# Comparing Trees

You work on a submarine and have just received an updated version the submarine's manual, which is several gigabytes in size. The captain doesn't want the crew to waste time leafing through the new version trying to figure out how it differs from the one they read in training. On the other hand, she would hate to accidentally launch a missile when trying to turn on the microwave. As the grunt on board, you're assigned the task of comparing the old and new versions and marking how they differ from each other. For this task, the captain graciously provides you with a highlighter and lots of coffee.

You have a better idea. Since both versions are available in electronic form, you think of writing a program that would compare them automatically. The manual is organized hierarchically in volumes, chapters, sections, subsections, paragraphs, etc., so you model it using a tree (described below). All you need to do now is write a program that takes two trees and outputs a description of how they differ.

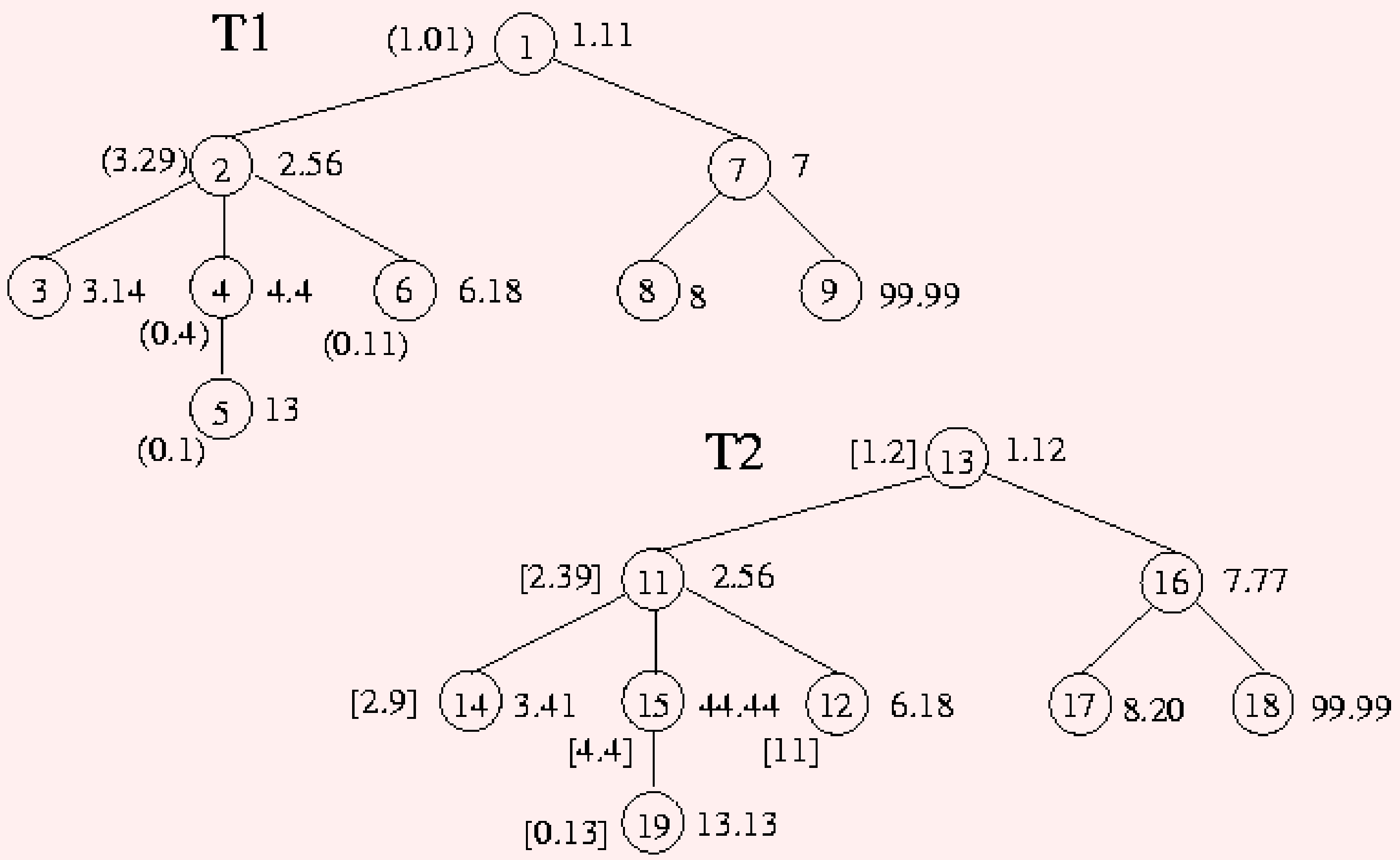

**Figure 1:** Comparing Trees

## Trees

We use the term *tree* to mean a *rooted, ordered, labeled tree*, such as those depicted in Figure 1. (Computer scientists like to draw their trees upside-down.) Each tree has a unique ``top'' node called its *root*. In the figure, the node numbered 1 is the root of tree $T_1$ . (The node number is used to identify a node and is indicated within the circle representing the node.) Each node in a tree also has a *label*, which is indicated next to that node in the figure. For this problem, we assume that labels are real numbers. For example, the label of node 4 is 4.40. Each node except the root also has a unique *parent* node. In the figure, we connect a node to its parent (depicted above the node) using a line. For example, the parent of node 7 is node 1. A node is called it's parent's *child*. Nodes that do not have any children are called *leaf nodes*; the rest are called *interior nodes*. The children of each interior node are ordered, and the figure depicts the children of each node left-to-right in this order. For example, the children of node 2, in order, are 3, 4, and 6.

All the nodes that lie on the path (of length zero or more) from a node to the root are called that node's *ancestors*. Note that every node is its own ancestor. If a node $n_1$ is the ancestor of node $n_2$ , then $n_2$ is called a *descendant* of $n_1$ . The tree consisting of all the descendants of of a node *n* is called its *subtree*. The length of the path from a node to the root is called that node's *depth*. Thus, the depth of the root is always 0. In our example, the depth of node 4 is 2.

The *preorder list* of a tree consists of the root followed by the preorder list of the subtrees rooted at its children, in order. For example, the preorder list of tree $T_1$ is $(1, 2, \ldots, 9)$ . The preorder list of tree $T_2$ is (13, 11, 14, 15, 19, 12, 16, 17, 18).

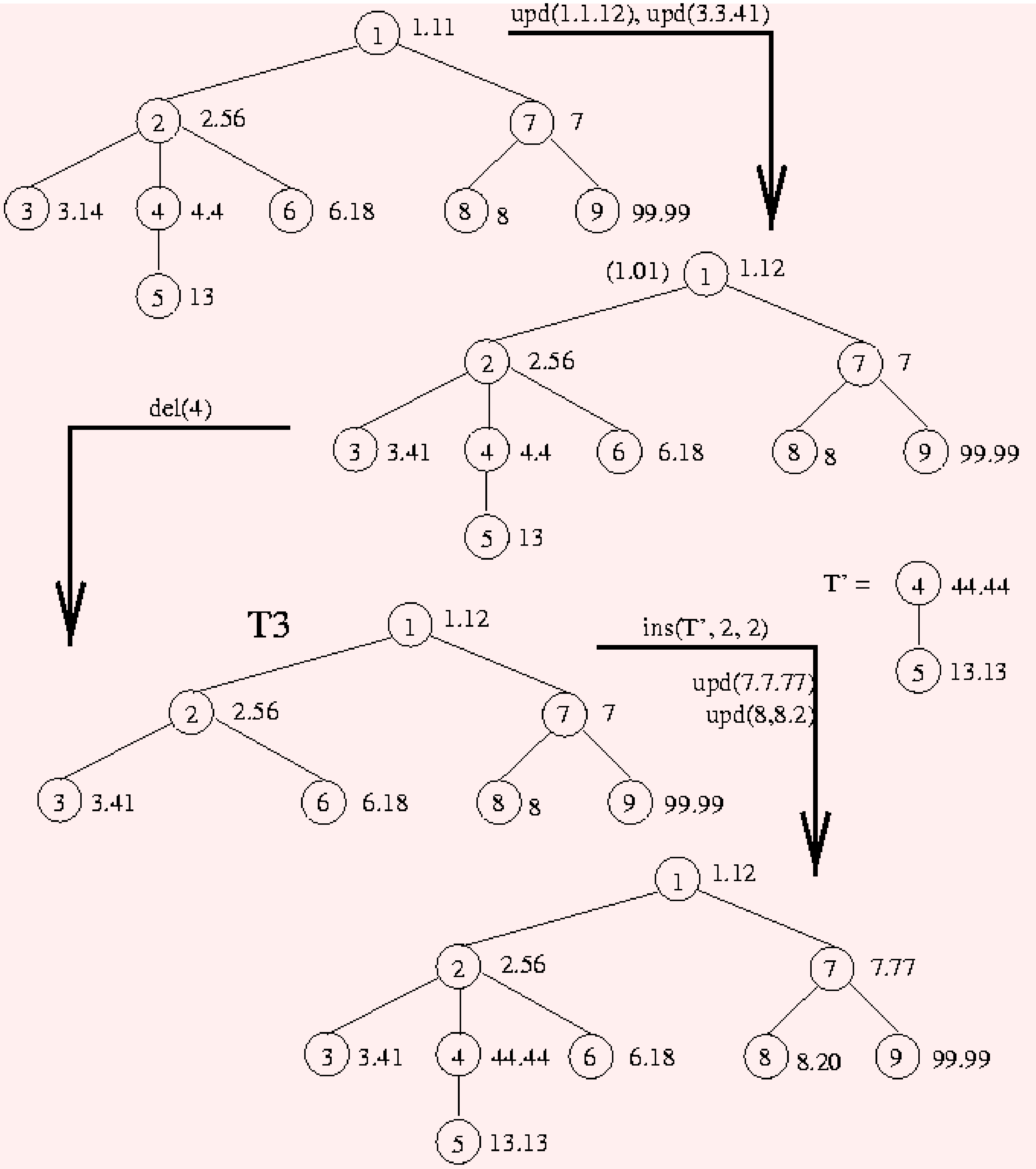

**Figure 2:** Transforming a tree using an edit script

## Editing Trees

Given such a tree, we can *edit* (modify) it using the following three kinds of *edit operations*:

- An *update operation* can be used to change the label of a node. For example, we can change the

label of node 3 to 3.41 by using the operation $upd(3, 3.41)$. That is, the operation $upd(n, l)$ changes the label of node *n* to *l*.
- A *delete operation* can be used to remove any subtree from a given tree. For example, we can remove the subtree rooted at node 4 by using the operation $del(4)$. That is, the operation $del(n)$ applied to any tree containing *n* all of *n*'s descendants (including *n*) from the tree.
- An *insert operation* can be used to attach one tree at some point in another tree. Refer to Figure 2. The third arrow in Figure 2 depicts the tree $T'$ being attached as the second child of the node 2 in tree $T_3$ by using the operation $ins(T', 5, 2)$. That is, an operation $ins(T, n, i)$ makes *T* the *i*th child of node *n*.

An *edit script* is a sequence of such edit operations. An edit script is applied to a tree by applying the operations one after another in the order they are listed. Figure 2 depicts the edit script $($$upd(1, 1.12)$, $upd(3, 3.41)$, $del(4)$, $ins(T', 2, 2)$, $upd(7, 7.77)$, $upd(8, 8.20)$$)$ applied to tree $T_1$ from Figure 1. Note that the resulting tree is identical to the tree $T_2$ (except for node identifiers). We say that our *edit script transforms* $T_1$ to $T_2$ .

## Costs

Each edit operation has a *cost* associated with it as described below:

- The *cost of an update operation* is simply the absolute value of the difference between the old and new labels. Thus updating a label 1.11 to 1.12 costs 0.01 units and updating a label 9.99 to 99.9 costs 89.91 units.
- Each node has a *deletion cost* that depends on its label and that is specified as part of the input. In Figure 1, the deletion cost of a node is indicated in parentheses next to the node. If no such number appears next to a node, its deletion cost is 1 unit by default. The *cost of a delete operation* is the sum of the costs of deleting all the nodes in the deleted subtree. For example, the operation $del(4)$ in our example costs 0.4 + 0.1 = 0.5 units.
- Similarly, each node has an *insertion cost* that depends on its label and that is specified as part of the input. In the figure, the insertion cost of a node is indicated in square brackets next to the node. If no such number appears next to a node, its insertion cost is 1 unit by default. The *cost of an insert operation* is the sum of the insertion costs all the nodes in the inserted subtree. For example, using the costs indicated in Figure 1, the cost of the operation $ins(T', 2, 2)$ in earlier example is 4.4 + 0.13 = 0.53 units.

  The cost of inserting or deleting a node whose label does not appear in any of the input trees is 1 unit by default.

The *cost of an edit script* is the sum of the costs of the operations it contains.

## Goal

Your goal is to write a program that takes as input two trees (and the node insertion and deletion costs) and produces as output the cost of a minimum-cost edit script that transforms one tree to another. (You need not compute such a minimum-cost edit script; you only need its cost.) As was the case for the trees in Figure 1, there is no correspondence between the node identifiers in the two trees; thus they are not specified in the input. You are also required to pretty-print the input trees as described below.

## Input format

The input file first lists the nodes in the first tree in preorder, one per line. For each node, the input file contains a line with three numbers that are separated using spaces. The first number is the depth of the node. The second is that node's label. The third is that node's deletion cost. After all the nodes in the first tree are listed in this manner, the input file contains a line ``-1 0 0'' to denote the end of the tree's listing.

Next, the input contains a blank line followed by a listing of the second tree. The second tree is listed using the method described above, the only difference being that the third number on each line is now the node's insertion cost. This listing of the nodes in the second tree is also followed by a line ``-1 0 0'' to denote the end of the listing.

Assume that all real numbers in the input are in the range [0,99.99] (inclusive).

## Output format

The output consists of two parts. First, you pretty-print the two input trees. Second, you report the cost of a minimum-cost edit script that transforms the first tree to the second.

You must first output the first tree by listing one node per line, in preorder. Each such line must begin with 4*d* space characters, where *d* is the depth of the node. Next, it contains the node's label, with two digits before and two digits after the decimal point. If the label is less than 10, include a leading space character. Thus, 9.87 is output as `` 9.87'' where denotes the ASCII space character. This label is followed by a single space character, followed by the node's deletion cost listed in the same format as that used for the label.

Next, you must output a blank line, followed by the listing of the second tree in the above format (with the deletion cost replaced by the insertion cost).

Finally, you output a blank line followed by a line containing the string ``Distance:'' followed by a space character followed by the the cost of the minimum-cost edit script. For outputting this cost, use the same format you used for the tree labels. (You can assume that the cost will always be a real number in the range [0,99.99].)

Do not include anything else in your output, not even blank lines.

## Example

For the trees depicted in Figure 1, the input and output is shown below. (For clarity, we use the character to denote the ASCII space character.) Note that the space characters at the beginning of lines in the input are optional; your program should work whether or not they occur in the input. As it turns out, the edit script depicted in Figure 2 is a minimum-cost edit script for this input, and its cost is 6.28, as reported in the output.

**Input 1:**

```
0 1.11 1.01
  1 2.56 3.29
    2 3.14 1
    2 4.4 0.4
      3 13 0.1
    2 6.18 0.11
  1 7 1
    2 8 1
    2 99.99 1
-1 0 0

0 1.12 1.20
  1 2.56 2.39
    2 3.41 2.9
    2 44.44 4.4
      3 13.13 0.13
    2 6.18 11.0
  1 7.77 1
    2 8.2 1
    2 99.99 1
-1 0 0
```

**Output 1:**

```
 1.11  1.01
     2.56  3.29
         3.14  1.00
         4.40  0.40
            13.00  0.10
         6.18  0.11
     7.00  1.00
         8.00  1.00
        99.99  1.00

 1.12  1.20
     2.56  2.39
         3.41  2.90
        44.44  4.40
            13.13  0.13
         6.18 11.00
            13.13  0.13
         6.18 11.00
     7.77  1.00
         8.20  1.00
        99.99  1.00

Distance:  6.28
```

**Input 2:**

```
0 1 1
  1 3 1
  1 2.2 1.1
    2 3.14 1.73
    2 7.8 0.5
  1 1 1
-1 0 0

0 1 1
  1 3 1
  1 2 1.1
    2 3.14 1.73
    2 8.7 0.3
  1 1 1
-1 0 0
```

**Output 2:**

```
 1.00  1.00
     3.00  1.00
     2.20  1.10
         3.14  1.73
         7.80  0.50
     1.00  1.00

 1.00  1.00
     3.00  1.00
     2.00  1.10
         3.14  1.73
         8.70  0.30
     1.00  1.00

Distance:  1.00
```

## Test data used in judging

| | |
|---|---|
| Input 1 | Output 1 |
| Input 2 | Output 2 |
| Input 3 | Output 3 |
| Input 4 | Output 4 |
| Input 5 | Output 5 |
| Input 6 | Output 6 |
| Input 7 | Output 7 |

## Our Solution

*Chau-Wen Tseng*
*Mon Mar 15 13:58:05 EST 1999*