C. Petya and Tree

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

One night, having had a hard day at work, Petya saw a nightmare. There was a binary search tree in the dream. But it was not the actual tree that scared Petya. The horrifying thing was that Petya couldn't search for elements in this tree. Petya tried many times to choose key and look for it in the tree, and each time he arrived at a wrong place. Petya has been racking his brains for long, choosing keys many times, but the result was no better. But the moment before Petya would start to despair, he had an epiphany: every time he was looking for keys, the tree didn't have the key, and occured exactly one mistake. "That's not a problem!", thought Petya. "Why not count the expectation value of an element, which is found when I search for the key". The moment he was about to do just that, however, Petya suddenly woke up.

Thus, you are given a binary search tree, that is a tree containing some number written in the node. This number is called the node key. The number of children of every node of the tree is equal either to 0 or to 2. The nodes that have 0 children are called leaves and the nodes that have 2 children, are called inner. An inner node has the left child, that is the child whose key is less than the current node's key, and the right child, whose key is more than the current node's key. Also, a key of any node is strictly larger than all the keys of the left subtree of the node and strictly smaller than all the keys of the right subtree of the node.

Also you are given a set of search keys, all of which are distinct and differ from the node keys contained in the tree. For each key from the set its search in the tree is realised. The search is arranged like this: initially we are located in the tree root, if the key of the current node is larger that our search key, then we move to the left child of the node, otherwise we go to the right child of the node and the process is repeated. As it is guaranteed that the search key is not contained in the tree, the search will always finish in some leaf. The key lying in the leaf is declared the search result.

It is known for sure that during the search we make a mistake in comparing exactly once, that is we go the wrong way, but we won't make any mistakes later. All possible mistakes are equiprobable, that is we should consider all such searches where exactly one mistake occurs. Your task is to find the expectation (the average value) of the search result for every search key, considering that exactly one mistake occurs in the search. That is, for a set of paths containing exactly one mistake in the given key search, you should count the average value of keys containing in the leaves of those paths.

Input

The first line contains an odd integer n ($3 \le n < 10^5$), which represents the number of tree nodes. Next n lines contain node descriptions. The $(i+1)$-th line contains two space-separated integers. The first number is the number of parent of the i-st node and the second number is the key lying in the i-th node. The next line contains an integer k ($1 \le k \le 10^5$), which represents the number of keys for which you should count the average value of search results containing one mistake. Next k lines contain the actual keys, one key per line.

All node keys and all search keys are positive integers, not exceeding 10^9. All $n + k$ keys are distinct.

All nodes are numbered from 1 to n. For the tree root "-1" (without the quote) will be given instead of the parent's node number. It is guaranteed that the correct binary search tree is given. For each node except for the root, it could be determined according to its key whether it is the left child or the right one.

Output

Print k real numbers which are the expectations of answers for the keys specified in the input. The answer should differ from the correct one with the measure of absolute or relative error not exceeding 10^{-9}.

Examples

input

```
7
-1 8
1 4
1 12
2 2
2 6
3 10
3 14
1
1
```

output

```
8.0000000000
```

input

```
3
-1 5
1 3
1 7
6
1
2
4
6
8
9
```

output

```
7.0000000000
7.0000000000
7.0000000000
3.0000000000
3.0000000000
3.0000000000
```

Note

In the first sample the search of key 1 with one error results in two paths in the trees: (1, 2, 5) and (1, 3, 6), in parentheses are listed numbers of nodes from the root to a leaf. The keys in the leaves of those paths are equal to 6 and 10 correspondingly, that's why the answer is equal to 8.