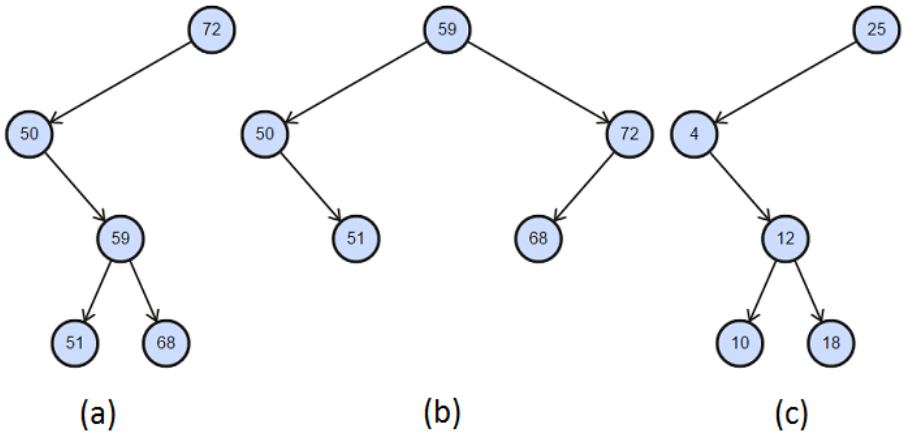
CSC2001F 2015 Practical 1

1. Introduction

You should read this document completely before starting the practical task. In this section the task is outlined, in section 2, we describe the relevant algorithms, and in section 3, we document the resources that we expect you to use. (You will be expected to implement a couple of Java classes, then develop your own to complete the task.)

1.1 Scenario

Suppose *T1* and *T2* are two Binary Search Trees (BSTs). *T1* and *T2* are ***similar*** if they have the same shape – that is, if their drawings look the same – For example, consider the three trees in the following depiction:



Trees (a) and (c) are similar, trees (a) and (b) are not similar, nor are trees (b) and (c).

Note that, while (a) and (c) are similar, they don't contain the same data.

1.2 Task

Write a program that:

1. accepts as input (from the command line) two lists of numbers,
2. uses them to construct two binary search trees, *T1* and *T2*,
3. prints a textual representation of *T1*,
4. prints a textual representation of *T2*,
5. reports whether or not they are similar, then
6. writes a textual representation of T1 to a file called T1.out, and
7. writes a textual representation of T2 to a file called T2.out.

The first number in a list should be assumed to be the value of the root node.

The textual representation written to file should be the same as that written to the screen i.e. the same algorithm is used in both cases.

You should provide evidence of testing your program on 6 pairs of BSTs: 3 where they are similar, and 3 where they are not.

Sample I/O:

```
Enter a comma separated list of numbers for tree one: 25, 4, 12, 18
Enter a comma separated list of numbers for tree two: 17, 11, 9, 2
Tree one:
              25
       4
          12
            18
Tree two:
              17
      11
   9
 2

The trees are not similar.
```

You are expected to construct your program using the framework detailed in section 3.

2. Algorithms

2.1 Similarity

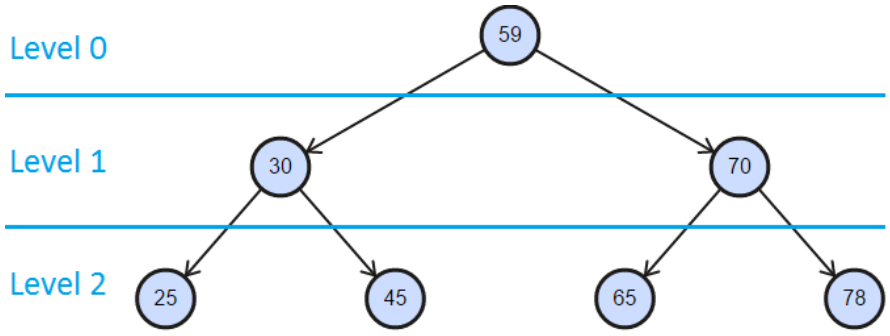
You should use a recursive algorithm to calculate similarity.

Given two trees *T1* and *T2* with root nodes *R1* and *R2* respectively, we say they are similar if:

1 *R1* and *R2* have the same configuration of sub trees.
(If *R1* has a left sub tree then *R2* must have a left sub tree and if *R1* has a right sub tree then *R2* must have a right sub tree).
2 The sub trees are similar.

2.2 Printing

Printing a tree requires a breadth first traversal. It helps to think of levels within the tree:



Given a list *Ln* of nodes at level *n*, we can easily obtain a list *Ln+1* of nodes at level *n+1*. (Just compile the list from the left and right sub nodes of each of the nodes in *Ln*.)

To print a tree, starting with a list containing only the root node, repeatedly, we print the list, and then get a list for the next level. We stop when we've reached the lowest level of the tree. (It helps to know the height of the tree to begin with.

The big issues in generating a textual output relate to spacing.

2.2.3 Incomplete trees

A tree is complete if all nodes are leaf nodes or have exactly two sub trees. If a tree is incomplete then knowledge of missing nodes is needed to be able to properly space the output.

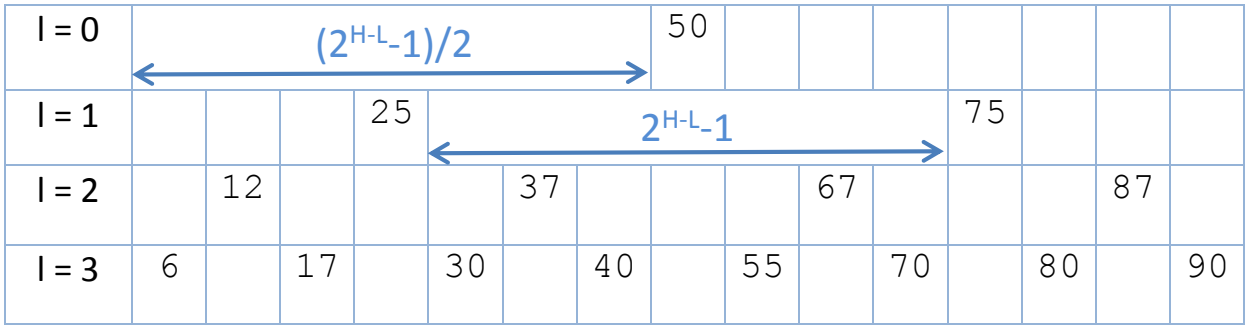
A simple approach is to use blank 'place holder' nodes. Given a list of nodes *Ln*, when compiling a list of nodes *Ln+1*, when a node in *Ln*, is missing a sub node, we insert a place holder in *Ln+1*.

2.2.4 Spacing

To format output properly we must address two issues:

- We need to use the same text width for each value in the tree.
  (If, for instance, the largest value is 999, then all values should be printed to a width of three characters.)
- Given a level L, we need to know the spacing before the first node, and we need to know the spacing between nodes.

The following diagram explains how to address the second issue:

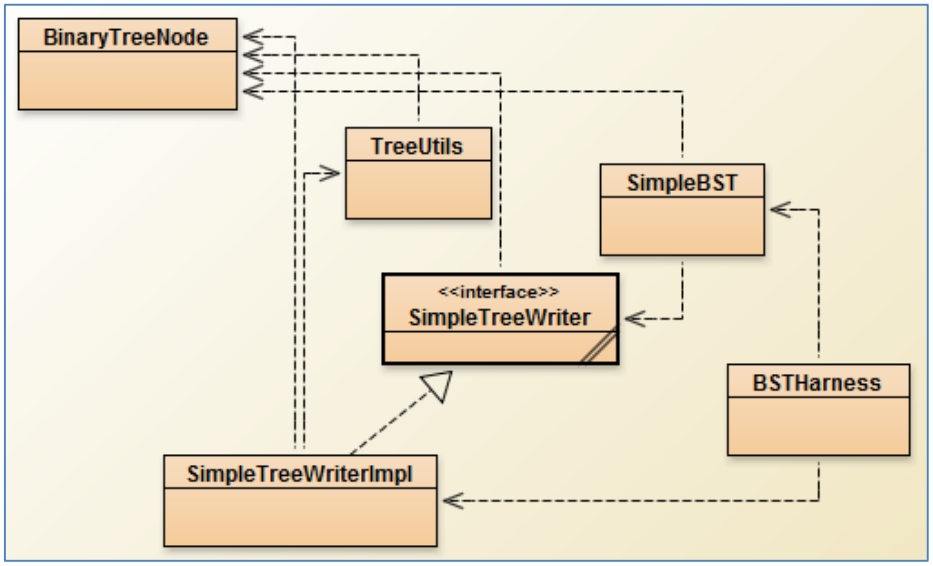


Given a height H, the number of leading blanks for level L is $(2^{H-L}-1)/2$.

Given a height H, the number of blanks between nodes for level L is $2^{H-L}-1$.

3. Resources

We have designed (but not completely implemented) a package of Java components that we would like you to use to structure your work:



- The SimpleBST and BinaryTreeNode classes implement an ordered binary tree ADT. *(You should not need to modify these components.)*
- SimpleTreeWriter describes a type of object that may be used to print a tree using a java.io.PrintStream object. (System.out is a PrintStream object and PrintStream objects can be created that print to Files.
- TreeUtils provides methods for: comparing two trees for similarity, creating a level 0 node list from a tree, creating a level *N+1* node list given a level *N* list.
- SimpleTreeWriterImpl is an implemention of SimpleTreeWriter. It provides a method that implements the printing algorithm sketched in section 2.2.
- BSTHarness is a simple test program that asks the user for a series of numbers, constructs a tree from them, and then prints it out.

Given these components, completing the program described in section 1 is straightforward, however, the resources are incomplete. The TreeUtils and SimpleTreeWriterImpl classes are missing. (You must implement them.)

Downloads

On the Vula assignment page you will find two zip files containing resources:

- A zip file of Java docs for the components.
- A zip file containing those components that have already been implemented

4. Marking

Tasks

To construct your program you must:

- Construct the TreeUtils class.
- Construct the SimpleTreeWriterImpl class.
- Develop additional class(es) that implement the behaviour describe in section 1.
- Provide evidence of testing on 6 pairs of BSTs: 3 pairs where the trees are similar, 3 where they are not.

If you complete these tasks you will gain 90% of the marks for the assignment. To obtain the final 10%, develop a printing algorithm that prints node values AND a representation of the branches between nodes.

Core Implementation (27)		
Command line input of *T1* and *T2*.	works (yes, partial, no)	(0, 2)
Conversion of each list to a BST.	works (yes, partial, no)	(0, 3)
Similarity comparison of *T1* and *T2* with recursive method.	works(yes, no)	(0, 4)
	Method is recursive(yes, no)	(0, 4)
T1 and *T2* printed to text files.	Printed in proper format (yes, no, partially)	(0, 4)
	Depictions are actually BSTs. (yes, no).	(0, 4)
Use of resources.	TreeUtil and SimpleTreeWriterImpl have been implemented (yes, no)	(0, 2)
	The SimpleTreeWriterImpl write method uses the TreeUtil methods. (yes, no ,partially)	(0, 2)
Tested on 6 pairs of tree (*T1*, *T2*).	3 same (yes, no)	(0, 1)
	3 different (yes, no)	(0, 1)
Coding Style (3)		
Meaningful naming	yes, no, partially	(0, 2)
Comments	yes, no.	(0, 1)
		Total 30 Marks

NOTE: You are expected to maintain a GIT repository and conduct unit testing for CSC2 practicals. (See requirements in the resources section of Vula.) Marks will be deducted from an assignment if these requirements are not met.