Building Improvements

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 megabytes

You have just been assigned to the CS Department's "Student Contentment Committee"and have been tasked with improving the comfort of Iribe (let us say nothing about CSIC).

The CS department is interested in adding more comfortable seating around Iribe to improve student happiness, but does not want to spend more money than necessary.

For this problem, let us model the building as a collection of areas (nodes) and one-way corridors between sitting areas (directed edges). Why one-way corridors? Well, just to make the problem more interesting...

Every area a_i has some cost c_i of building new seating in that area.

You learn from the building coordinator that if a student can get from an area a_i to another area a_j, and also get from a_j back to a_i, then there is no need to build seating at both a_i and a_j—it is sufficient to build seating at only one of the areas—and of course the cheaper option is better!

Your task is to determine the minimum total cost to ensure that every area either (1) has new seating built at its location, or (2) can reach (and can be reached) by an area with seating.

Input

The first line will contain an integer $1 \le n \le 10^5$ for the number of areas.

In the next line, n space-separated integers representing the costs c_i will be given where $\forall i, 0 \le c_i \le 10^9$.

The next line will contain an integer $0 \le m \le 10^6$ representing the number of one-way corridors. Each of the next m lines will contain two integers a_i and a_j representing the areas that the next corridor connects.

Note that there will be no corridors connecting an area to itself.

Output

Output the minimum possible cost required to ensure that each area (1) has new seating built at its location, or (2) can reach (and can be reached) by an area with seating.

Example

standard input	standard output
3 100 20 999 3 0 1 1 0 1 2	1019

Note

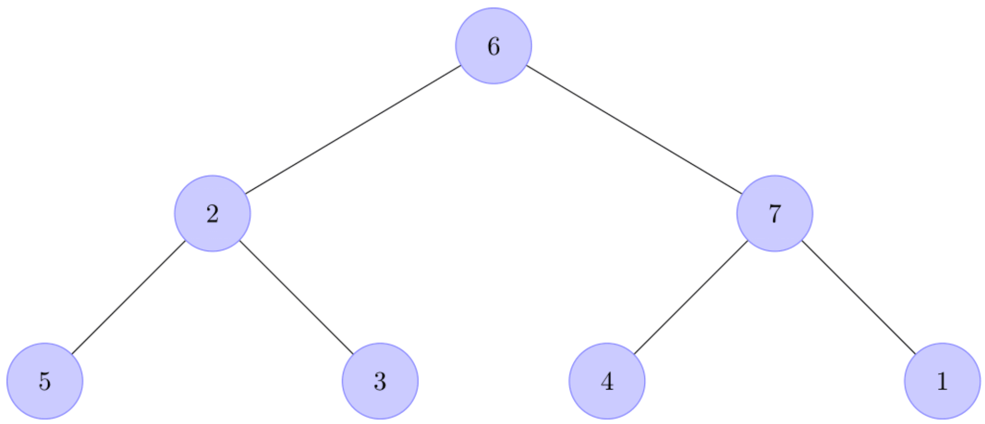
In the example, the first two areas can both reach each other, so only one seating location needs to be built. The cheaper option is to build on area 2 at a cost of 20. The third area also needs to have a seating location built, at a cost of 999. The total cost will be 1019.

Path Sums

Input file:	`standard input`
Output file:	`standard output`
Time limit:	0.5 seconds
Memory limit:	256 megabytes

You are given a rooted binary tree with n nodes (that is, each node has at most 2 children). Each node has a value attached to it. Given a query k, you are asked to count the number of simple paths whose values sum to exactly k.

For simplicity, we will consider a path to be valid in this context if it starts at some node and only moves down the tree. A path is not required to start at the root nor end at a leaf. For example, consider the following binary tree.



6-2-3 would be considered a valid path, as would **6-7**, **2-5**, and even **4**. However, **4-7-1** is not a valid path because it goes up and then down. Nor would any paths which double back on themselves like **6-7-1-7**. We are also not considering the empty path to be valid.

Input

The first line contains a single integer n, the number of nodes in the tree. Let's call the nodes $v_0, v_1, \ldots, v_{n-1}$. v_0 is guaranteed to be the root of the tree.

The next n lines $L_0, L_1, \ldots, L_{n-1}$ each contain three values c_i, l_i, and r_i. c_i is the value of v_i. l_i and r_i are the indices of the children of v_i. That is, v_i's left child is v_{l_i}, and its right child is v_{r_i}. If l_i or r_i is -1, then that means v_i has no left or right child, respectively.

The final line contains a single integer k, the sum you are querying for.

Output

A single integer representing the number of paths whose values sum to k.

Example

standard input	standard output
7 6 1 2 2 3 4 7 5 6 5 -1 -1 3 -1 -1 4 -1 -1 1 -1 -1 11	2

Note

$1 \leq n \leq 1,000$

$1 \leq k \leq 100,000$

$\forall i \in [0, n), -100 \leq c_i \leq 100$

The values c_i are not guaranteed to be unique.

For test cases 1–80, you may assume the tree is balanced.

For test cases 1–40, you may further assume that $n \leq 100$.

For test cases 81–100, you may make neither of these assumptions.

You should aim for a runtime of $O(n^2)$ or better.