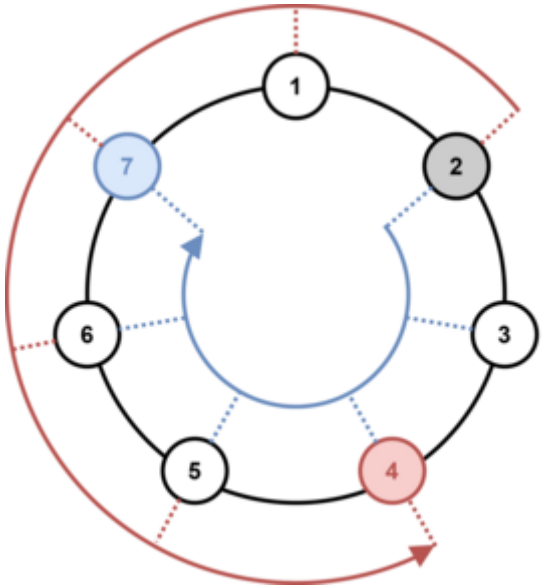
2 seconds, 256 megabytes

Rudolf and Bernard decided to play a game with their friends. n people stand in a circle and start throwing a ball to each other. They are numbered from 1 to n in the clockwise order.

Let's call a transition a movement of the ball from one player to his neighbor. The transition can be made clockwise or counterclockwise.

Let's call the clockwise (counterclockwise) distance from player y_1 to player y_2 the number of transitions clockwise (counterclockwise) that need to be made to move from player y_1 to player y_2. For example, if $n = 7$ then the clockwise distance from 2 to 5 is 3, and the counterclockwise distance from 2 to 5 is 4.

Initially, the ball is with the player number x (players are numbered clockwise). On the i-th move the person with the ball throws it at a distance of r_i ($1 \le r_i \le n - 1$) clockwise or counterclockwise. For example, if there are 7 players, and the 2nd player, after receiving the ball, throws it a distance of 5, then the ball will be caught by either the 7th player (throwing clockwise) or the 4th player (throwing counterclockwise). An illustration of this example is shown below.



The game was interrupted after m throws due to unexpected rain. When the rain stopped, the guys gathered again to continue. However, no one could remember who had the ball. As it turned out, Bernard remembered the distances for each of the throws and the direction for **some** of the throws (clockwise or counterclockwise).

Rudolf asks you to help him and based on the information from Bernard, calculate the numbers of the players who could have the ball after m throws.

Input

The first line of the input contains a single integer t ($1 \le t \le 10^4$) — the number of test cases. Then follow the descriptions of the test cases.

The first line of each test case contains three integers n, m, x ($2 \le n \le 1000, 1 \le m \le 1000, 1 \le x \le n$) — the number of players, the number of throws made, and the number of the player who threw the ball first, respectively.

The next m lines contain information about each throw in order. Each of them contains an integer r_i ($1 \le r_i \le n - 1$) — the distance at which the i-th throw was made, and a symbol c_i, equal to '`0`', '`1`', or '`?`':

- if c_i='`0`', then the i-th throw was made clockwise,
- if c_i='`1`', then the i-th throw was made counterclockwise,
- if c_i='`?`', then Bernard does not remember the direction and the i-th throw could have been made either clockwise or counterclockwise.

It is guaranteed that the sum $n \cdot m$ (n multiplied by m) over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, output two lines.

In the first line, output the number of players k ($1 \le k \le n$) who could have the ball at the end of the game.

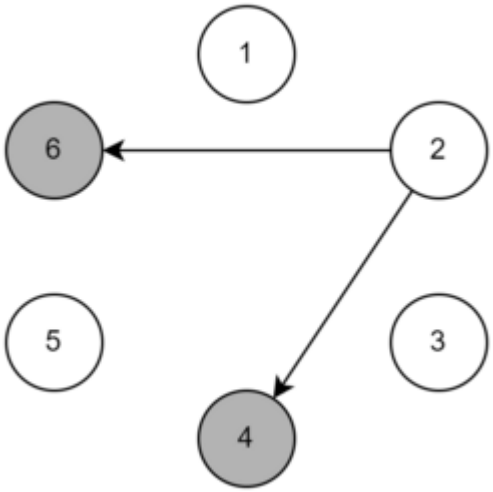
In the next line, output k numbers b_i ($1 \le b_i \le n$) — the numbers of the players in increasing order. All numbers must be different.

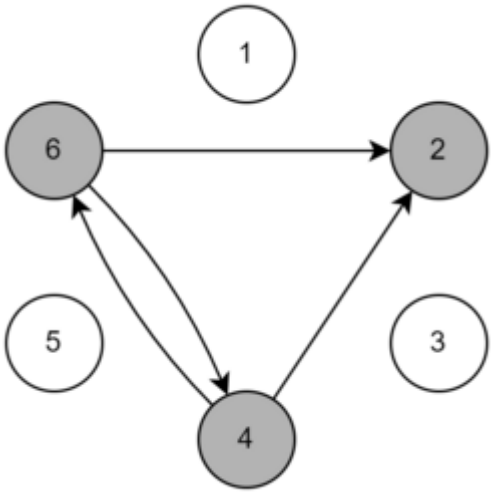
input

```
5
6 3 2
2 ?
2 ?
2 ?
12 1 2
3 1
10 7 4
2 ?
9 1
4 ?
7 0
2 0
8 1
5 ?
5 3 1
4 0
4 ?
1 ?
4 1 1
2 ?
```

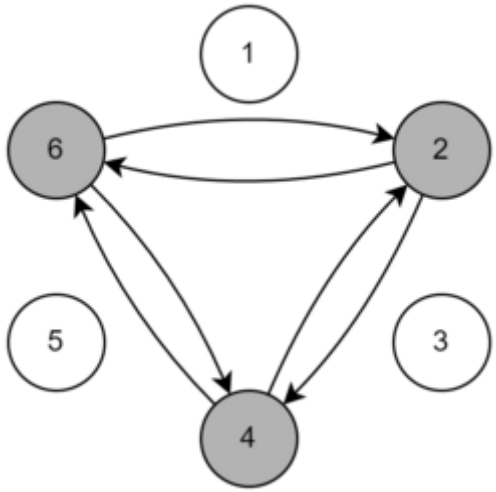
output

```
3
2 4 6
1
11
4
3 5 7 9
3
2 3 5
1
3
```





Below is an illustration of three throws for the first test case. The arrows denote possible throw directions. Players who could have the ball after the throw are highlighted in gray.



E. Rudolf and k Bridges

2 seconds, 256 megabytes

Bernard loves visiting Rudolf, but he is always running late. The problem is that Bernard has to cross the river on a ferry. Rudolf decided to help his friend solve this problem.

The river is a grid of n rows and m columns. The intersection of the i-th row and the j-th column contains the number $a_{i,j}$ — the depth in the corresponding cell. All cells in the **first** and **last** columns correspond to the river banks, so the depth for them is 0.

0	1	2	3	4	5	4	3	2	1	0
0	1	2	3	2	1	2	3	3	2	0
0	1	2	3	5	5	5	5	5	2	0

The river may look like this.

Rudolf can choose the row $(i, 1), (i, 2), \ldots, (i, m)$ and build a bridge over it. In each cell of the row, he can install a support for the bridge. The cost of installing a support in the cell (i, j) is $a_{i,j} + 1$. Supports must be installed so that the following conditions are met:

1. A support must be installed in cell $(i, 1)$;
2. A support must be installed in cell (i, m);
3. The distance between any pair of adjacent supports must be **no more than** d. The distance between supports (i, j_1) and (i, j_2) is $|j_1 - j_2| - 1$.

Building just one bridge is boring. Therefore, Rudolf decided to build k bridges on **consecutive** rows of the river, that is, to choose some i ($1 \le i \le n - k + 1$) and independently build a bridge on each of the rows $i, i + 1, \ldots, i + k - 1$. Help Rudolf **minimize** the total cost of installing supports.

Input

The first line contains a single integer t ($1 \le t \le 10^3$) — the number of test cases. The descriptions of the test cases follow.

The first line of each test case contains four integers n, m, k, and d ($1 \le k \le n \le 100$, $3 \le m \le 2 \cdot 10^5$, $1 \le d \le m$) — the number of rows and columns of the field, the number of bridges, and the maximum distance between supports.

Then follow n lines, i-th line contains m positive integers $a_{i,j}$ ($0 \le a_{i,j} \le 10^6$, $a_{i,1} = a_{i,m} = 0$) — the depths of the river cells.

It is guaranteed that the sum of $n \cdot m$ for all sets of input data does not exceed $2 \cdot 10^5$.

Output

For each test case, output a single number — the minimum total cost of supports installation.

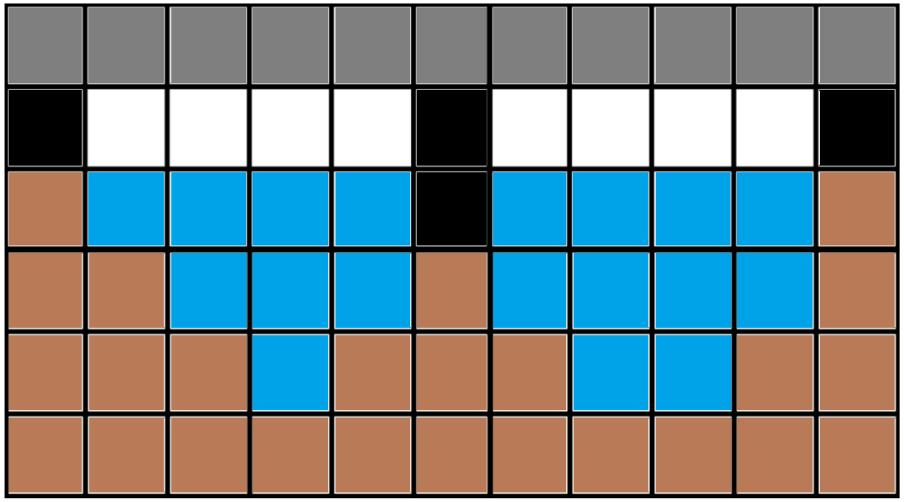
input

```
5
3 11 1 4
0 1 2 3 4 5 4 3 2 1 0
0 1 2 3 2 1 2 3 3 2 0
0 1 2 3 5 5 5 5 5 2 0
4 4 2 1
0 3 3 0
0 2 1 0
0 1 2 0
0 3 3 0
4 5 2 5
0 1 1 1 0
0 2 2 2 0
0 2 1 1 0
0 3 2 1 0
1 8 1 1
0 10 4 8 4 4 2 0
4 5 3 2
0 8 4 4 0
0 3 4 8 0
0 8 1 10 0
0 10 1 5 0
```

output

```
4
8
4
15
14
```

In the first test case, it is most profitable to build a bridge on the second row.



It is not a top view, but **side view**: gray cells — bridge itself, white cells are empty, black cells — supports, blue cells — water, brown cells — river bottom.

In the second test case, it is most profitable to build bridges on the second and third rows. The supports will be placed in cells $(2, 3)$, $(3, 2)$, and on the river banks.

In the third test case the supports can be placed along the river banks.

F. Rudolf and Imbalance

3 seconds, 256 megabytes

Rudolf has prepared a set of n problems with complexities $a_1 < a_2 < a_3 < \cdots < a_n$. He is not entirely satisfied with the balance, so he wants to add **at most one** problem to fix it.

For this, Rudolf came up with m models of problems and k functions. The complexity of the i-th model is d_i, and the complexity of the j-th function is f_j. To create a problem, he selects values i and j ($1 \le i \le m$, $1 \le j \le k$) and by combining the i-th model with the j-th function, he obtains a new problem with complexity $d_i + f_j$.

To determine the *imbalance* of the set, Rudolf sorts the complexities of the problems in ascending order and finds the largest value of $a_i - a_{i-1}$ ($i > 1$).

What is the minimum value of *imbalance* that Rudolf can achieve by adding at most one problem, created according to the described rules?

Input

The first line of the input contains a single integer t ($1 \le t \le 10^4$) — the number of testcases.

The first line of each test case contains three integers n, m, and k ($2 \le n \le 10^5$, $1 \le m, k \le 2 \cdot 10^5$) — the number of prepared problems, the number of models, and the number of functions, respectively.

The second line of each test case contains n integers $a_1, a_2, a_3, \ldots a_n$ ($1 \le a_i \le 2 \cdot 10^9$, $a_i < a_{i+1}$) — the complexities of the prepared problems.

The third line of each test case contains m integers $d_1, d_2, d_3, \ldots d_m$ ($1 \le d_i \le 10^9$) — the complexities of the models.

The fourth line of each test case contains k integers $f_1, f_2, f_3, \ldots f_k$ ($1 \le f_i \le 10^9$) — the complexities of the functions.

It is guaranteed that the sum of n over all testcases does not exceed 10^5.

It is guaranteed that the sum of m over all testcases does not exceed $2 \cdot 10^5$.

It is guaranteed that the sum of k over all testcases does not exceed $2 \cdot 10^5$.

Output

For each testcase, output a single number — the minimum *imbalance* that Rudolf can achieve.

input

```
7
5 5 5
5 10 15 20 26
11 14 16 13 8
16 4 5 3 1
7 6 5
1 4 7 10 18 21 22
2 3 5 7 4 2
6 8 9 3 2
7 6 5
1 4 7 10 18 21 22
2 3 5 7 4 2
6 8 13 3 2
5 6 3
2 10 13 20 25
11 6 10 16 14 5
6 17 15
4 2 2
11 12 14 15
19 14
10 6
8 4 2
3 10 16 18 21 22 29 30
9 13 16 15
4 2
2 4 7
4 21
4 15 14 5
20 1 15 1 12 5 11
```

output

```
5
4
5
8
2
7
11
```

G. Rudolf and Subway

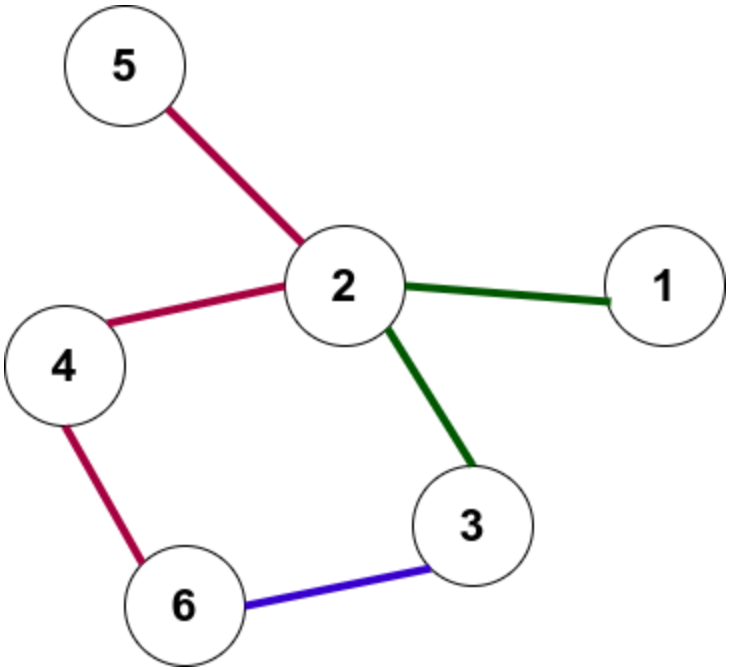
2 seconds, 256 megabytes

Building bridges did not help Bernard, and he continued to be late everywhere. Then Rudolf decided to teach him how to use the subway.

Rudolf depicted the subway map as an undirected connected graph, without self-loops, where the vertices represent stations. There is at most one edge between any pair of vertices.

Two vertices are connected by an edge if it is possible to travel directly between the corresponding stations, bypassing other stations. The subway in the city where Rudolf and Bernard live has a color notation. This means that any edge between stations has a specific color. Edges of a specific color together form a subway line. A subway line **cannot** contain unconnected edges and forms a connected subgraph of the given subway graph.

An example of the subway map is shown in the figure.



Rudolf claims that the route will be optimal if it passes through the minimum number of subway lines.

Help Bernard determine this minimum number for the given departure and destination stations.

Input

The first line contains an integer t ($1 \le t \le 10^4$) — the number of test cases.

This is followed by descriptions of the test cases.

The first line of each test case contains two integers n and m ($2 \le n \le 2 \cdot 10^5, 1 \le m \le 2 \cdot 10^5$) — the number of subway stations and the number of direct routes between stations (i.e., graph edges).

This is followed by m lines — the description of the edges. Each line of the description contains three integers u, v, and c ($1 \le u, v \le n, u \ne v, 1 \le c \le 2 \cdot 10^5$) — the numbers of the vertices between which there is an edge, and the color of this edge. It is guaranteed that edges of the same color form a connected subgraph of the given subway graph. There is at most one edge between a pair of any two vertices.

This is followed by two integers b and e ($1 \le b, e \le n$) — the departure and destination stations.

The sum of all n over all test cases does not exceed $2 \cdot 10^5$. The sum of all m over all test cases does not exceed $2 \cdot 10^5$.

Output

For each testcase, output a single integer — the minimum number of subway lines through which the route from station b to station e can pass.

input

```
5
6 6
1 2 1
2 3 1
5 2 2
2 4 2
4 6 2
3 6 3
1 3
6 6
1 2 1
2 3 1
5 2 2
2 4 2
4 6 2
3 6 3
1 6
6 6
1 2 1
2 3 1
5 2 2
2 4 2
4 6 2
3 6 3
6 6
4 3
1 2 1
1 3 1
4 1 1
2 3
6 7
1 2 43
1 3 34
4 6 43
6 3 43
2 3 43
5 3 43
4 5 43
1 6
```

output

```
1
2
0
1
1
```

input

```
3
7 9
2 4 1
3 6 1
2 3 5
1 7 1
4 7 1
2 5 4
5 4 4
3 4 1
3 7 1
5 3
6 5
6 5 83691
4 1 83691
5 4 83691
3 2 83691
4 3 83691
5 1
6 7
6 1 83691
6 2 83691
2 5 83691
5 6 83691
2 3 83691
5 4 83574
3 5 83691
1 4
```

output

```
2
1
2
```

The subway graph for the first example is shown in the figure in the problem statement.

In the first test case, from vertex 1 to vertex 3, you can travel along the path $1 \to 2 \to 3$, using only the green line.

In the second test case, from vertex 1 to vertex 6, you can travel along the path $1 \to 2 \to 3 \to 6$, using the green and blue lines.

In the third test case, there is no need to travel from vertex 6 to the same vertex, so the number of lines is 0.

In the fourth test case, all edges of the graph belong to one line, so the answer is 1.
