Help the problem setter

Preparing a problem for a programming contest takes a lot of time. Not only do you have to write the problem description and write a solution, but you also have to create difficult input files. In this problem, you get the chance to help the problem setter to create some input for a certain problem. For this purpose, let us select the problem which was not solved during last year's local contest. The problem was about finding the optimal binary search tree, given the probabilities that certain nodes are accessed. Your job will be: given the desired optimal binary search tree, find some access probabilities for which this binary search tree is the unique optimal binary search tree. Don't worry if you have not read last year's problem, all required definitions are provided in the following.
Let us define a **binary search tree** inductively as follows:

- The empty tree which has no node at all is a binary search tree
- Each non-empty binary search tree has a root, which is a node labelled with an integer, and two binary search trees as left and right subtree of the root
- A left subtree contains no node with a label ≥ than the label of the root
- A right subtree contains no node with a label ≤ than the label of the root

Given such a binary search tree, the following **search procedure** can be used to locate a node in the tree:
Start with the root node. Compare the label of the current node with the desired label. If it is the same, you have found the right node. Otherwise, if the desired label is smaller, search in the left subtree, otherwise search in the right subtree.
The **access cost** to locate a node is the number of nodes you have to visit until you find the right node. An **optimal binary search tree** is a binary search tree with the minimum expected access cost.

Input Specification

The input file contains several test cases.
Each test case starts with an integer n ($1 \le n \le 50$), the number of nodes in the optimal binary search tree. For simplicity, the labels of the nodes will be integers from 1 to n. The following n lines describe the structure of the tree. The i-th line contains the labels of the roots of the left and right subtree of the node with label i (or -1 for an empty subtree). You can assume that the input always defines a valid binary search tree.
The last test case is followed by a zero.

Output Specification

For each test case, write one line containing the access frequency for each node in increasing order of the labels of the nodes. To avoid problems with floating point precision, the frequencies should be written as integers, meaning the access probability for a node will be the frequency divided by the sum of all frequencies. Make sure that you do not write any integer bigger than $2^{63} - 1$ (the maximum value fitting in the C/C++ data type *long long* or the Java data type *long*).
Otherwise, you may produce any solution ensuring that there is exactly one optimal binary search tree: the binary search tree given in the input.

Sample Input

```
3
-1 -1
1 3
-1 -1
10
-1 2
-1 3
-1 4
-1 5
-1 6
-1 7
-1 8
-1 9
-1 10
-1 -1
0
```

Sample Output

```
1 1 1
512 256 128 64 32 16 8 4 2 1
```

Note that the first test case in the sample input describes a tree looking like

```
  2
 / \
1   3
```