Codeforces Round #575 (Div. 3)

A. Three Piles of Candies

1 second, 256 megabytes

Alice and Bob have received three big piles of candies as a gift. Now they want to divide these candies as fair as possible. To do this, Alice takes one pile of candies, then Bob takes one of the other two piles. The last pile is split between Alice and Bob as they want: for example, it is possible that Alice takes the whole pile, and Bob gets nothing from it.

After taking the candies from the piles, if Alice has more candies than Bob, she discards some candies so that the number of candies she has is equal to the number of candies Bob has. Of course, Bob does the same if he has more candies.

Alice and Bob want to have as many candies as possible, and they plan the process of dividing candies accordingly. Please calculate the maximum number of candies Alice can have after this division process (of course, Bob will have the same number of candies).

You have to answer q independent queries.

Let's see the following example: $[1, 3, 4]$. Then Alice can choose the third pile, Bob can take the second pile, and then the only candy from the first pile goes to Bob — then Alice has 4 candies, and Bob has 4 candies.

Another example is $[1, 10, 100]$. Then Alice can choose the second pile, Bob can choose the first pile, and candies from the third pile can be divided in such a way that Bob takes 54 candies, and Alice takes 46 candies. Now Bob has 55 candies, and Alice has 56 candies, so she has to discard one candy — and after that, she has 55 candies too.

Input

The first line of the input contains one integer q ($1 \le q \le 1000$) — the number of queries. Then q queries follow.

The only line of the query contains three integers a, b and c ($1 \le a, b, c \le 10^{16}$) — the number of candies in the first, second and third piles correspondingly.

Output

Print q lines. The i-th line should contain the answer for the i-th query — the maximum number of candies Alice can have after the division, if both Alice and Bob act optimally (of course, Bob will have the same number of candies).

input

```
4
1 3 4
1 10 100
10000000000000000 10000000000000000 10000000000000000
23 34 45
```

output

```
4
55
15000000000000000
51
```

B. Odd Sum Segments

3 seconds, 256 megabytes

You are given an array a consisting of n integers $a_1, a_2, \dots, a_n$. You want to split it into exactly k **non-empty non-intersecting subsegments** such that each subsegment has odd sum (i. e. for each subsegment, the sum of all elements that belong to this subsegment is odd). It is impossible to rearrange (shuffle) the elements of a given array. Each of the n elements of the array a must belong to exactly one of the k subsegments.

Let's see some examples of dividing the array of length 5 into 3 subsegments (not necessarily with odd sums): $[1, 2, 3, 4, 5]$ is the initial array, then all possible ways to divide it into 3 non-empty non-intersecting subsegments are described below:

- $[1], [2], [3, 4, 5]$;
- $[1], [2, 3], [4, 5]$;
- $[1], [2, 3, 4], [5]$;
- $[1, 2], [3], [4, 5]$;
- $[1, 2], [3, 4], [5]$;
- $[1, 2, 3], [4], [5]$.

Of course, it can be impossible to divide the initial array into exactly k subsegments in such a way that each of them will have odd sum of elements. In this case print "NO". Otherwise, print "YES" and **any** possible division of the array. See the output format for the detailed explanation.

You have to answer q independent queries.

Input

The first line contains one integer q ($1 \le q \le 2 \cdot 10^5$) — the number of queries. Then q queries follow.

The first line of the query contains two integers n and k ($1 \le k \le n \le 2 \cdot 10^5$) — the number of elements in the array and the number of subsegments, respectively.

The second line of the query contains n integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le 10^9$), where a_i is the i-th element of a.

It is guaranteed that the sum of n over all queries does not exceed $2 \cdot 10^5$ ($\sum n \le 2 \cdot 10^5$).

Output

For each query, print the answer to it. If it is impossible to divide the initial array into exactly k subsegments in such a way that each of them will have odd sum of elements, print "NO" in the first line. Otherwise, print "YES" in the first line and **any** possible division of the array in the second line. The division can be represented as k integers r_1, r_2, ..., r_k such that $1 \le r_1 < r_2 < \dots < r_k = n$, where r_j is the right border of the j-th segment (the index of the last element that belongs to the j-th segment), so the array is divided into subsegments $[1; r_1], [r_1 + 1; r_2], [r_2 + 1, r_3], \dots, [r_{k-1} + 1, n]$ **Note that r_k is always n but you should print it anyway**.

input

```
3
5 3
7 18 3 14 1
5 4
1 2 3 4 5
6 2
1 2 8 4 10 2
```

output

```
YES
1 3 5
NO
NO
```

C. Robot Breakout

3 seconds, 256 megabytes

n robots have escaped from your laboratory! You have to find them as soon as possible, because these robots are experimental, and their behavior is not tested yet, so they may be really dangerous!

Fortunately, even though your robots have escaped, you still have some control over them. First of all, you know the location of each robot: the world you live in can be modeled as an infinite coordinate plane, and the i-th robot is currently located at the point having coordinates (x_i, y_i). Furthermore, you may send exactly one command to all of the robots. The command should contain two integer numbers X and Y, and when each robot receives this command, it starts moving towards the point having coordinates (X, Y). The robot stops its movement in two cases:

- either it reaches (X, Y);
- or it cannot get any closer to (X, Y).

Normally, all robots should be able to get from any point of the coordinate plane to any other point. Each robot usually can perform four actions to move. Let's denote the current coordinates of the robot as (x_c, y_c). Then the movement system allows it to move to any of the four adjacent points:

1. the first action allows it to move from (x_c, y_c) to $(x_c - 1, y_c)$;
2. the second action allows it to move from (x_c, y_c) to $(x_c, y_c + 1)$;
3. the third action allows it to move from (x_c, y_c) to $(x_c + 1, y_c)$;
4. the fourth action allows it to move from (x_c, y_c) to $(x_c, y_c - 1)$.

Unfortunately, it seems that some movement systems of some robots are malfunctioning. For each robot you know which actions it can perform, and which it cannot perform.

You want to send a command so all robots gather at the same point. To do so, you have to choose a pair of integer numbers X and Y so that each robot can reach the point (X, Y). Is it possible to find such a point?

Input

The first line contains one integer q ($1 \le q \le 10^5$) — the number of queries.

Then q queries follow. Each query begins with one line containing one integer n ($1 \le n \le 10^5$) — the number of robots in the query. Then n lines follow, the i-th of these lines describes the i-th robot in the current query: it contains six integer numbers x_i, y_i, $f_{i,1}$, $f_{i,2}$, $f_{i,3}$ and $f_{i,4}$ ($-10^5 \le x_i, y_i \le 10^5$, $0 \le f_{i,j} \le 1$). The first two numbers describe the initial location of the i-th robot, and the following four numbers describe which actions the i-th robot can use to move ($f_{i,j} = 1$ if the i-th robot can use the j-th action, and $f_{i,j} = 0$ if it cannot use the j-th action).

It is guaranteed that the total number of robots over all queries does not exceed 10^5.

Output

You should answer each query independently, in the order these queries appear in the input.

To answer a query, you should do one of the following:

- if it is impossible to find a point that is reachable by all n robots, print one number 0 on a separate line;
- if it is possible to find a point that is reachable by all n robots, print three space-separated integers on the same line: 1 X Y, where X and Y are the coordinates of the point reachable by all n robots. *Both X and Y should not exceed 10^5 by absolute value; it is guaranteed that if there exists at least one point reachable by all robots, then at least one of such points has both coordinates not exceeding 10^5 by absolute value.*

input

```
4
2
-1 -2 0 0 0 0
-1 -2 0 0 0 0
3
1 5 1 1 1 1
2 5 0 1 0 1
3 5 1 0 0 0
2
1337 1337 0 1 1 1
1336 1337 1 1 0 1
1
3 5 1 1 1 1
```

output

```
1 -1 -2
1 2 5
0
1 -100000 -100000
```

D1. RGB Substring (easy version)

2 seconds, 256 megabytes

The only difference between easy and hard versions is the size of the input.

You are given a string s consisting of n characters, each character is 'R', 'G' or 'B'.

You are also given an integer k. Your task is to change the minimum number of characters in the initial string s so that after the changes there will be a string of length k that is a substring of s, and is also a substring of the infinite string "RGBRGBRGB ...".

A string a is a substring of string b if there exists a positive integer i such that $a_1 = b_i$, $a_2 = b_{i+1}$, $a_3 = b_{i+2}$, ..., $a_{|a|} = b_{i+|a|-1}$. For example, strings "GBRG", "B", "BR" are substrings of the infinite string "RGBRGBRGB ..." while "GR", "RGR" and "GGG" are not.

You have to answer q independent queries.

Input

The first line of the input contains one integer q ($1 \le q \le 2000$) — the number of queries. Then q queries follow.

The first line of the query contains two integers n and k ($1 \le k \le n \le 2000$) — the length of the string s and the length of the substring.

The second line of the query contains a string s consisting of n characters 'R', 'G' and 'B'.

It is guaranteed that the sum of n over all queries does not exceed 2000 ($\sum n \le 2000$).

Output

For each query print one integer — the minimum number of characters you need to change in the initial string s so that after changing there will be a substring of length k in s that is also a substring of the infinite string "RGBRGBRGB ...".

input

```
3
5 2
BGGGG
5 3
RBRGR
5 5
BBBRR
```

output

```
1
0
3
```

In the first example, you can change the first character to 'R' and obtain the substring "RG", or change the second character to 'R' and obtain "BR", or change the third, fourth or fifth character to 'B' and obtain "GB".

In the second example, the substring is "BRG".

D2. RGB Substring (hard version)

2 seconds, 256 megabytes

The only difference between easy and hard versions is the size of the input.

You are given a string s consisting of n characters, each character is 'R', 'G' or 'B'.

You are also given an integer k. Your task is to change the minimum number of characters in the initial string s so that after the changes there will be a string of length k that is a substring of s, and is also a substring of the infinite string "RGBRGBRGB ...".

A string a is a substring of string b if there exists a positive integer i such that $a_1 = b_i$, $a_2 = b_{i+1}$, $a_3 = b_{i+2}$, ..., $a_{|a|} = b_{i+|a|-1}$. For example, strings "GBRG", "B", "BR" are substrings of the infinite string "RGBRGBRGB ..." while "GR", "RGR" and "GGG" are not.

You have to answer q independent queries.

Input

The first line of the input contains one integer q ($1 \le q \le 2 \cdot 10^5$) — the number of queries. Then q queries follow.

The first line of the query contains two integers n and k ($1 \le k \le n \le 2 \cdot 10^5$) — the length of the string s and the length of the substring.

The second line of the query contains a string s consisting of n characters 'R', 'G' and 'B'.

It is guaranteed that the sum of n over all queries does not exceed $2 \cdot 10^5$ ($\sum n \le 2 \cdot 10^5$).

Output

For each query print one integer — the minimum number of characters you need to change in the initial string s so that after changing there will be a substring of length k in s that is also a substring of the infinite string "RGBRGBRGB ...".

input

```
3
5 2
BGGGG
5 3
RBRGR
5 5
BBBRR
```

output

```
1
0
3
```

In the first example, you can change the first character to 'R' and obtain the substring "RG", or change the second character to 'R' and obtain "BR", or change the third, fourth or fifth character to 'B' and obtain "GB".

In the second example, the substring is "BRG".

E. Connected Component on a Chessboard

2 seconds, 256 megabytes

You are given two integers b and w. You have a chessboard of size $10^9 \times 10^9$ with the top left cell at $(1; 1)$, the cell $(1; 1)$ is painted **white**.

Your task is to find a connected component on this chessboard that contains exactly b black cells and exactly w white cells. Two cells are called connected if they share a side (i.e. for the cell (x, y) there are at most four connected cells: $(x-1, y), (x+1, y), (x, y-1), (x, y+1)$). A set of cells is called a connected component if for every pair of cells C_1 and C_2 from this set, there exists a sequence of cells c_1, c_2, ..., c_k such that $c_1 = C_1$, $c_k = C_2$, all c_i from 1 to k are belong to this set of cells and for every $i \in [1, k-1]$, cells c_i and c_{i+1} are connected.

Obviously, it can be impossible to find such component. In this case print "NO". Otherwise, print "YES" and **any** suitable connected component.

You have to answer q independent queries.

Input

The first line of the input contains one integer q ($1 \le q \le 10^5$) — the number of queries. Then q queries follow.

The only line of the query contains two integers b and w ($1 \le b, w \le 10^5$) — the number of black cells required and the number of white cells required.

It is guaranteed that the sum of numbers of cells does not exceed $2 \cdot 10^5$ ($\sum w + \sum b \le 2 \cdot 10^5$).

Output

For each query, print the answer to it.

If it is impossible to find the required component, print "NO" on the first line.

Otherwise, print "YES" on the first line. In the next $b + w$ lines print coordinates of cells of your component in any order. There should be exactly b black cells and w white cells in your answer. The printed component should be connected.

If there are several answers, you can print **any**. All coordinates in the answer should be in the range $[1; 10^9]$.

input

```
3
1 1
1 4
2 5
```

output

```
YES
2 2
1 2
YES
2 3
1 3
3 3
2 2
2 4
YES
2 3
2 4
2 5
1 3
1 5
3 3
3 5
```

F. K-th Path

2.5 seconds, 256 megabytes

You are given a connected undirected weighted graph consisting of n vertices and m edges.

You need to print the k-th smallest shortest path in this graph (paths from the vertex to itself are not counted, paths from i to j and from j to i are counted as one).

More formally, if d is the matrix of shortest paths, where $d_{i,j}$ is the length of the shortest path between vertices i and j ($1 \le i < j \le n$), then you need to print the k-th element in the sorted array consisting of all $d_{i,j}$, where $1 \le i < j \le n$.

Input

The first line of the input contains three integers n, m and k ($2 \le n \le 2 \cdot 10^5$, $n - 1 \le m \le \min\left(\frac{n(n-1)}{2}, 2 \cdot 10^5\right)$, $1 \le k \le \min\left(\frac{n(n-1)}{2}, 400\right)$ — the number of vertices in the graph, the number of edges in the graph and the value of k, correspondingly.

Then m lines follow, each containing three integers x, y and w ($1 \le x, y \le n$, $1 \le w \le 10^9$, $x \ne y$) denoting an edge between vertices x and y of weight w.

It is guaranteed that the given graph is **connected** (there is a path between any pair of vertices), there are no self-loops (edges connecting the vertex with itself) and multiple edges (for each pair of vertices x and y, there is at most one edge between this pair of vertices in the graph).

Output

Print one integer — the length of the k-th smallest shortest path in the given graph (paths from the vertex to itself are not counted, paths from i to j and from j to i are counted as one).

input

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

output

```
3
```

input

```
7 15 18
2 6 3
5 7 4
6 5 4
3 6 9
6 7 7
1 6 4
7 1 6
7 2 1
4 3 2
3 2 8
5 3 6
2 5 5
3 7 9
4 1 8
2 1 1
```

output

```
9
```