Codeforces Round #668 (Div. 2)

A. Permutation Forgery

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

A permutation of length n is an array consisting of n distinct integers from 1 to n in arbitrary order. For example, $[2,3,1,5,4]$ is a permutation, but $[1,2,2]$ is not a permutation (2 appears twice in the array) and $[1,3,4]$ is also not a permutation ($n=3$ but there is 4 in the array).

Let p be any permutation of length n. We define the **fingerprint** $F(p)$ of p as the sorted array of sums of adjacent elements in p. More formally,

$$F(p) = \text{sort}([p_1+p_2, p_2+p_3, \ldots, p_{n-1}+p_n]).$$

For example, if $n=4$ and $p=[1,4,2,3]$, then the fingerprint is given by $F(p) = \text{sort}([1+4, 4+2, 2+3]) = \text{sort}([5,6,5]) = [5,5,6]$.

You are given a permutation p of length n. Your task is to find a **different** permutation p' with the same fingerprint. Two permutations p and p' are considered different if there is some index i such that $p_i \neq p'_i$.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 668$). Description of the test cases follows.

The first line of each test case contains a single integer n ($2 \le n \le 100$) — the length of the permutation.

The second line of each test case contains n integers $p_1, \ldots, p_n$ ($1 \le p_i \le n$). It is guaranteed that p is a permutation.

Output

For each test case, output n integers $p'_1, \ldots, p'_n$ — a permutation such that $p' \neq p$ and $F(p') = F(p)$.

We can prove that for every permutation satisfying the input constraints, a solution exists.

If there are multiple solutions, you may output any.

Example

input
3 2 1 2 6 2 1 6 5 4 3 5 2 4 3 1 5
output
2 1 1 2 5 6 3 4 3 1 5 2 4

Note

In the first test case, $F(p) = \text{sort}([1+2]) = [3]$.

And $F(p') = \text{sort}([2+1]) = [3]$.

In the second test case, $F(p) = \text{sort}([2+1, 1+6, 6+5, 5+4, 4+3]) = \text{sort}([3,7,11,9,7]) = [3,7,7,9,11]$.

And $F(p') = \text{sort}([1+2, 2+5, 5+6, 6+3, 3+4]) = \text{sort}([3,7,11,9,7]) = [3,7,7,9,11]$.

In the third test case, $F(p) = \text{sort}([2+4, 4+3, 3+1, 1+5]) = \text{sort}([6,7,4,6]) = [4,6,6,7]$.

And $F(p') = \text{sort}([3+1, 1+5, 5+2, 2+4]) = \text{sort}([4,6,7,6]) = [4,6,6,7]$.

B. Array Cancellation

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input

output: standard output

You're given an array a of n integers, such that $a_1 + a_2 + \cdots + a_n = 0$.

In one operation, you can choose two **different** indices i and j ($1 \le i, j \le n$), decrement a_i by one and increment a_j by one. If $i < j$ this operation is free, otherwise it costs one coin.

How many coins do you have to spend in order to make all elements equal to 0?

Input
Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 5000$). Description of the test cases follows.

The first line of each test case contains an integer n ($1 \le n \le 10^5$) — the number of elements.

The next line contains n integers $a_1, \ldots, a_n$ ($-10^9 \le a_i \le 10^9$). It is given that $\sum_{i=1}^{n} a_i = 0$.

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

Output
For each test case, print the minimum number of coins we have to spend in order to make all elements equal to 0.

Example

input
7 4 -3 5 -3 1 2 1 -1 4 -3 2 -3 4 4 -1 1 1 -1 7 -5 7 -6 -4 17 -13 4 6 -1000000000 -1000000000 -1000000000 1000000000 1000000000 1000000000 1 0
output
3 0 4 1 8 3000000000 0

Note
Possible strategy for the first test case:

- Do $(i = 2, j = 3)$ three times (free), $a = [-3, 2, 0, 1]$.
- Do $(i = 2, j = 1)$ two times (pay two coins), $a = [-1, 0, 0, 1]$.
- Do $(i = 4, j = 1)$ one time (pay one coin), $a = [0, 0, 0, 0]$.

C. Balanced Bitstring

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

A bitstring is a string consisting only of the characters `0` and `1`. A bitstring is called k-**balanced** if every substring of size k of this bitstring has an equal amount of `0` and `1` characters ($\frac{k}{2}$ of each).

You are given an integer k and a string s which is composed only of characters `0`, `1`, and `?`. You need to determine whether you can make a k-balanced bitstring by replacing every `?` characters in s with either `0` or `1`.

A string a is a substring of a string b if a can be obtained from b by deletion of several (possibly, zero or all) characters from the beginning and several (possibly, zero or all) characters from the end.

Input
Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10^4$). Description of the test cases follows.

The first line of each test case contains two integers n and k ($2 \le k \le n \le 3 \cdot 10^5$, k is even) — the length of the string and the parameter for a balanced bitstring.

The next line contains the string s ($|s| = n$). It is given that s consists of only `0`, `1`, and `?`.

It is guaranteed that the sum of n over all test cases does not exceed $3 \cdot 10^5$.

Output

For each test case, print YES if we can replace every ? in s with 0 or 1 such that the resulting bitstring is k-balanced, or NO if it is not possible.

Example

input
9 6 4 100110 3 2 1?1 3 2 1?0 4 4 ???? 7 4 1?0??1? 10 10 11??11??11 4 2 1??1 4 4 ?0?0 6 2 ????00
output
YES YES NO YES YES NO NO YES NO

Note

For the first test case, the string is already a 4-balanced bitstring.

For the second test case, the string can be transformed into 101.

For the fourth test case, the string can be transformed into 0110.

For the fifth test case, the string can be transformed into 1100110.

D. Tree Tag

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Alice and Bob are playing a fun game of tree tag.

The game is played on a tree of n vertices numbered from 1 to n. Recall that a tree on n vertices is an undirected, connected graph with $n-1$ edges.

Initially, Alice is located at vertex a, and Bob at vertex b. They take turns alternately, and Alice makes the first move. In a move, Alice can jump to a vertex with distance **at most** da from the current vertex. And in a move, Bob can jump to a vertex with distance **at most** db from the current vertex. The distance between two vertices is defined as the number of edges on the unique simple path between them. In particular, either player is allowed to stay at the same vertex in a move. Note that when performing a move, a player only occupies the starting and ending vertices of their move, not the vertices between them.

If after at most 10^{100} moves, Alice and Bob occupy the same vertex, then Alice is declared the winner. Otherwise, Bob wins.

Determine the winner if both players play optimally.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10^4$). Description of the test cases follows.

The first line of each test case contains five integers n, a, b, da, db ($2 \le n \le 10^5$, $1 \le a, b \le n$, $a \ne b$, $1 \le da, db \le n-1$) — the number of vertices, Alice's vertex, Bob's vertex, Alice's maximum jumping distance, and Bob's maximum jumping distance, respectively.

The following $n-1$ lines describe the edges of the tree. The i-th of these lines contains two integers u, v ($1 \le u, v \le n, u \ne v$), denoting an edge between vertices u and v. It is guaranteed that these edges form a tree structure.

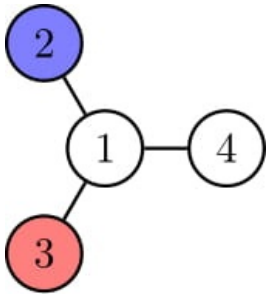
It is guaranteed that the sum of n across all test cases does not exceed 10^5.

Output
For each test case, output a single line containing the winner of the game: "Alice" or "Bob".

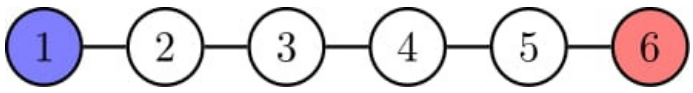
Example

input
4 4 3 2 1 2 1 2 1 3 1 4 6 6 1 2 5 1 2 6 5 2 3 3 4 4 5 9 3 9 2 5 1 2 1 6 1 9 1 3 9 5 7 9 4 8 4 3 11 8 11 3 3 1 2 11 9 4 9 6 5 2 10 3 2 5 9 8 3 7 4 7 10
output
Alice Bob Alice Alice

Note
In the first test case, Alice can win by moving to vertex 1. Then wherever Bob moves next, Alice will be able to move to the same vertex on the next move.



In the second test case, Bob has the following strategy to win. Wherever Alice moves, Bob will always move to whichever of the two vertices 1 or 6 is farthest from Alice.



E. Fixed Point Removal

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

Let $a_1, \ldots, a_n$ be an array of n positive integers. In one operation, you can choose an index i such that $a_i = i$, and remove a_i from the array (after the removal, the remaining parts are concatenated).

The weight of a is defined as the maximum number of elements you can remove.

You must answer q independent queries (x, y): after replacing the x first elements of a and the y last elements of a by $n+1$ (making them impossible to remove), what would be the weight of a?

Input
The first line contains two integers n and q ($1 \leq n, q \leq 3 \cdot 10^5$) — the length of the array and the number of queries.

The second line contains n integers a_1, a_2, ..., a_n ($1 \le a_i \le n$) — elements of the array.

The i-th of the next q lines contains two integers x and y ($x, y \ge 0$ and $x + y < n$).

Output

Print q lines, i-th line should contain a single integer — the answer to the i-th query.

Examples

input
13 5 2 2 3 9 5 4 6 5 7 8 3 11 13 3 1 0 0 2 4 5 0 0 12
output
5 11 6 1 0

input
5 2 1 4 1 2 4 0 0 1 0
output
2 0

Note

Explanation of the first query:

After making first $x = 3$ and last $y = 1$ elements impossible to remove, a becomes $[\times, \times, \times, 9, 5, 4, 6, 5, 7, 8, 3, 11, \times]$ (we represent 14 as $\times$ for clarity).

Here is a strategy that removes 5 elements (the element removed is colored in red):

- $[\times, \times, \times, 9, \color{red}{5}, 4, 6, 5, 7, 8, 3, 11, \times]$
- $[\times, \times, \times, 9, 4, 6, 5, 7, 8, 3, \color{red}{11}, \times]$
- $[\times, \times, \times, 9, 4, \color{red}{6}, 5, 7, 8, 3, \times]$
- $[\times, \times, \times, 9, 4, 5, 7, \color{red}{8}, 3, \times]$
- $[\times, \times, \times, 9, 4, 5, \color{red}{7}, 3, \times]$
- $[\times, \times, \times, 9, 4, 5, 3, \times]$ (final state)

It is impossible to remove more than 5 elements, hence the weight is 5.