CSC9A3 Practical 4: Trees & Sorting

Introduction

This practical is aimed at giving you some practice with building and manipulating trees and also implementing a search algorithm.

For the first part of the practical, you will start with a simple binary tree node that holds an integer value and a method to add nodes to the tree.

Copy the directory \\Wide\Groups\CSCU9A3\Practicals\Practical4 to a suitable location in your own file space. Create a BlueJ or Eclipse based project that contains the Java files in the copied directory. You should find a class called IntTreeNode and IntegerTree and a JUnit test called IntegerTreeTest. IntTreeNode represents a binary node consisting of an integer value and references to left and right child nodes. IntegerTree provides methods to add nodes to a tree built from IntTreeNode and to perform an in-order traversal of the tree and also a method to print out the tree (from a side on view point). The results of the in-order traversal are returned as a String that you can print out and use to check that you are getting the correct results. IntegerTreeTest performs some simple tests to confirm that the above methods work as intended.

Read through the code and ensure that you understand how it works then try adding some new nodes in the method IntegerTreeTest.inorderTest and checking the results with new tests.

Task 1: Adding new traversal methods

You should notice that IntegerTree contains empty methods for walking the trees. Using the printTree method as a guide, implement the methods IntegerTree.preOrder, IntegerTree.inOrder and IntegerTree.postOrder and then write suitable tests in IntegerTreeTest.testWalks to confirm that they work. Note that instead of printing out a node, all you have to do to 'visit' it is to append the nodes contents to a StringBuffer as follows:

```
sb.append(n + ",");
```

Try altering the order in which the initial set of nodes are added to the tree in testWalkd and confirm that the results of walking the trees with a pre-order, in-order and post-order walk are what you expect. Note how the initial node added to the tree can skew the tree that is produced.

Checkpoint 1: Once you are satisfied that these methods work, show the code for your traversal methods and the test results to a demonstrator.

Task 2: Sorting Data

For the second part of the practical, we are going to look at implementing the selection sort algorithm and time how long it takes to sort increasingly large data sets. In the directory \\Wide\Groups\CSCU9A3\Practicals\Practical4, you should find the files Sorting.java and SortingTest.java. Create a new BlueJ or Eclipse project in your home file space and add these files to it.

The class Sorting.java contains just one method called *selectionSort* which takes an array of integer values and is meant to sort it using the selection sort algorithm. Your first task is to implement this method using the pseudo code given in lectures, then check that it works using the method *insertionSortTest* that is provided in *SortingTest.java*. This test method introduces a new assertion test that you may have not seen before called *assertArrayEquals*. This test takes an expected array and compares its contents with an array that you have produced.

Once you have managed to get the *selectionSort* algorithm to work, your next task is to see how long it takes to run for different sized data sets. A method called *sortForSize* has been provided for you in SortingTest.java. This method will create an array of a given size 'n', call your *insertionSort* method and note how long it took to run in nano-seconds. The test method *timeInsertionSort* has been provided which will make a single call to *sortForSize* and print out the number of items sorted and how long it took.

Add some additional calls to *sortForSize* with different sizes of data set and build up a list of items to sort versus time taken. Use this data in Excel to plot the relationship between data set size and time to sort and confirm the shape matches the N^2 computational order discussed in lectures.

Note: Do not try to sort more than 50,000 items – it will take increasingly longer to run and your computer may slow down when large value of 'n' are used. To plot this data, use a scatter plot in Excel (with connecting lines if you like). The value for 'n' should be on the 'x' axis and the time taken should be on the 'y' axis'.

Once you have completed this analysis, show your code and chart to a demonstrator.