A. The Fair Nut and the Best Path

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

The Fair Nut is going to travel to the Tree Country, in which there are n cities. Most of the land of this country is covered by forest. Furthermore, the local road system forms a tree (connected graph without cycles). Nut wants to rent a car in the city u and go by a simple path to city v. He hasn't determined the path, so it's time to do it. Note that chosen path can consist of only one vertex.

A filling station is located in every city. Because of strange law, Nut can buy only w_i liters of gasoline in the i-th city. We can assume, that he has **infinite money**. Each road has a length, and as soon as Nut drives through this road, the amount of gasoline decreases by length. Of course, Nut can't choose a path, which consists of roads, where he runs out of gasoline. He can buy gasoline in **every** visited city, even in **the first** and **the last**.

He also wants to find the maximum amount of gasoline that he can have at the end of the path. Help him: count it.

Input

The first line contains a single integer n ($1 \le n \le 3 \cdot 10^5$) — the number of cities.

The second line contains n integers $w_1, w_2, \ldots, w_n$ ($0 \le w_i \le 10^9$) — the maximum amounts of liters of gasoline that Nut can buy in cities.

Each of the next $n - 1$ lines describes road and contains three integers u, v, c ($1 \le u, v \le n$, $1 \le c \le 10^9$, $u \ne v$), where u and v — cities that are connected by this road and c — its length.

It is guaranteed that graph of road connectivity is a tree.

Output

Print one number — the maximum amount of gasoline that he can have at the end of the path.

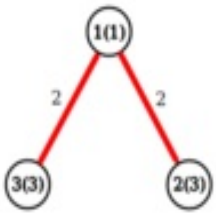
Examples

input
3 1 3 3 1 2 2 1 3 2
output
3

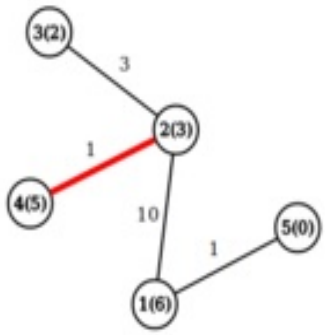
input
5 6 3 2 5 0 1 2 10 2 3 3 2 4 1 1 5 1
output
7

Note

The optimal way in the first example is $2 \to 1 \to 3$.



The optimal way in the second example is $2 \to 4$.



B. The Fair Nut and Strings

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

Recently, the Fair Nut has written k strings of length n, consisting of letters "a" and "b". He calculated c — the number of strings that are prefixes of at least one of the written strings. **Every string was counted only one time**.

Then, he lost his sheet with strings. He remembers that all written strings were lexicographically **not smaller** than string s and **not bigger** than string t. He is interested: what is the maximum value of c that he could get.

A string a is lexicographically smaller than a string b if and only if one of the following holds:

- a is a prefix of b, but $a \neq b$;
- in the first position where a and b differ, the string a has a letter that appears earlier in the alphabet than the corresponding letter in b.

Input

The first line contains two integers n and k ($1 \leq n \leq 5 \cdot 10^5$, $1 \leq k \leq 10^9$).

The second line contains a string s ($|s| = n$) — the string consisting of letters "a" and "b.

The third line contains a string t ($|t| = n$) — the string consisting of letters "a" and "b.

It is guaranteed that string s is lexicographically not bigger than t.

Output

Print one number — maximal value of c.

Examples

input
2 4 aa bb
output
6

input
3 3 aba bba
output
8

input
4 5 abbb baaa
output
8

Note

In the first example, Nut could write strings "aa", "ab", "ba", "bb". These 4 strings are prefixes of at least one of the written strings, as well as "a" and "b". Totally, 6 strings.

In the second example, Nut could write strings "aba", "baa", "bba".

In the third example, there are only two different strings that Nut could write. If both of them are written, $c = 8$.

C. Max Mex

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

Once Grisha found a tree (connected graph without cycles) with a root in node 1.

But this tree was not just a tree. A permutation p of integers from 0 to $n - 1$ is written in nodes, a number p_i is written in node i.

As Grisha likes to invent some strange and interesting problems for himself, but not always can solve them, you need to help him deal with two types of queries on this tree.

Let's define a function $MEX(S)$, where S is a set of non-negative integers, as a smallest non-negative integer that is not included in this set.

Let l be a simple path in this tree. So let's define indices of nodes which lie on l as u_1, u_2, ..., u_k.

Define $V(l)$ as a set $\{p_{u_1}, p_{u_2}, \ldots, p_{u_k}\}$.

Then queries are:

1. For two nodes i and j, swap p_i and p_j.
2. Find the maximum value of $MEX(V(l))$ in all possible l.

Input

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

The second line contains n integers — p_1, p_2, ..., p_n ($0 \le p_i < n$) — the permutation p, it's guaranteed that all numbers are different .

The third line contains $n - 1$ integers — d_2, d_3, ..., d_n ($1 \le d_i < i$), where d_i is a direct ancestor of node i in a tree.

The fourth line contains a single integer q ($1 \le q \le 2 \cdot 10^5$) — the number of queries.

The following q lines contain the description of queries:

At the beginning of each of next q lines, there is a single integer t (1 or 2) — the type of a query:

1. If $t = 1$, the line also contains two integers i and j ($1 \le i, j \le n$) — the indices of nodes, where values of the permutation should be swapped.
2. If $t = 2$, you need to find the maximum value of $MEX(V(l))$ in all possible l.

Output

For each type 2 query print a single integer — the answer for this query.

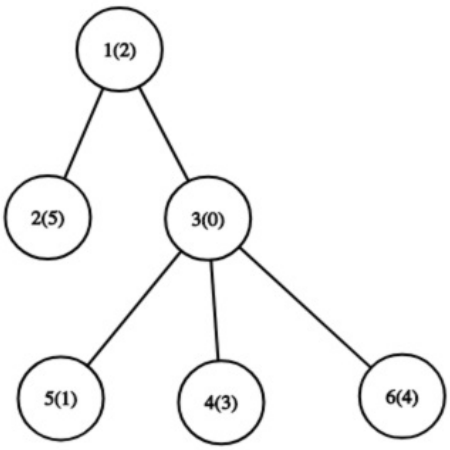
Examples

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

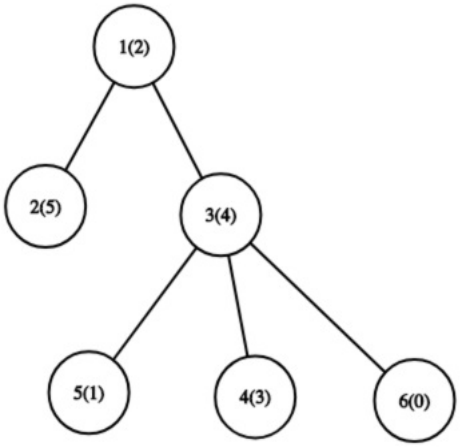
input
6 5 2 1 4 3 0 1 1 1 3 3 9 2 1 5 3 2 1 6 1 2 1 4 2 2 1 1 6 2
output
3 2 4 4 2

Note

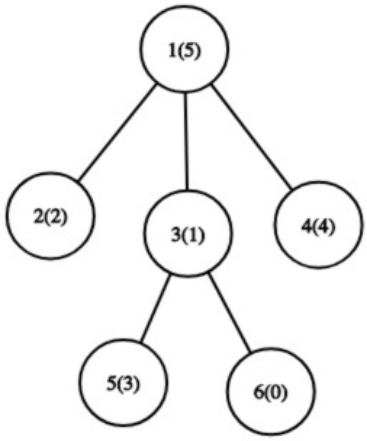
Number written in brackets is a permutation value of a node.



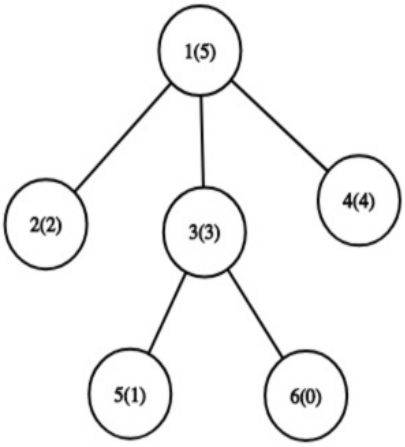
In the first example, for the first query, optimal path is a path from node 1 to node 5. For it, set of values is $\{0, 1, 2\}$ and MEX is 3.



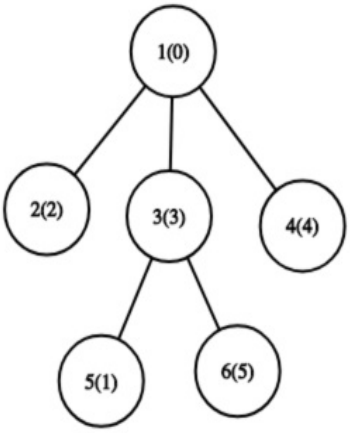
For the third query, optimal path is a path from node 5 to node 6. For it, set of values is $\{0, 1, 4\}$ and MEX is 2.



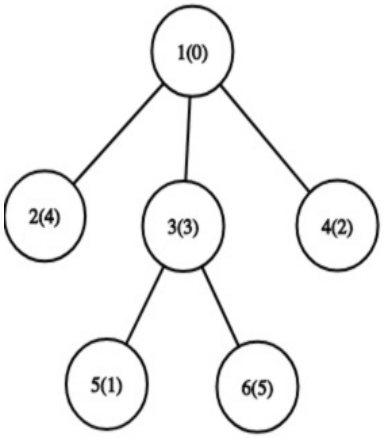
In the second example, for the first query, optimal path is a path from node 2 to node 6. For it, set of values is $\{0, 1, 2, 5\}$ and MEX is 3.



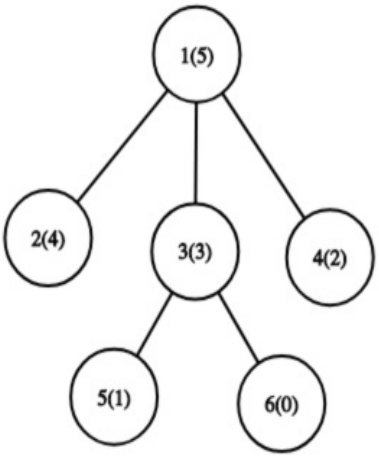
For the third query, optimal path is a path from node 5 to node 6. For it, set of values is $\{0, 1, 3\}$ and MEX is 2.



For the fifth query, optimal path is a path from node 5 to node 2. For it, set of values is $\{0, 1, 2, 3\}$ and MEX is 4.



For the seventh query, optimal path is a path from node 5 to node 4. For it, set of values is $\{0, 1, 2, 3\}$ and MEX is 4.



For the ninth query, optimal path is a path from node 6 to node 5. For it, set of values is $\{0, 1, 3\}$ and MEX is 2.

D. The Fair Nut's getting crazy

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

The Fair Nut has found an array a of n integers. We call subarray $l \dots r$ a sequence of consecutive elements of an array with indexes from l to r, i.e. $a_l, a_{l+1}, a_{l+2}, \dots, a_{r-1}, a_r$.

No one knows the reason, but he calls a pair of subsegments good if and only if the following conditions are satisfied:

1. These subsegments should not be nested. That is, each of the subsegments should contain an element (as an index) that does not belong to another subsegment.
2. Subsegments intersect and each element that belongs to the intersection belongs each of segments only once.

For example $a = [1, 2, 3, 5, 5]$. Pairs $(1 \dots 3; 2 \dots 5)$ and $(1 \dots 2; 2 \dots 3))$ — are good, but $(1 \dots 3; 2 \dots 3)$ and $(3 \dots 4; 4 \dots 5)$ — are not (subsegment $1 \dots 3$ contains subsegment $2 \dots 3$, integer 5 belongs both segments, but occurs twice in subsegment $4 \dots 5$).

Help the Fair Nut to find out the number of pairs of good subsegments! The answer can be rather big so print it modulo $10^9 + 7$.

Input

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

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($-10^9 \le a_i \le 10^9$) — the array elements.

Output
Print single integer — the number of pairs of good subsegments modulo $10^9 + 7$.

Examples

input
3 1 2 3
output
1

input
5 1 2 1 2 3
output
4

Note
In the first example, there is only one pair of good subsegments: $(1 \ldots 2, 2 \ldots 3)$.

In the second example, there are four pairs of good subsegments:

- $(1 \ldots 2, 2 \ldots 3)$
- $(2 \ldots 3, 3 \ldots 4)$
- $(2 \ldots 3, 3 \ldots 5)$
- $(3 \ldots 4, 4 \ldots 5)$

E. The Fair Nut and Rectangles

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

The Fair Nut got stacked in planar world. He should solve this task to get out.

You are given n rectangles with vertexes in $(0, 0)$, $(x_i, 0)$, (x_i, y_i), $(0, y_i)$. For each rectangle, you are also given a number a_i. Choose some of them that the area of union minus sum of a_i of the chosen ones is maximum.

It is guaranteed that there are no nested rectangles.

Nut has no idea how to find the answer, so he asked for your help.

Input
The first line contains one integer n ($1 \le n \le 10^6$) — the number of rectangles.

Each of the next n lines contains three integers x_i, y_i and a_i ($1 \le x_i, y_i \le 10^9$, $0 \le a_i \le x_i \cdot y_i$).

It is guaranteed that there are no nested rectangles.

Output
In a single line print the answer to the problem — the maximum value which you can achieve.

Examples

input
3 4 4 8 1 5 0 5 2 10
output
9

input
4 6 2 4 1 6 2 2 4 3 5 3 8
output
10

Note
In the first example, the right answer can be achieved by choosing the first and the second rectangles.

In the second example, the right answer can also be achieved by choosing the first and the second rectangles.

F. The Fair Nut and Amusing Xor

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

The Fair Nut has two arrays a and b, consisting of n numbers. He found them so long ago that no one knows when they came to him.

The Fair Nut often changes numbers in his arrays. He also is interested in how similar a and b are after every modification.

Let's denote similarity of two arrays as the minimum number of operations to apply to make arrays equal (every operation can be applied for both arrays). If it is impossible, similarity will be equal -1.

Per one operation you can choose a subarray with length k (k is fixed), and change every element a_i, which belongs to the chosen subarray, to $a_i \oplus x$ (x can be chosen), where $\oplus$ denotes the bitwise XOR operation.

Nut has already calculated the similarity of the arrays after every modification. Can you do it?

Note that you just need to calculate those values, that is you do not need to apply any operations.

Input
The first line contains three numbers n, k and q ($1 \leq k \leq n \leq 2 \cdot 10^5$, $0 \leq q \leq 2 \cdot 10^5$) — the length of the arrays, the length of the subarrays, to which the operations are applied, and the number of queries.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($0 \leq a_i < 2^{14}$) — elements of array a.

The third line contains n integers $b_1, b_2, \ldots, b_n$ ($0 \leq b_i < 2^{14}$) — elements of array b.

Each of the next q lines describes query and contains string s and two integers p and v ($1 \leq p \leq n$, $0 \leq v < 2^{14}$) — array, which this query changes («a» or «b» without quotes), index of changing element and its new value.

Output
On the first line print initial similarity of arrays a and b.

On the i-th of following q lines print similarity of a and b after applying first i modifications.

Examples

input
3 3 1 0 4 2 1 2 3 b 2 5
output
-1 1

input
3 2 2 1 3 2 0 0 0 a 1 0 b 1 1
output
2 -1 2

Note
In the first sample making arrays $[0, 4, 2]$ and $[1, 2, 3]$ is impossible with $k = 3$. After the modification, you can apply the operation with $x = 1$ to the whole first array (its length is equal to k), and it will be equal to the second array.

In order to make arrays equal in the second sample before changes, you can apply operations with $x = 1$ on subarray $[1, 2]$ of a and with $x = 2$ on subarray $[2, 3]$ of b.

After all queries arrays will be equal $[0, 3, 2]$ and $[1, 0, 0]$. The same operations make them equal $[1, 2, 2]$.