CS 112 Summer 2, 2020 -- Homework Five

Due Friday, August 7th @ midnight with 24-hour grace period

The submission time for the entire assignment is the submission time of the last file submitted.

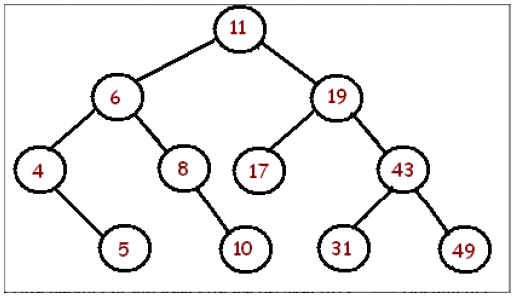
Introduction

You must observe the following requirements (for all homework submitted in this course):

- All programs should be literate, i.e., easily understandable by a human (say, your grader) and follow the Java Style Guidelines for CS112 posted on the class web site;
- All files for this homework should be submitted using WebSubmit, following the instructions on the class web site;
- You may not use data structure libraries such as ArrayList, since we are learning to write Java from the ground up and you must learn how these libraries are built; however, you are free to use (unless specifically directed otherwise) the basic libraries String, Character, Scanner, and Math; for this assignment, you may also use Double.parseDouble(....);
- You may freely use code from the class web site, the textbook, or lecture (unless specifically directed otherwise) as long as you cite the source in your comments; but you may NEVER use code from the web or other students' programs---this will be considered plagiarism and penalized accordingly.

Part A: Analytical Problems (14 points)

Before getting started on the remaining problems, review the lecture and textbook materials on binary search trees and recursion. The following problems have to do with the BST in this diagram:

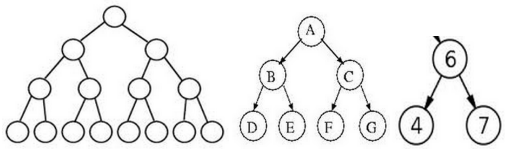


1. (5 parts): For this tree, give the (a) size, (b) depth of the node 31, (c) height, (d) length of the path from 11 to 49, and (e) list of all leaf nodes.

2. (2 parts) (a) Draw the result of inserting the keys 15, 7, 16, 12, & 13 into this tree; (b) assuming we can only insert integers, and no duplicates, into the original tree, what keys could possibly be inserted to the left of 31?

3. (2 parts) (a) Draw the result of taking the tree in the diagram and deleting the root three times using the deletion algorithm from lecture (on 3/13); (b) suppose we do not wish to unbalance the tree by deleting from the same side each time, and we decide that we will alternately delete from the right, left, right, left, etc. (starting with the right); delete the root of the tree in the diagram 4 times using this new strategy and show the resulting tree.

4. (4 parts) Let us call a tree "perfect" if it is a perfect triangle, i.e.,



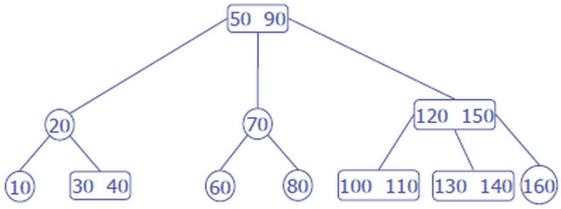
Suppose H is the height of a perfect binary tree, and N the number of nodes; (a) express H as a function of N, and (b) N as a function of H.

Now, let us call a binary tree "degenerate" if every node has either 1 or 0 children. Again, (c) express H as a function of N, and (d) N as a function of H.

5. (2 parts): Suppose we are going to insert the letters A, B, C, and D, into an initially empty BST. If we insert them in order, we get a pathological tree which is really just a linked list (and whose worst case time to find a node is linear, i.e., O(N)). But there are many different worst cases! (a) List 5 more of the the possible worst cases (hint: there are 8 in all!). Now suppose we have the letters A, B, C, D, E, F, and G. We would like to have a perfect tree which has the shape of a perfect triangle: for example, we could insert in the order D, B, F, A, C, E, G. (b) List 5 more of the possible insertion orders that would give you a complete binary search tree.

6. Insert the following sequence of keys into an intially empty 2-3 tree. You should write all the trees down as you do the transformations (this is what would be expected on an exam) but for the purposes of grading this exercise you can just draw the final tree that results. 15, 25, 20, 10, 12, 4, 19, 6

7. Consider the following 2-3 tree:



Suppose we count ONLY comparisons between two keys. (i) How many comparisons would necessary to find the key 10? (ii) How about 140? (iii) How about 60? (iv) Which key(s) occurring at leaf nodes would require a minimum number of comparisons? State which and how many comparisons. (v) Which key(s) would require a maximum number of comparisons? State which and how many comparisons. (vi) What is the average number of comparisons to find the keys in this tree (count for all and then divide by the size of the tree).

Part B: Programming Problems (86 points)

Problem B.0: Lab Challenge Problem (5 points)

Hand in the challenge problem(s) from Lab 03.

Problem B.1: Binary Search Tree Practice (25 points)

For this problem, you must develop the following methods which perform basic tests and operations on binary trees; the template TreeRecursion.java contains a unit test which you must uncomment one test at a time to test your implementation of these methods. The definitions of these kinds of trees are taken from the Wikipedia page on Binary Trees, which contains some more explanations and diagrams.

You may not use any loops in your implementation of these methods, so, yes, **they must be recursive.**

```
// B.1.1:  return the number of nodes in the tree -- just to get you started, already solved in lecture!

private static int size(Node t) {...}

// B.1.2:  print out a string representation of a binary tree using parentheses, infix style (see unit test)

private static String treeToString(Node r) {...}

// B.1.3:  print out a string representation of a binary tree using parentheses, prefix style (see unit test)

private static String treeToString2(Node r) {...}

// B.1.4: Count the number of leaves in a binary tree


private static int numLeaves(Node r) {...}

// B.1.5:  make a copy of a binary tree

private static Node copy(Node r) {

// B.1.6:  reverse the tree by exchanging left and right pointers (you will modify the original tree)

public static Node reverse(Node r) {...}

// B.1.7:  return true if the binary tree satisfies the binary search tree property, false otherwise
// Hint: write a helper method isBSTHelper(Node r, int lo, int hi) which checks if all items in
//     the tree are between the lower bound lo and the upper bound hi. If you do not know the
//     lower bound, use Integer.MIN_VALUE and if you do not know the upper bound use Integer.MAX_VALUE.

public static boolean isBST(Node r) {...}

// B.1.8:  return true if r is a degenerate binary tree, ie.,  in which all nodes have 0 or 1 child;
// a null tree is not degenerate, but a one-node tree is degenerate.

public static boolean isDegenerate(Node r) {...}

// B.1.9:  A perfect binary tree is a perfect triangle; test by checking recursively that both subtrees of each node have same size, OR
// find the height and the size and determine if these two quantities have exactly the relationship implied by a perfect tree.

public static boolean isPerfect(Node r) {...}

// B.1.10:  Determine if the two binary trees are structurally identical
// You MUST do this by recursion on the structure of the trees, and can not
// for example just return treeTostring(r).equals(treeToString(s))....

public static boolean equals(Node r, Node s) {...}
```

I STRONGLY suggest that you use step-wise refinement, developing one method at a time, and verifying that it passes its performance tests before moving onto the next method.

Problem B.2: Index Table and Inverted Index Table-- 50 pts

For this problem, you must develop a variation of a symbol table called an i**ndex table** (or sometimes just "index"); this is exactly the same as a symbol table, except that it has a list of values associated with it instead of just a single value. Such a data structure is very useful in a wide variety of applications.

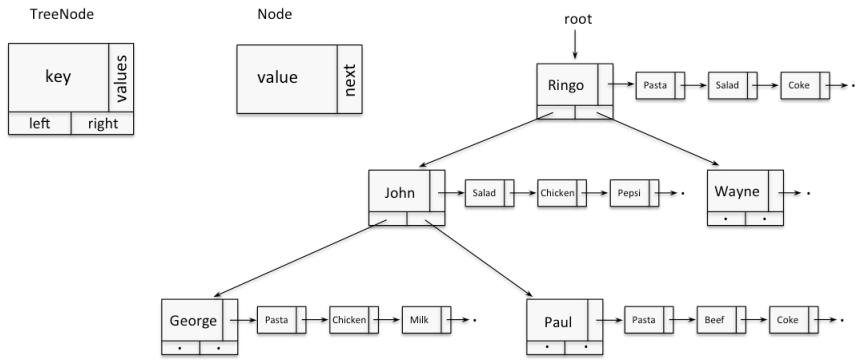
BST Representation of an Index

For this problem, you must, first of all, write an ADT `Index.java` which stores nodes as a binary search tree, holding String keys with a value which is a list of Strings, stored as a linked list. Thus, you would have two different kinds of nodes, e.g.,

```
private  class Node {      // Nodes for linked lists of values
  String value;
  Node next;
}

private class TreeNode {   // Nodes for the binary tree
  String key;
  Node values;             // pointer to linked list of values
  TreeNode left;
  TreeNode right;
}
```

Thus, for the index table used in the unit test, which records what five people ate at lunch, we might have the following structure:



The template for this program, containing the unit test, is here: Index.java. You must write the following interface methods for this program:

```
public void insert(String key) {...}

      Insert a new TreeNode with the given key, and
      a null list of values; if the key is already in the
      tree, do nothing.
```

```
public void insert(String key, String val) {...}

      If the key is already in the table, add
      val to the list of values; but if val is already
      in the list of values, do nothing;
      If the key is not in the table, insert a new TreeNode
      and create a linked list for the values, containing
      a single node with val.


public void insert(String key, String[] values) {...}

      The functionality of this is the same as the last,
      except that you must add all the values to the linked list of values;
      you should never insert duplicates into the LL.


  public String getValues(String key) { ....  }

      Return a String representation of the values associated with
      the key; if the key is not in the table, return null; if
      the list of values is null, return "[]".


  public void delete(String key) {

      Remove the key and all its values from the tree;
      if the key is not in the tree, do nothing.

  }
```

Verify that your code satisfies the unit tests, as usual

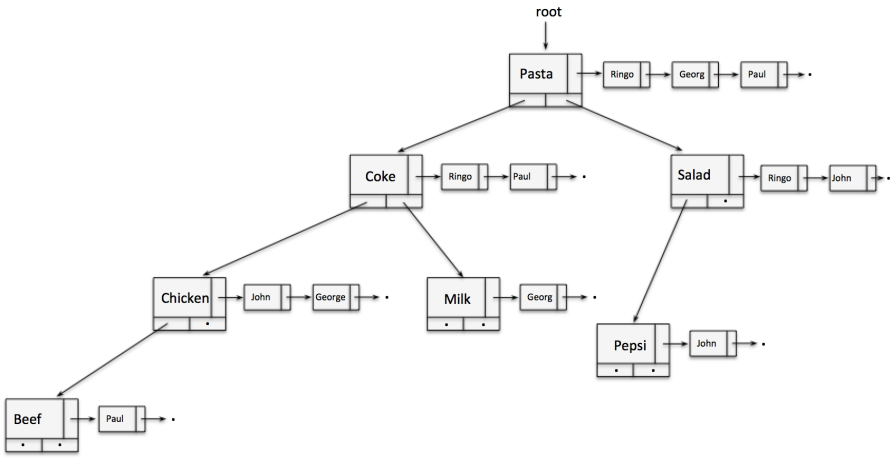
.

Inverted Index Tables

You can think of an index table as a representation of a table such as this:

	Pasta	Salad	Chicken	Beef	Coke	Pepsi	Milk
John		X	X			X	
Paul	X			X	X		
Ringo	X	X			X		
George	X		X				X
Wayne							

The keys are the row headings (the names), and the values are a selection of the column headings. However, it is often useful to be able to go the other way, and have keys which are the column headings, and values which are the row headings. This form of table could be used, for example, to see how popular various foods are (e.g., Pasta was chosen by George, Ringo, and Paul). This is called an Inverted Index Table. Here is an inverted index table corresponding to the same data shown in the table and in the previous figure:



Note that Wayne, alas, does not occur in the inverted table because he was too star-struck to eat a thing at his lunch with the Fab Four.

In order to create an inverted index from an index table, you "simply" have to create another, empty index, then traverse your original tree, both the tree nodes and the linked lists, and insert the appropriate values into the new index: the values become the new keys, and the keys become the new values. This is an interesting application of tree traversal combined with linked list traversal.

You must write the following method as part of the Index class, which constructs an entirely new index based on the ideas just explained, and returns it. Thus, if your original table is called `Orders`, then you would create a new inverted index table corresponding to `Orders` by calling `Orders.getInvertedIndex()`.

```
// method to build inverted index

public Index getInvertedIndex() {
  Index NewTable = new Index();

   ......

  return NewTable;
}
```

Verify that your code satisfies the unit tests, as usual

Problem B.3: A Practical Application of the Index ADT: The IMDB table (6 points)

For this problem, you will fill in the stub to read in values from a sample text file containing 5 movies and their casts; the name of the movie is the key, and the names of the cast members are the values. You can see from the template IMDB.java and from the text file moviesTest.java what is required to do. The main complication is how to parse the input file, but this is explained in the code template, and you can examine the text files as well.

Here is the data file: moviesTest.txt. You can right click on these links and select Download File.... to put them in your `src` directory where your code lives.

If you put the data file here, then `readInFile(....)` can find the file as in the main method of the template:

```
Index I = new Index();
readInFile("moviesTest.txt", I);
```

Verify that your program satisfies the unit tests as usual.

Submission:

Files to submit:

- hw05.txt
- TreeRecursion.java
- Index.java
- IMDB.java

Make sure that all programs are properly commented, neat and readable, all debugging trace statements are turned off or removed, and verify that you correctly pass all the performance tests in the unit tests.