Codeforces Round #782 (Div. 2)

A. Red Versus Blue

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

Team Red and Team Blue competed in a competitive FPS. Their match was streamed around the world. They played a series of n matches.

In the end, it turned out Team Red won r times and Team Blue won b times. Team Blue was less skilled than Team Red, so b was **strictly less** than r.

You missed the stream since you overslept, but you think that the match must have been neck and neck since so many people watched it. So you imagine a string of length n where the i-th character denotes who won the i-th match — it is R if Team Red won or B if Team Blue won. You imagine the string was such that the **maximum** number of times a team **won in a row** was **as small as possible**. For example, in the series of matches RBBRRRB, Team Red won 3 times in a row, which is the maximum.

You must find a string satisfying the above conditions. If there are multiple answers, print any.

Input

The first line contains a single integer t ($1 \le t \le 1000$) — the number of test cases.

Each test case has a single line containing three integers n, r, and b ($3 \le n \le 100$; $1 \le b < r \le n$, $r + b = n$).

Output

For each test case, output a single line containing a string satisfying the given conditions. If there are multiple answers, print any.

Examples

input
3 7 4 3 6 5 1 19 13 6
output
RBRBRBR RRRBRR RRBRRBRRBRRBRRBRRBR

input
6 3 2 1 10 6 4 11 6 5 10 9 1 10 8 2 11 9 2
output
RBR RRBRBRBRBR RBRBRBRBRBR RRRRRBRRRR RRRBRRRBRR RRRBRRRBRRR

Note

The first test case of the first example gives the optimal answer for the example in the statement. The maximum number of times a team wins in a row in RBRBRBR is 1. We cannot minimize it any further.

The answer for the second test case of the second example is **RR**BRBRBRBR. The maximum number of times a team wins in a row is 2, given by RR at the beginning. We cannot minimize the answer any further.

B. Bit Flipping

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

You are given a binary string of length n. You have **exactly** k moves. In one move, you must select a single bit. The state of all bits **except** that bit will get flipped (0 becomes 1, 1 becomes 0). You need to output the lexicographically largest string that you can get after using **all** k moves. Also, output the number of times you will select each bit. If there are multiple ways to do this, you may output any of them.

A binary string a is lexicographically larger than a binary string b of the same length, if and only if the following holds:

- in the first position where a and b differ, the string a contains a 1, and the string b contains a 0.

Input
The first line contains a single integer t ($1 \le t \le 1000$) — the number of test cases.

Each test case has two lines. The first line has two integers n and k ($1 \le n \le 2 \cdot 10^5$; $0 \le k \le 10^9$).

The second line has a binary string of length n, each character is either 0 or 1.

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

Output
For each test case, output two lines.

The first line should contain the lexicographically largest string you can obtain.

The second line should contain n integers $f_1, f_2, \ldots, f_n$, where f_i is the number of times the i-th bit is selected. The sum of all the integers **must be equal to** k.

Example

input
6 6 3 100001 6 4 100011 6 0 000000 6 1 111001 6 11 101100 6 12 001110
output
111110 1 0 0 2 0 0 111110 0 1 1 1 0 1 000000 0 0 0 0 0 0 100110 1 0 0 0 0 0 111111 1 2 1 3 0 4 111110 1 1 4 2 0 4

Note
Here is the explanation for the first testcase. Each step shows how the binary string changes in a move.

- Choose bit 1: $\underline{1}00001 \rightarrow \underline{1}11110$.
- Choose bit 4: $111\underline{1}10 \rightarrow 000\underline{1}01$.
- Choose bit 4: $000\underline{1}01 \rightarrow 111\underline{1}10$.

The final string is 111110 and this is the lexicographically largest string we can get.

C. Line Empire

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

You are an ambitious king who wants to be the Emperor of The Reals. But to do that, you must first become Emperor of The Integers.

Consider a number axis. The capital of your empire is initially at 0. There are n unconquered kingdoms at positions $0 < x_1 < x_2 < \ldots < x_n$. You want to conquer all other kingdoms.

There are two actions available to you:

- You can change the location of your capital (let its current position be c_1) to any other **conquered** kingdom (let its position be c_2

) at a cost of $a \cdot |c_1 - c_2|$.
- From the current capital (let its current position be c_1) you can conquer an unconquered kingdom (let its position be c_2) at a cost of $b \cdot |c_1 - c_2|$. You **cannot** conquer a kingdom if there is an unconquered kingdom between the target and your capital.

Note that you **cannot** place the capital at a point without a kingdom. In other words, at any point, your capital can only be at 0 or one of $x_1, x_2, \ldots, x_n$. Also note that conquering a kingdom does not change the position of your capital.

Find the minimum total cost to conquer all kingdoms. Your capital can be anywhere at the end.

Input
The first line contains a single integer t ($1 \le t \le 1000$) — the number of test cases. The description of each test case follows.

The first line of each test case contains 3 integers n, a, and b ($1 \le n \le 2 \cdot 10^5$; $1 \le a, b \le 10^5$).

The second line of each test case contains n integers $x_1, x_2, \ldots, x_n$ ($1 \le x_1 < x_2 < \ldots < x_n \le 10^8$).

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

Output
For each test case, output a single integer — the minimum cost to conquer all kingdoms.

Example

input
4 5 2 7 3 5 12 13 21 5 6 3 1 5 6 21 30 2 9 3 10 15 11 27182 31415 16 18 33 98 874 989 4848 20458 34365 38117 72030
output
173 171 75 3298918744

Note
Here is an optimal sequence of moves for the second test case:

1. Conquer the kingdom at position 1 with cost $3 \cdot (1 - 0) = 3$.
2. Move the capital to the kingdom at position 1 with cost $6 \cdot (1 - 0) = 6$.
3. Conquer the kingdom at position 5 with cost $3 \cdot (5 - 1) = 12$.
4. Move the capital to the kingdom at position 5 with cost $6 \cdot (5 - 1) = 24$.
5. Conquer the kingdom at position 6 with cost $3 \cdot (6 - 5) = 3$.
6. Conquer the kingdom at position 21 with cost $3 \cdot (21 - 5) = 48$.
7. Conquer the kingdom at position 30 with cost $3 \cdot (30 - 5) = 75$.

The total cost is $3 + 6 + 12 + 24 + 3 + 48 + 75 = 171$. You cannot get a lower cost than this.

D. Reverse Sort Sum

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

Suppose you had an array A of n elements, each of which is 0 or 1.

Let us define a function $f(k, A)$ which returns another array B, the result of sorting the first k elements of A in non-decreasing order. For example, $f(4, [0, 1, 1, 0, 0, 1, 0]) = [0, 0, 1, 1, 0, 1, 0]$. Note that the first 4 elements were sorted.

Now consider the arrays $B_1, B_2, \ldots, B_n$ generated by $f(1, A), f(2, A), \ldots, f(n, A)$. Let C be the array obtained by taking the element-wise sum of $B_1, B_2, \ldots, B_n$.

For example, let $A = [0, 1, 0, 1]$. Then we have $B_1 = [0, 1, 0, 1]$, $B_2 = [0, 1, 0, 1]$, $B_3 = [0, 0, 1, 1]$, $B_4 = [0, 0, 1, 1]$. Then $C = B_1 + B_2 + B_3 + B_4 = [0, 1, 0, 1] + [0, 1, 0, 1] + [0, 0, 1, 1] + [0, 0, 1, 1] = [0, 2, 2, 4]$.

You are given C. Determine a binary array A that would give C when processed as above. It is guaranteed that an array A exists for given C in the input.

Input
The first line contains a single integer t ($1 \le t \le 1000$) — the number of test cases.

Each test case has two lines. The first line contains a single integer n ($1 \le n \le 2 \cdot 10^5$).

The second line contains n integers $c_1, c_2, \ldots, c_n$ ($0 \le c_i \le n$). It is guaranteed that a valid array A exists for the given C.

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

Output

For each test case, output a single line containing n integers $a_1, a_2, \ldots, a_n$ (a_i is 0 or 1). If there are multiple answers, you may output any of them.

Example

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

Note

Here's the explanation for the first test case. Given that $A = [1, 1, 0, 1]$, we can construct each B_i:

- $B_1 = [1, 1, 0, 1]$;
- $B_2 = [1, 1, 0, 1]$;
- $B_3 = [0, 1, 1, 1]$;
- $B_4 = [0, 1, 1, 1]$

And then, we can sum up each column above to get
$C = [1 + 1 + 0 + 0, 1 + 1 + 1 + 1, 0 + 0 + 1 + 1, 1 + 1 + 1 + 1] = [2, 4, 2, 4]$.

E. AND-MEX Walk

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

There is an undirected, connected graph with n vertices and m weighted edges. A *walk* from vertex u to vertex v is defined as a sequence of vertices $p_1, p_2, \ldots, p_k$ (which are not necessarily distinct) starting with u and ending with v, such that p_i and p_{i+1} are connected by an edge for $1 \le i < k$.

We define the *length* of a walk as follows: take the ordered sequence of edges and write down the weights on each of them in an array. Now, write down the bitwise AND of every nonempty prefix of this array. The *length* of the walk is the MEX of all these values.

More formally, let us have $[w_1, w_2, \ldots, w_{k-1}]$ where w_i is the weight of the edge between p_i and p_{i+1}. Then the *length* of the walk is given by $\text{MEX}(\{w_1,\ w_1 \& w_2,\ \ldots,\ w_1 \& w_2 \& \ldots \& w_{k-1}\})$, where $\&$ denotes the bitwise AND operation.

Now you must process q queries of the form u v. For each query, find the **minimum** possible length of a walk from u to v.

The MEX (minimum excluded) of a set is the smallest non-negative integer that does not belong to the set. For instance:

- The MEX of $\{2, 1\}$ is 0, because 0 does not belong to the set.
- The MEX of $\{3, 1, 0\}$ is 2, because 0 and 1 belong to the set, but 2 does not.
- The MEX of $\{0, 3, 1, 2\}$ is 4 because 0, 1, 2 and 3 belong to the set, but 4 does not.

Input

The first line contains two integers n and m ($2 \le n \le 10^5$; $n - 1 \le m \le \min\left(\frac{n(n-1)}{2}, 10^5\right)$).

Each of the next m lines contains three integers a, b, and w ($1 \le a, b \le n$, $a \ne b$; $0 \le w < 2^{30}$) indicating an undirected edge between vertex a and vertex b with weight w. The input will not contain self-loops or duplicate edges, and the provided graph will be connected.

The next line contains a single integer q ($1 \le q \le 10^5$).

Each of the next q lines contains two integers u and v ($1 \le u, v \le n$, $u \ne v$), the description of each query.

Output

For each query, print one line containing a single integer — the answer to the query.

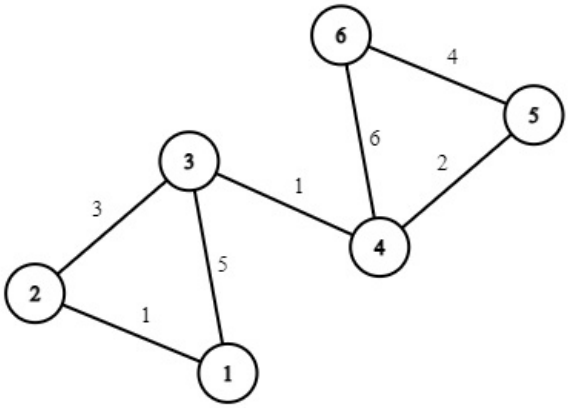
Examples

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

input
9 8 1 2 5 2 3 11 3 4 10 3 5 10 5 6 2 5 7 1 7 8 5 7 9 5 10 5 7 2 5 7 1 6 4 5 2 7 6 4 1 6 2 4 7 2 8
output
0 0 2 0 0 2 1 0 1 1

Note

The following is an explanation of the first example.



The graph in the first example.

Here is one possible walk for the first query:

$$1 \xrightarrow{5} 3 \xrightarrow{3} 2 \xrightarrow{1} 1 \xrightarrow{5} 3 \xrightarrow{1} 4 \xrightarrow{2} 5.$$

The array of weights is $w = [5, 3, 1, 5, 1, 2]$. Now if we take the bitwise AND of every prefix of this array, we get the set $\{5, 1, 0\}$. The MEX of this set is 2. We cannot get a walk with a smaller length (as defined in the statement).

F. Tree and Permutation Game

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

There is a tree of n vertices and a permutation p of size n. A token is present on vertex x of the tree.

Alice and Bob are playing a game. Alice is in control of the permutation p, and Bob is in control of the token on the tree. In Alice's turn, she **must** pick two **distinct numbers** u and v (**not** positions; $u \neq v$), such that the token is neither at vertex u nor vertex v on the tree, and swap their positions in the permutation p. In Bob's turn, he **must** move the token to an adjacent vertex from the one it is currently on.

Alice wants to sort the permutation in increasing order. Bob wants to prevent that. Alice wins if the permutation is sorted in increasing order at the beginning or end of her turn. Bob wins if he can make the game go on for an infinite number of moves (which means that Alice is never able to get a sorted permutation). Both players play optimally. Alice makes the first move.

Given the tree, the permutation p, and the vertex x on which the token initially is, find the winner of the game.

Input

The first line contains a single integer t ($1 \leq t \leq 1000$) — the number of test cases. The description of each test case follows.

The first line of each test case has two integers n and x ($3 \leq n \leq 2 \cdot 10^5$; $1 \leq x \leq n$).

Each of the next $n - 1$ lines contains two integers a and b ($1 \leq a, b \leq n$, $a \neq b$) indicating an undirected edge between vertex a and vertex b. It is guaranteed that the given edges form a tree.

The next line contains n integers $p_1, p_2, \ldots, p_n$ ($1 \leq p_i \leq n$) — the permutation p.

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

Output

For each test case, output one line containing `Alice` or `Bob` — the winner of the game. The output is case-sensitive.

Examples

input
3 6 3 1 3

3 2
4 3
3 6
5 3
2 1 3 6 4 5
3 2
1 2
3 2
1 3 2
3 2
1 2
3 2
1 2 3

output

Alice
Bob
Alice

input

1
11 4
3 11
5 9
10 3
4 8
2 4
1 8
6 8
8 7
4 5
5 11
7 4 9 8 6 5 11 10 2 3 1

output

Alice

Note

Here is the explanation for the first example:

In the first test case, there is a way for Alice to win. Here is a possible sequence of moves:

1. Alice swaps 5 and 6, resulting in $[2, 1, 3, 5, 4, 6]$.
2. Bob moves the token to vertex 5.
3. Alice swaps 1 and 2, resulting in $[1, 2, 3, 5, 4, 6]$.
4. Bob moves the token to vertex 3.
5. Alice swaps 4 and 5, resulting in $[1, 2, 3, 4, 5, 6]$, and wins.

In the second test case, Alice cannot win since Bob can make the game go on forever. Here is the sequence of moves:

1. Alice can only swap 1 and 3, resulting in $[3, 1, 2]$.
2. Bob moves the token to vertex 1.
3. Alice can only swap 2 and 3, resulting in $[2, 1, 3]$.
4. Bob moves the token to vertex 2.
5. Alice can only swap 1 and 3, resulting in $[2, 3, 1]$.
6. Bob moves the token to vertex 3.
7. Alice can only swap 1 and 2, resulting in $[1, 3, 2]$.
8. Bob moves the token to vertex 2.

And then the sequence repeats forever.
In the third test case, Alice wins immediately since the permutation is already sorted.