Educational Codeforces Round 128 (Rated for Div. 2)

A. Minimums and Maximums

time limit per test: 2 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

An array is *beautiful* if both of the following two conditions meet:

- there are **at least** l_1 and **at most** r_1 elements in the array equal to its minimum;
- there are **at least** l_2 and **at most** r_2 elements in the array equal to its maximum.

For example, the array $[2, 3, 2, 4, 4, 3, 2]$ has 3 elements equal to its minimum (1-st, 3-rd and 7-th) and 2 elements equal to its maximum (4-th and 5-th).

Another example: the array $[42, 42, 42]$ has 3 elements equal to its minimum and 3 elements equal to its maximum.

Your task is to calculate the **minimum** possible number of elements in a *beautiful* array.

Input

The first line contains one integer t ($1 \le t \le 5000$) — the number of test cases.

Each test case consists of one line containing four integers l_1, r_1, l_2 and r_2 ($1 \le l_1 \le r_1 \le 50$; $1 \le l_2 \le r_2 \le 50$).

Output

For each test case, print one integer — the minimum possible number of elements in a beautiful array.

Example

input
7 3 5 4 6 5 8 5 5 3 3 10 12 1 5 3 3 1 1 2 2 2 2 1 1 6 6 6 6
output
4 5 13 3 3 3 6

Note

Optimal arrays in the test cases of the example:

1. $[1, 1, 1, 1]$, it has 4 minimums and 4 maximums;
2. $[4, 4, 4, 4, 4]$, it has 5 minimums and 5 maximums;
3. $[1, 2, 1, 2, 2, 1, 2, 2, 2, 2, 2, 2]$, it has 3 minimums and 10 maximums;
4. $[8, 8, 8]$, it has 3 minimums and 3 maximums;
5. $[4, 6, 6]$, it has 1 minimum and 2 maximums;
6. $[3, 4, 3]$, it has 2 minimums and 1 maximum;
7. $[5, 5, 5, 5, 5, 5]$, it has 6 minimums and 6 maximums.

B. Robots

time limit per test: 2 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

There is a field divided into n rows and m columns. Some cells are empty (denoted as E), other cells contain robots (denoted as R).

You can send a command to **all robots** at the same time. The command can be of one of the four types:

- move up;

- move right;
- move down;
- move left.

When you send a command, **all robots at the same time** attempt to take one step in the direction you picked. If a robot tries to move outside the field, it explodes; otherwise, **every robot** moves to an adjacent cell in the chosen direction.

You can send as many commands as you want (possibly, zero), in any order. Your goal is to make at least one robot reach the upper left corner of the field. Can you do this without forcing any of the robots to explode?

Input
The first line contains one integer t ($1 \le t \le 5000$) — the number of test cases.

Each test case starts with a line containing two integers n and m ($1 \le n, m \le 5$) — the number of rows and the number of columns, respectively. Then n lines follow; each of them contains a string of m characters. Each character is either E (empty cell} or R (robot).

Additional constraint on the input: in each test case, there is at least one robot on the field.

Output
If it is possible to make at least one robot reach the upper left corner of the field so that no robot explodes, print YES. Otherwise, print NO.

Example

input
6 1 3 ERR 2 2 ER RE 2 2 ER ER 1 1 R 4 3 EEE EEE ERR EER 3 3 EEE EER REE
output
YES NO YES YES YES NO

Note
Explanations for test cases of the example:

1. in the first test case, it is enough to send a command to move left.
2. in the second test case, if you try to send any command, at least one robot explodes.
3. in the third test case, it is enough to send a command to move left.
4. in the fourth test case, there is already a robot in the upper left corner.
5. in the fifth test case, the sequence "move up, move left, move up" leads one robot to the upper left corner;
6. in the sixth test case, if you try to move any robot to the upper left corner, at least one other robot explodes.

C. Binary String

time limit per test: 2 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

You are given a string s consisting of characters 0 and/or 1.

You have to remove several (possibly zero) characters from the beginning of the string, and then several (possibly zero) characters from the end of the string. **The string may become empty after the removals**. The cost of the removal is the **maximum** of the following two values:

- the number of characters 0 left in the string;
- the number of characters 1 removed from the string.

What is the **minimum** cost of removal you can achieve?

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

Each test case consists of one line containing the string s ($1 \le |s| \le 2 \cdot 10^5$), consisting of characters `0` and/or `1`.

The total length of strings s in all test cases does not exceed $2 \cdot 10^5$.

Output
For each test case, print one integer — the minimum cost of removal you can achieve.

Example

input
5 101110110 1001001001001 0000111111 00000 1111
output
1 3 0 0 0

Note
Consider the test cases of the example:

1. in the first test case, it's possible to remove two characters from the beginning and one character from the end. Only one `1` is deleted, only one `0` remains, so the cost is 1;
2. in the second test case, it's possible to remove three characters from the beginning and six characters from the end. Two characters `0` remain, three characters `1` are deleted, so the cost is 3;
3. in the third test case, it's optimal to remove four characters from the beginning;
4. in the fourth test case, it's optimal to remove the whole string;
5. in the fifth test case, it's optimal to leave the string as it is.

D. Dog Walking

time limit per test: 4 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are walking with your dog, and now you are at the promenade. The promenade can be represented as an infinite line. Initially, you are in the point 0 with your dog.

You decided to give some freedom to your dog, so you untied her and let her run for a while. Also, you watched what your dog is doing, so you have some writings about how she ran. During the i-th minute, the dog position changed from her previous position by the value a_i (it means, that the dog ran for a_i meters during the i-th minute). If a_i is positive, the dog ran a_i meters to the right, otherwise (if a_i is negative) she ran a_i meters to the left.

During some minutes, you were chatting with your friend, so you don't have writings about your dog movement during these minutes. These values a_i equal zero.

You want your dog to return to you after the end of the walk, so the destination point of the dog after n minutes **should be** 0.

Now you are wondering: what is the maximum possible number of different **integer points** of the line your dog could visit on her way, if you replace every 0 with some integer from $-k$ to k (and your dog **should** return to 0 after the walk)? The dog visits an integer point if she runs through that point or reaches in it at the end of any minute. Point 0 is always visited by the dog, since she is initially there.

If the dog cannot return to the point 0 after n minutes regardless of the integers you place, print `-1`.

Input
The first line of the input contains two integers n and k ($1 \le n \le 3000; 1 \le k \le 10^9$) — the number of minutes and the maximum possible speed of your dog during the minutes without records.

The second line of the input contains n integers $a_1, a_2, \ldots, a_n$ ($-10^9 \le a_i \le 10^9$), where a_i is the number of meters your dog ran during the i-th minutes (to the left if a_i is negative, to the right otherwise). If $a_i = 0$ then this value is **unknown** and can be replaced with any integer from the range $[-k; k]$.

Output
If the dog cannot return to the point 0 after n minutes regardless of the set of integers you place, print `-1`. Otherwise, print one integer — the maximum number of **different** integer points your dog could visit if you fill all the unknown values optimally and the

dog will return to the point 0 at the end of the walk.

Examples

input
3 2 5 0 -4
output
6

input
6 4 1 -2 0 3 -4 5
output
7

input
3 1000000000 0 0 0
output
1000000001

input
5 9 -7 -3 8 12 0
output
-1

input
5 3 -1 0 3 3 0
output
7

input
5 4 0 2 0 3 0
output
9

E. Moving Chips

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given a board of size $2 \times n$ (2 rows, n columns). Some cells of the board contain chips. The chip is represented as '*', and an empty space is represented as '.'. It is guaranteed that there is at least one chip on the board.

In one move, you can choose **any** chip and move it to any adjacent (by side) cell of the board (if this cell is inside the board). It means that if the chip is in the first row, you can move it left, right or down (but it shouldn't leave the board). Same, if the chip is in the second row, you can move it left, right or up.

If the chip moves to the cell with another chip, the chip in the destination cell disappears (i. e. our chip captures it).

Your task is to calculate the **minimum** number of moves required to leave **exactly** one chip on the board.

You have to answer t independent test cases.

Input

The first line of the input contains one integer t ($1 \le t \le 2 \cdot 10^4$) — the number of test cases. Then t test cases follow.

The first line of the test case contains one integer n ($1 \le n \le 2 \cdot 10^5$) — the length of the board. The second line of the test case contains the string s_1 consisting of n characters '*' (chip) and/or '.' (empty cell). The third line of the test case contains the string s_2 consisting of n characters '*' (chip) and/or '.' (empty cell).

Additional constraints on the input:

- in each test case, there is at least one chip on a board;

- the sum of n over all test cases does not exceed $2 \cdot 10^5$ ($\sum n \le 2 \cdot 10^5$).

Output
For each test case, print one integer — the **minimum** number of moves required to leave **exactly** one chip on the board.

Example

input
5 1 * . 2 .* ** 3 *.* .*. 4 **.* **.. 5 **... ...**

output
0 2 3 5 5

F. Lenient Vertex Cover

time limit per test: 5 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

You are given a simple connected undirected graph, consisting of n vertices and m edges. The vertices are numbered from 1 to n.

A vertex cover of a graph is a set of vertices such that each edge has at least one of its endpoints in the set.

Let's call a *lenient* vertex cover such a vertex cover that **at most one** edge in it has both endpoints in the set.

Find a *lenient* vertex cover of a graph or report that there is none. If there are multiple answers, then print any of them.

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

The first line of each testcase contains two integers n and m ($2 \le n \le 10^6$; $n - 1 \le m \le \min(10^6, \frac{n \cdot (n-1)}{2})$) — the number of vertices and the number of edges of the graph.

Each of the next m lines contains two integers v and u ($1 \le v, u \le n$; $v \ne u$) — the descriptions of the edges.

For each testcase, the graph is connected and doesn't have multiple edges. The sum of n over all testcases doesn't exceed 10^6. The sum of m over all testcases doesn't exceed 10^6.

Output
For each testcase, the first line should contain YES if a *lenient* vertex cover exists, and NO otherwise. If it exists, the second line should contain a binary string s of length n, where $s_i = 1$ means that vertex i is in the vertex cover, and $s_i = 0$ means that vertex i isn't.

If there are multiple answers, then print any of them.

Examples

input
4 6 5 1 3 2 4 3 4 3 5 4 6 4 6 1 2 2 3 3 4 1 4 1 3 2 4 8 11 1 3

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

output

```
YES
001100
NO
YES
01100110
YES
0110
```

input

```
1
10 15
9 4
3 4
6 4
1 2
8 2
8 3
7 2
9 5
7 8
5 10
1 4
2 10
5 3
5 7
2 9
```

output

```
YES
0101100100
```

input

```
1
10 19
7 9
5 3
3 4
1 6
9 4
1 4
10 5
7 1
9 2
8 3
7 3
10 9
2 10
9 8
3 2
1 5
10 7
9 5
1 2
```

output

```
YES
1010000011
```

Note

Here are the graphs from the first example. The vertices in the *lenient* vertex covers are marked red.