CS 1410 Introduction to Computer Science – CS2
Assignment #8: Binary Search Tree
Given: Apr 15, 2015
Due: 11:59 p.m. Apr 25, 2015
Total Points: 50 points

Problem description

You should submit your solution on Canvas as a cpp file named as Firstname_Lastname_HW10.cpp. For example, if your name is Adam Smith, the submitted file will be Adam_Smith_HW10.cpp.

This assignment has the same problem of word counting as Assignment 1. Please read its problem description for the detailed requirements of input and output.

In Assignment 1, you used arrays for storing words and their frequencies. As you have observed, that implementation is inflexible because you can store only a limited number of words (not to exceed the array size). In addition, adding a new word is inefficient, as you need to shift other words in the arrays in order to insert the new one.

In this assignment, binary search trees will be used. You will use a binary search tree T for counting words. That is, when you read a word from the input file, you will find the node containing that word in the tree. If that node exists, you increase its frequency. If not, you create a new node for the word (with initial frequency of 1) and insert it into the tree.

After reading all the words, you will use another binary search tree F to sort the frequencies. You will visit each node of T and insert the corresponding word and frequency into F based on frequencies. Because different words might have the same frequency, when inserting, you compare the frequencies first. If they are equal, you will compare the words.

Programming tasks

Task 1 (5 points). Declare a class WordNode to store a word and its frequency. You need store the words in a binary tree, so this class will have two pointers to the left and right WordNode. Then, implement a constructor for class WordNode with four parameters for its fields. The default value for the frequency is 1 and those of the left and right pointers are NULL. The prototype is:

WordNode(int word, int freq = 1, WordNode *left = NULL, WordNode *right = NULL)

Then, declare a class WordTree to represent binary search trees for words. This class has a WordNode* pointer for the root of the tree.

Task 2 (5 points). Implement a member function of class WordTree to insert a word into the corresponding binary search tree sorted by words. Its prototype is:

void insert(int word)

Task 3 (5 points). Implement a member function class WordTree to insert a word into the corresponding binary sorted (descendingly) by frequencies and then (increasingly) by words. Its prototype is:

void insert(string word, int freq)

Task 4 (10 points). Implement a member function of class WordTree to traverse this tree to build another binary search tree (sorted by frequencies). Its prototype is:

void copyTo(WordTree &tree)

This function copies words and their frequencies from the current tree to the binary search tree referred by ‘tree’. It traverses all nodes of the current tree in the **pre-order** mode and insert its word and frequency into 'tree' using its 'insert' function implemented in Task 3.

Task 5 (5 points). Implement a member function of class WordTree to read the words in an input file one-by-one and insert them into the tree. Its prototype is:

void load(string filename)

Task 6 (5 points). Implement a member function of class WordTree to write the words and frequencies to an output file in desending order. The prototype is:

void save(string filename)

To write the frequencies by descending order, you need to traverse the tree using the **in-order** mode.

Task 7 (10 points). Implement your main function with the following steps:

a. Initialize two WordTree objects named 'treeWord' and named 'treeFreq'.

b. Call function ‘load’ in Task 5 to read input words and store them in 'treeWord'.

c. After reading, call function 'copyTo' in Task 4 to copy words from 'treeWord' to 'treeFreq'.

d. Call the ‘save' function from Task 6 to write the words and frequencies stored in 'treeFreq' to the output file.

Task 8 (5 points). Test your program with a large input file (e.g. a web page or a book) with at least 10 million words (you could also use the input file given in Assignment 1). Compute the running time and compare the result to the best result you got in Assignment 1 with the same input file. Report the size and height of 'treeWord' and 'treeFreq'.