Codeforces Round #793 (Div. 2)

A. Palindromic Indices

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

You are given a **palindromic** string s of length n.

You have to count the number of indices i $(1 \le i \le n)$ such that the string after removing s_i from s still remains a palindrome.

For example, consider s = "aba"

1. If we remove s_1 from s, the string becomes "ba" which is not a palindrome.
2. If we remove s_2 from s, the string becomes "aa" which is a palindrome.
3. If we remove s_3 from s, the string becomes "ab" which is not a palindrome.

A palindrome is a string that reads the same backward as forward. For example, "abba", "a", "fef" are palindromes whereas "codeforces", "acd", "xy" are not.

Input

The input consists of multiple test cases. The first line of the input contains a single integer t $(1 \le t \le 10^3)$ — the number of test cases. Description of the test cases follows.

The first line of each testcase contains a single integer n $(2 \le n \le 10^5)$ — the length of string s.

The second line of each test case contains a string s consisting of lowercase English letters. **It is guaranteed that s is a palindrome**.

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

Output

For each test case, output a single integer — the number of indices i $(1 \le i \le n)$ such that the string after removing s_i from s still remains a palindrome.

Example

input
3 3 aba 8 acaaaaca 2 dd
output
1 4 2

Note

The first test case is described in the statement.

In the second test case, the indices i that result in palindrome after removing s_i are $3, 4, 5, 6$. Hence the answer is 4.

In the third test case, removal of any of the indices results in "d" which is a palindrome. Hence the answer is 2.

B. AND Sorting

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

You are given a permutation p of integers from 0 to $n-1$ (each of them occurs exactly once). Initially, the permutation is **not sorted** (that is, $p_i > p_{i+1}$ for at least one $1 \le i \le n-1$).

The permutation is called X-sortable for some non-negative integer X if it is possible to sort the permutation by performing the operation below some finite number of times:

- Choose two indices i and j $(1 \le i < j \le n)$ such that $p_i \& p_j = X$.
- Swap p_i and p_j.

Here $\&$ denotes the bitwise AND operation.

Find the **maximum** value of X such that p is X-sortable. It can be shown that there always exists some value of X such that p is X-sortable.

Input

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

The first line of each test case contains a single integer n $(2 \le n \le 2 \cdot 10^5)$ — the length of the permutation.

The second line of each test case contains n integers $p_1, p_2, \ldots, p_n$ $(0 \le p_i \le n-1$, all p_i are distinct$)$ — the elements of p. It is guaranteed that p is not sorted.

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

Output

For each test case output a single integer — the maximum value of X such that p is X-sortable.

Example

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

Note

In the first test case, the only X for which the permutation is X-sortable are $X = 0$ and $X = 2$, maximum of which is 2.

Sorting using $X = 0$:

- Swap p_1 and p_4, $p = [2, 1, 3, 0]$.
- Swap p_3 and p_4, $p = [2, 1, 0, 3]$.
- Swap p_1 and p_3, $p = [0, 1, 2, 3]$.

Sorting using $X = 2$:

- Swap p_3 and p_4, $p = [0, 1, 2, 3]$.

In the second test case, we must swap p_1 and p_2 which is possible only with $X = 0$.

C. LIS or Reverse LIS?

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

You are given an array a of n positive integers.

Let $\mathrm{LIS}(a)$ denote the length of longest strictly increasing subsequence of a. For example,

- $\mathrm{LIS}([2, \underline{1}, 1, \underline{3}]) = 2$.
- $\mathrm{LIS}([\underline{3}, \underline{5}, \underline{10}, \underline{20}]) = 4$.
- $\mathrm{LIS}([3, \underline{1}, \underline{2}, \underline{4}]) = 3$.

We define array a' as the array obtained after reversing the array a i.e. $a' = [a_n, a_{n-1}, \ldots, a_1]$.

The beauty of array a is defined as $min(\mathrm{LIS}(a), \mathrm{LIS}(a'))$.

Your task is to determine the maximum possible beauty of the array a if you can rearrange the array a arbitrarily.

Input

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

The first line of each test case contains a single integer n $(1 \le n \le 2 \cdot 10^5)$ — the length of array a.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ $(1 \le a_i \le 10^9)$ — the elements of the array a.

It is guaranteed that 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 maximum possible beauty of a after rearranging its elements arbitrarily.

Example

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

Note

In the first test case, $a = [6, 6, 6]$ and $a' = [6, 6, 6]$. $\mathrm{LIS}(a) = \mathrm{LIS}(a') = 1$. Hence the beauty is $min(1, 1) = 1$.

In the second test case, a can be rearranged to $[2, 5, 4, 5, 4, 2]$. Then $a' = [2, 4, 5, 4, 5, 2]$. $\mathrm{LIS}(a) = \mathrm{LIS}(a') = 3$. Hence the beauty is 3 and it can be shown that this is the maximum possible beauty.

In the third test case, a can be rearranged to $[1, 2, 3, 2]$. Then $a' = [2, 3, 2, 1]$. $\mathrm{LIS}(a) = 3$, $\mathrm{LIS}(a') = 2$. Hence the beauty is $min(3, 2) = 2$ and it can be shown that 2 is the maximum possible beauty.

D. Circular Spanning Tree

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

There are n nodes arranged in a circle numbered from 1 to n in the clockwise order. You are also given a binary string s of length n.

Your task is to construct a tree on the given n nodes satisfying the two conditions below or report that there such tree does not exist:

- For each node i $(1 \le i \le n)$, the degree of node is even if $s_i = 0$ and odd if $s_i = 1$.
- No two edges of the tree intersect internally in the circle. The edges are allowed to intersect on the circumference.

Note that all edges are drawn as straight line segments. For example, edge (u, v) in the tree is drawn as a line segment connecting u and v on the circle.

A tree on n nodes is a connected graph with $n - 1$ edges.

Input

The input consists of multiple test cases. The first line contains a single integer t $(1 \le t \le 2 \cdot 10^4)$ — the number of test cases. Description of the test cases follows.

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

The second line of each test case contains a binary string s of length n.

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

Output

For each test case, if there does not exist a tree that satisfies the given conditions, then output "NO" (without quotes), otherwise output "YES" followed by the description of tree.

You can output each letter in any case (for example, "YES", "Yes", "yes", "yEs", "yEs" will be recognized as a positive answer).

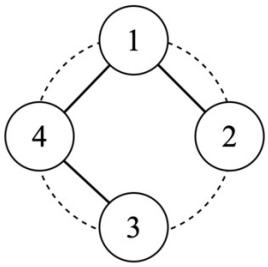
If there exists a tree, then output $n - 1$ lines, each containing two integers u and v $(1 \le u, v \le n, u \ne v)$ denoting an edge between u and v in the tree. If there are multiple possible answers, output any.

Example

input
3 4 0110 2 10 6 110110
output
YES 2 1 3 4

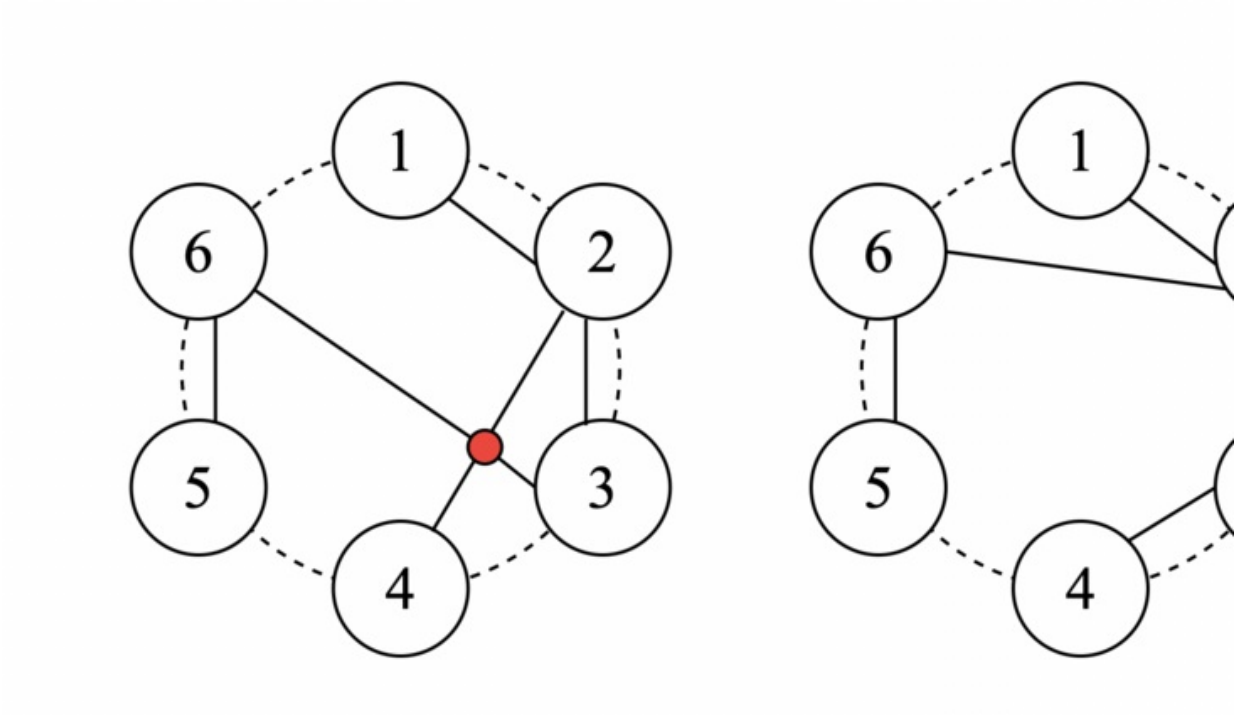
```
1 4
NO
YES
2 3
1 2
5 6
6 2
3 4
```

Note
In the first test case, the tree looks as follows:



In the second test case, there is only one possible tree with an edge between 1 and 2, and it does not satisfy the degree constraints.

In the third test case,



The tree on the left satisfies the degree constraints but the edges intersect internally, therefore it is not a valid tree, while the tree on the right is valid.

E. Unordered Swaps

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

Alice had a permutation p of numbers from 1 to n. Alice can swap a pair (x, y) which means swapping elements at positions x and y in p (i.e. swap p_x and p_y). Alice recently learned her first sorting algorithm, so she decided to sort her permutation in the **minimum** number of swaps possible. She wrote down all the swaps in the order in which she performed them to sort the permutation on a piece of paper.

For example,

- $[(2, 3), (1, 3)]$ is a valid swap sequence by Alice for permutation $p = [3, 1, 2]$ whereas $[(1, 3), (2, 3)]$ is not because it doesn't sort the permutation. Note that we cannot sort the permutation in less than 2 swaps.
- $[(1, 2), (2, 3), (2, 4), (2, 3)]$ cannot be a sequence of swaps by Alice for $p = [2, 1, 4, 3]$ even if it sorts the permutation because p can be sorted in 2 swaps, for example using the sequence $[(4, 3), (1, 2)]$.

Unfortunately, Bob shuffled the sequence of swaps written by Alice.

You are given Alice's permutation p and the swaps performed by Alice in arbitrary order. Can you restore the correct sequence of swaps that sorts the permutation p? Since Alice wrote correct swaps before Bob shuffled them up, it is guaranteed that there exists some order of swaps that sorts the permutation.

Input
The first line contains 2 integers n and m $(2 \le n \le 2 \cdot 10^5, 1 \le m \le n - 1)$ — the size of permutation and the minimum number of swaps required to sort the permutation.

The next line contains n integers $p_1, p_2, \ldots, p_n$ ($1 \le p_i \le n$, all p_i are distinct) — the elements of p. It is guaranteed that p forms a permutation.

Then m lines follow. The i-th of the next m lines contains two integers x_i and y_i — the i-th swap (x_i, y_i).

It is guaranteed that it is possible to sort p with these m swaps and that there is no way to sort p with less than m swaps.

Output
Print a permutation of m integers — a valid order of swaps written by Alice that sorts the permutation p. See sample explanation for better understanding.

In case of multiple possible answers, output any.

Examples

input
4 3

2 3 4 1
1 4
2 1
1 3

output

2 3 1

input

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

output

4 1 3 2

Note

In the first example, $p = [2, 3, 4, 1]$, $m = 3$ and given swaps are $[(1, 4), (2, 1), (1, 3)]$.

There is only one correct order of swaps i.e $[2, 3, 1]$.

1. First we perform the swap 2 from the input i.e $(2, 1)$, p becomes $[3, 2, 4, 1]$.
2. Then we perform the swap 3 from the input i.e $(1, 3)$, p becomes $[4, 2, 3, 1]$.
3. Finally we perform the swap 1 from the input i.e $(1, 4)$ and p becomes $[1, 2, 3, 4]$ which is sorted.

In the second example, $p = [6, 5, 1, 3, 2, 4]$, $m = 4$ and the given swaps are $[(3, 1), (2, 5), (6, 3), (6, 4)]$.

One possible correct order of swaps is $[4, 2, 1, 3]$.

1. Perform the swap 4 from the input i.e $(6, 4)$, p becomes $[6, 5, 1, 4, 2, 3]$.
2. Perform the swap 2 from the input i.e $(2, 5)$, p becomes $[6, 2, 1, 4, 5, 3]$.
3. Perform the swap 1 from the input i.e $(3, 1)$, p becomes $[1, 2, 6, 4, 5, 3]$.
4. Perform the swap 3 from the input i.e $(6, 3)$ and p becomes $[1, 2, 3, 4, 5, 6]$ which is sorted.

There can be other possible answers such as $[1, 2, 4, 3]$.

F. MCMF?

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

You are given two integer arrays a and b ($b_i \neq 0$ and $|b_i| \leq 10^9$). Array a is sorted in **non-decreasing** order.

The cost of a subarray $a[l : r]$ is defined as follows:

- If $\sum_{j=l}^{r} b_j \neq 0$, then the cost is not defined.
- Otherwise:
  - Construct a bipartite flow graph with $r - l + 1$ vertices, labeled from l to r, with all vertices having $b_i < 0$ on the left and those with $b_i > 0$ on right. For each i, j such that $l \leq i, j \leq r$, $b_i < 0$ and $b_j > 0$, draw an edge from i to j with infinite capacity and cost of unit flow as $|a_i - a_j|$.
  - Add two more vertices: source S and sink T.
  - For each i such that $l \leq i \leq r$ and $b_i < 0$, add an edge from S to i with cost 0 and capacity $|b_i|$.
  - For each i such that $l \leq i \leq r$ and $b_i > 0$, add an edge from i to T with cost 0 and capacity $|b_i|$.
  - The cost of the subarray is then defined as the minimum cost of maximum flow from S to T.

You are given q queries in the form of two integers l and r. You have to compute the cost of subarray $a[l : r]$ for each query, modulo $10^9 + 7$.

If you don't know what the minimum cost of maximum flow means, read here.

Input

The first line of input contains two integers n and q ($2 \leq n \leq 2 \cdot 10^5, 1 \leq q \leq 2 \cdot 10^5$) — length of arrays a, b and the number of queries.

The next line contains n integers $a_1, a_2 \ldots a_n$ ($0 \leq a_1 \leq a_2 \ldots \leq a_n \leq 10^9$) — the array a. It is guaranteed that a is sorted in **non-decreasing** order.

The next line contains n integers $b_1, b_2 \ldots b_n$ ($-10^9 \leq b_i \leq 10^9, b_i \neq 0$) — the array b.

The i-th of the next q lines contains two integers l_i, r_i ($1 \leq l_i \leq r_i \leq n$). It is guaranteed that $\sum_{j=l_i}^{r_i} b_j = 0$.

Output

For each query l_i, r_i — print the cost of subarray $a[l_i : r_i]$ modulo $10^9 + 7$.

Example

input

8 4
1 2 4 5 9 10 10 13
6 -1 1 -3 2 1 -1 1
2 3
6 7
3 5
2 6

output

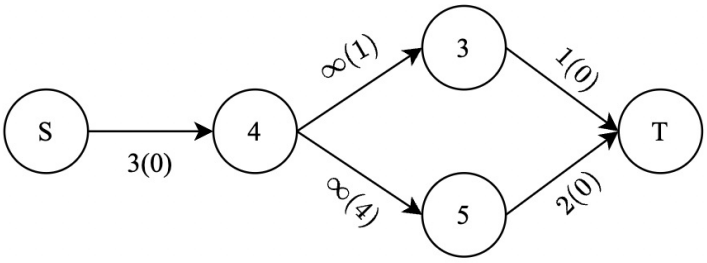
2
0
9
15

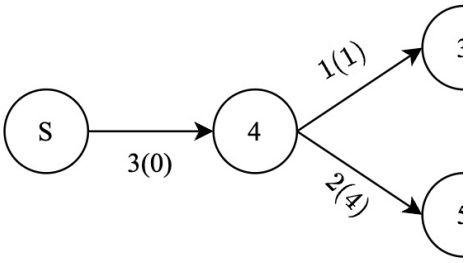
Note

In the **first query**, the maximum possible flow is 1 i.e one unit from source to 2, then one unit from 2 to 3, then one unit from 3 to sink. The cost of the flow is $0 \cdot 1 + |2 - 4| \cdot 1 + 0 \cdot 1 = 2$.

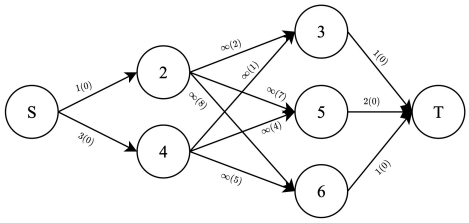
In the **second query**, the maximum possible flow is again 1 i.e from source to 7, 7 to 6, and 6 to sink with a cost of $0 \cdot |10 - 10| \cdot 1 + 0 \cdot 1 = 0$.

In the **third query**, the flow network is shown on the left with capacity written over the edge and the cost written in bracket. The image on the right shows the flow through each edge in an optimal configuration.

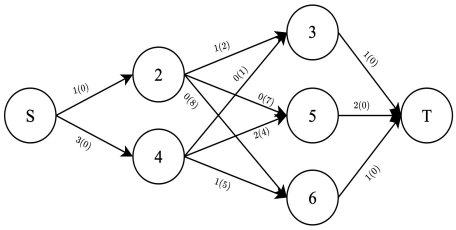




Maximum flow is 3 with a cost of $0 \cdot 3 + 1 \cdot 1 + 4 \cdot 2 + 0 \cdot 1 + 0 \cdot 2 = 9$.
In the **fourth query**, the flow network looks as –



The minimum cost maximum flow is achieved in the configuration –



The maximum flow in the above network is 4 and the minimum cost of such flow is 15.