CS 2420 Section 2 and Section 3
Spring 2016
Assignment 4: AVL Tree and Binary Search Tree
Part I: Due 11:59 a.m. March 1 (Tuesday), 2016 (Fixed Deadline; No Gift)
Part II: Due 11:59 p.m. March 4 (Friday), 2016
Total Points: 125 points

Part I [25 points]: Submit your solution via Canvas.

1) **[5 points]** For each of the trees shown in Figure 1 and Figure 2, determine whether it is a binary search tree or an AVL tree or both. If it is not an AVL tree, indicate which nodes are unbalanced.

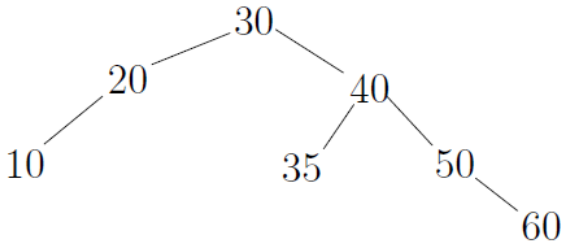


Figure 1

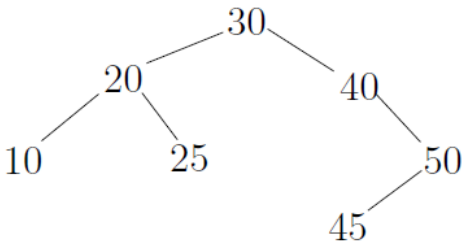


Figure 2

2) **[5 points]** For the AVL tree shown in Figure 3, draw the new tree after inserting 28, show the lowest unbalanced node if applicable, list the name of the appropriate rotation operation to rebalance the tree, and show the final AVL tree after this insertion.

3) **[5 points]** For the AVL tree shown in Figure 4, draw the new tree after inserting 55, show the lowest unbalanced node if applicable, list the name of the appropriate rotation operation to rebalance the tree, and show the final AVL tree after this insertion.

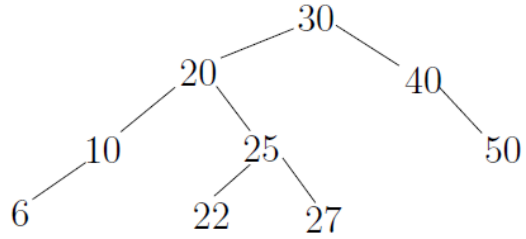


Figure 3

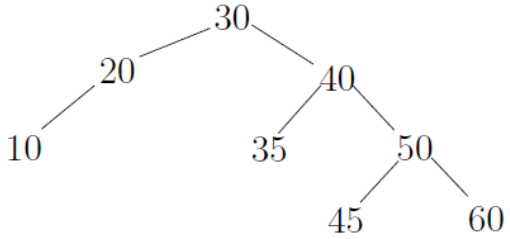


Figure 4

4) **[5 point]** For the AVL tree shown in Figure 5, draw the new tree after removing 30 by using the smallest key in the right subtree of 30 to replace it. For completeness, please show each intermediate tree after performing the appropriate rotation operation. Make sure to clearly show the unbalanced node and list the name of the rotation operation for each intermediate step.

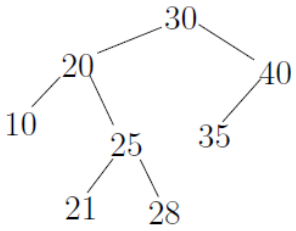


Figure 5

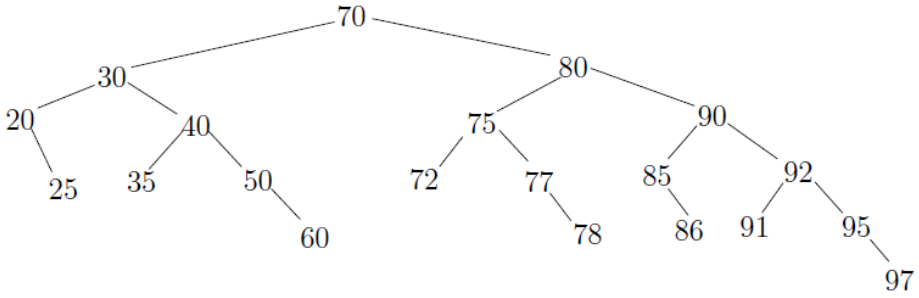


Figure 6

5) **[5 points]** For the AVL tree shown in Figure 6, draw the new tree after removing 20. For completeness, please show each intermediate tree after performing the appropriate rotation operation. Make sure to clearly show the unbalanced node and list the name of the rotation operation for each intermediate step.

Part II [100 points]: (dictionary.txt and letter.txt are the same documents used in Assignment 3)

You are required to develop a spell checker using an AVL tree, which makes searching each word in a document to be faster than using a binary search tree. You are supplied with a dictionary of about 14,000 words, which are saved in a file "**dictionary.txt**" in alphabetic order.

[55 points] You must create an AVL tree, which contains the following public methods, which can be **implemented using either recursive solutions or non-recursive solutions. The new functions, which are different from the ones in Assignment 3, are highlighted in red for your quick reference.**

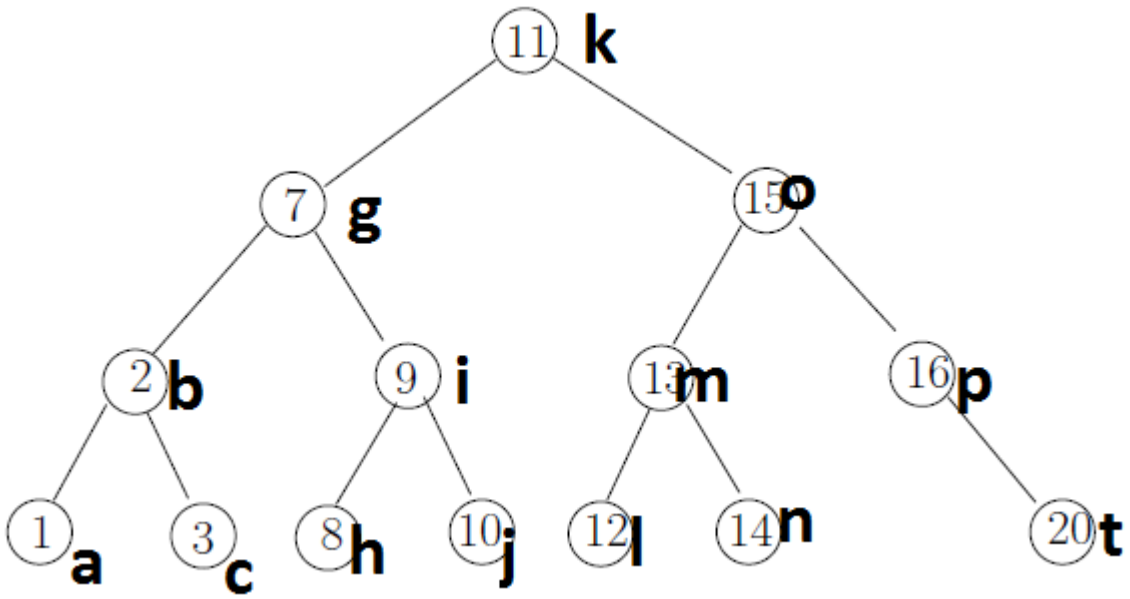
- **[1 point] Constructor**
- **[2 points] Copy constructor**
- **[2 points] Destructor**
- **[2 points] Size**: Return a count of the number of nodes in the tree.
- **[2 points] Height**: Return the height of the tree, which is defined as the length of the path from the root to the deepest node in the tree (i.e., the maximum distance from all leaf nodes to the root).
- **[8 points] Delete**: Delete a word from the AVL tree if a node in the AVL tree contains the desired word. Otherwise, no delete operation is performed.
- **[5 points] Traverse**: Traverse **and print** the AVL tree using the **pre-order traversal** and return a string containing all the words, which are stored in the AVL tree, in the descending order. Note: Spaces are used to separate the two words. For example, if "test", "good", "score" are inserted into an AVL tree, the returned string after calling Traverse function is: test score good.
- **[8 points] Insert**: Insert a string into the AVL tree.
- **[2 points] Find**: Return true if the string is found in the tree. Return false otherwise.
- **[5 points] FindComparisons**: Return the number of comparisons to determine whether the string is in the tree.
- **[5 points] CountTwoChildren**: Return the number of nodes with two children.
- **[5 points] CountOneChild**: Return the number of nodes with one child.
- **[5 points] CountLeaves**: Return the number of leaf nodes (e.g., nodes without any child).
- **[2 points] GetHeight**: Return the height for a node
- **[1 point] GetRoot**: Return the word contained in the root node

[10 points] Modify BSTree.h to contain the above 15 public functions and other applicable private functions. Modify BSTree.cpp to contain the C++ implementation of each function listed in BSTree.h.

Write a driver program to do the following tasks:

1. **[5 points]** Demonstrate the correctness of all the public member functions of the AvlTree by **reading an input file "hw4_input.txt"**. Each line of the input file is either "insert x" or "remove x". The program reads the input file line by line and perform the operations accordingly. Finally, the program prints both pre-order traveral lists of tree and the words in the descending order (e.g., calling Traverse function). In addition, the program also prints the information contained in the root, the height of the tree, the number of nodes in the tree, the number of nodes with two children, the number of nodes with one child, and the number of leaf nodes on the console.
   The sample output on the console is as follows:
   The pre-order traversal list is:
   k g b a c i h j o m l n p t
   The descending traversal list is:
   t p o n m l k j i h g c b a
   The word in the root is: k
   The height of the tree is: 3
   The number of nodes in the tree is: 14
   The number of the nodes with two children is: 6
   The number of the nodes with one child is: 1
   The number of leaf nodes is: 7

   The final tree is like the following, where 1, 2, 3, 4, 5, 6, …, and 26 represent a, b, c, d, e, f, …, and z, respectively.



2. **[5 points]** Read the file "**dictionary.txt**", where each word is on a separate line, and store these words into an AVL tree. **After reading the dictionary into the AVL tree, have your program report the size of the tree (i.e., the number of nodes) and the height of the tree.** Since the dictionary is in the alphabetic order, you cannot sequentially read in each word and insert these

words in the tree (i.e., it will essentially be a linked list if doing so). **In other words, you must insert words in a random order as you did in the previous assignment to make an AVL tree that can be efficiently searched.**

3. **[5 points]** Read the file "**letter.txt**" which contains some words (with mixed small and capitalized letters) used in a letter without any punctuations. Output to the console all the misspelled words (i.e., any word not found in the dictionary). Keep in mind the grader will use a different file when doing the grading.
4. **[10 points] Repeat the tasks 1, 2, and 3 using the modified BSTree class. Make sure that you use the same randomized strings generated in task 2 to create a binary search tree.**
5. **[10 points]** Print the following information related to the two trees created from the same randomized strings side-by-side:
   - The word in the root of the BSTree vs. the word in the root of the AvlTree
   - The size of the BSTree vs. the size of the AvlTree
   - The height of the BSTree vs. the height of the AvlTree.
   - The number of BSTree nodes with two children vs. the number of AvlTree nodes with two children.
   - The number of BSTree nodes with one child vs. the number of AvlTree nodes with one child.
   - The number of BSTree leaf nodes vs. the number of AvlTree leaf nodes.
   - The number of comparisons performed for each word in "letter.txt" file using BSTree vs. the number of comparisons performed for each word in "letter.txt" file using AvlTree.
   - The average number of comparisons used by BSTree vs. the average number of comparisons used by AvlTree.

Below is the AvlTree.h file, which contains C++ AvlTree class interface, for your reference.

```
#ifndef AVLTREE_H
#define AVLREE_H

#include <string>
#include <iostream>
using namespace std ;

class AvlNode
{
public:
        string word ;
        int height ;
        AvlNode *ptrLeft, *ptrRight ;
        AvlNode() ;     // Default constructor
        AvlNode(string s) ;  // Constructor with one parameter
} ;

typedef AvlNode * AvlNodeptr ;

class AvlTree
{
```

```
private:
        AvlNodeptr root ;

public:
        AvlTree() ;  // Initializes root to NULL.
        AvlTree (const AvlTree &);  //Copy constructor
        ~ AvlTree () ;  // Destructor.
        int Size ();  // Return the number of nodes in the tree.
        int Height();    // Return the path length from the root node to a deepest leaf node in the tree.
        void Delete(string s) ; // Delete a string.
        string Traverse() ;  // return a string containing all strings stored in the binary search tree in
                             // the descending order and print the strings in pre-order

        void Insert (string s) ;  // Insert a string into the binary search tree.
        bool Find (string s) ;   // search s in the list.  Return true if s is found; otherwise, return false.
        int FindComparisons (string s) ;  // search s in the list and return the number of comparisons to
                                          // determine whether s is in the tree.
        int CountTwoChildren() ; // return the number of nodes with two children
        int CountOneChild() ;      // return the number of nodes with one child
        int CountLeaves() ;        // return the number of leave nodes
        int GetHeight(AvlNodeptr v);   // return the height of node v
        string GetRoot() ;           // return the word contained in the root node

private:
        // You need to add the following private functions to implement the rebalance operations.
       //  You may need to add other private functions you feel appropriate
        void balance(AvlNodeptr &);  // make the tree balanced after insert and delete
        void rightRotate (AvlNodeptr &); // Right rotation
        void leftRotate (AvlNodeptr &);  // Left rotation
        void doubleLeftRightRotate (AvlNodeptr &);  // Left right double rotation
        void doubleRightLeftRotate (AvlNodeptr &);  // Right left double rotation
} ;
#endif
```