ICS 46
Homework Assignment 6

Programming Assignment

This programming assignment will consist of 4 parts:

1. Implement the classes *TreeNode* and *BinarySearchTree*
2. Test and measure the performance of *BinarySearchTree*
3. Graph the data from part 2.
4. Convert *TreeNode* and *BinarySearchTree* into a template.

A Submission Summary is Available on the Last Page

1. (30 points) Implement the classes *TreeNode* and *BinarySearchTree*

- *TreeNode* can be implemented as described in the lecture slides.
  - It will have two data members:
    - a **KeyType** which will be of type *string*.
    - an **ElementType** which will be of type *int*. This will count how many times you have seen this word in the input file – yes, I know each word only appears once in our test input files, but pretend we are writing a general program to count frequency of occurrence of each word in a given file.
  - Note: Part 4 involves converting this class to use templates.
    - You may consider making your class as a template in this part (Part 1)
    - This is not a requirement, but it may save time.
- *BinarySearchTree* can be implemented as described in the lecture slides.
  - You have the option of writing some functions recursively.
  - You do not have to show T(N) derivations if your functions are recursive.
  - Note: Part 4 involves converting this class to use templates.
    - You may consider making your class as a template in this part (Part 1)
    - This is not a requirement, but it may save time.
- **For your report -** Copy/Paste your code for the functions below with T(N) and time complexities in Big-Oh notation.
  - void BinarySearchTree::insert(string s, int count)
  - int BinarySearchTree::find(string s)
  - void BinarySearchTree::remove(string s)
  - TreeNode * TreeNode::insert(string s, int i, TreeNode *T)
  - int & BinarySearchTree::operator[] (string s)

2. (20 points) Test and measure the performance of *BinarySearchTree*

- Similar to Homework 5, test your *BinarySearchTree* implementation by testing on one file of words.
    - Note: we will use random.txt for testing, but I want you to run it on words.txt to see for yourself what happens when the input is sorted.
- To test, write plain functions (not methods) (which must remove them one at a time by making a loop over each word in the appropriate input file and calling BinarySearchTree::remove(string)).
- Execute your program, results of measuring T(N) over 10 partitions of the data for each of your three XAllWords functions (which stands for insertAllWords, findAllWords, and removeAllWords).
- You can do this by passing in a counter, N, to XAllWords which indicates how many words to insert, find, or remove. That function will insert, find, or remove only the first N words from random.txt. Where you call XAllWords, you will place it in a loop with the loop control variable ranging from 1 to 10 and you will set this N parameter by multiplying the loop control variable times 1/10th of the large N which is 45,392. 1/10th of this big N is 4,529. Here is a sketch of how each will work:
- Psuedocode*:*

```
For each file random.txt and words.txt:
        For each partition consisting of 1/10th, 2/10,...up to the full 10/10 file:
                Measure and report the time for:
                        insertAllWords
                        findAllWords
                        removeAllWords
```

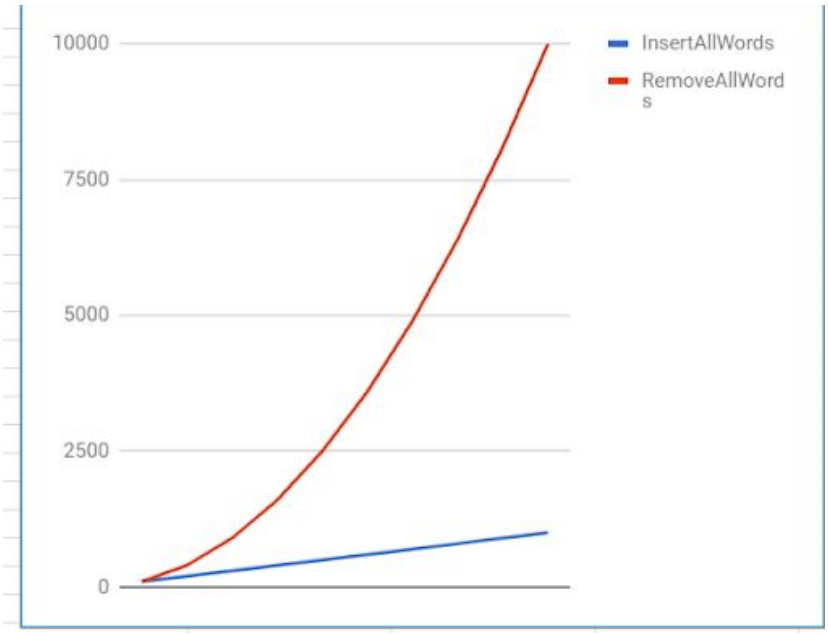
- Example skeleton code:

```
void insertAllWords(BinarySearchTree & T, int partition, char *fileName)
{
    // open the file, start a timer,
    // insert the first partition*N/10 words into T,
    // stop the timer, close the file, report the time by printing to console/cout
 }
void measureAll(char *fileName)
{
     for (int i=1; i<=10; ++i)
     {
        BinarySearchTree T; // important to start with an empty Tree here
        insertAllWords(T, i, fileName);
        findAllWords(T, i, fileName);
        removeAllWords(T, i, fileName);
     }
}
```

- You may reuse your code from the previous Homework 5 assignment if you find it helpful. However, you may not use someone else's code or any code you find in a book or on the Internet or in my lecture slides.
- **For your report -** Include a screenshot of the valgrind execution (without memory leaks) of your program's test & measure functions.
    - Example:
        - File: Random.txt. Partition: 1/10. Function: insertAllWords. Time: 50s
        - File: Random.txt. Partition: 1/10. Function: findAllWords. Time: 60s
        - File: Random.txt. Partition: 1/10. Function: removeAllWords. Time: 40s
        - File: Random.txt. Partition: 2/10. Function: insertAllWords. Time: 50s
        - …

3. (30 points) Graph the results of Part 2

- (30 pts) Graph the results of measuring T(N) over 10 partitions of the data for each of your three XAllWords functions (which stands for insertAllWords, findAllWords, and removeAllWords).
    - Your graph should have N on the X axis and T(N) on the Y axis.
    - Use Google Docs spreadsheet program or MS Excel or some similar program to graph your data. Here is a sample, https://docs.google.com/spreadsheet/ccc?key=0Ajv5LmqlazNHdC0xbkxtaEVGbk01ZF8xMDBZakZIRXc
- **For your report -** Include one graph for the measured times as a function of number of words inserted for each of the XAllWords functions over the partitioned files, just for random.txt only. Remember words.txt may cause your program to segmentation fault if you used recursion for insert, find, or remove.
    - Example from the Google Docs spreadsheet (missing findAllWords):



4. (20 points) Convert *TreeNode* and *BinarySearchTree* into a template.

- Convert *TreeNode* and *BinarySearchTree* into a template so that KeyType and ElementType may be specified with different types.
- Write a public method for *BinarySearchTree* called *countLengths* that traverses the entire tree and counts how many words of each length appears in the file.
- **For your report -** Include a screenshot of the valgrind execution (without memory leaks) of your program's function outputting the number of words for each length.
    - Example output:
        - Words in random.txt of:
            - length 1: 20 words
            - length 2: 50 words
            - etc...

Submission Requirements

1. Submit a zip of your program to Canvas.
    - The zip file should consist of a directory named by your UCINetID containing all your program files and a Makefile to build them.

2. Submit a pdf report to Gradescope with the following format:
    - Part 1: Code Implementation & Big-O Time complexity Analysis
        - Five points off for each error in time complexity and for each incorrect method or function up to the maximum.
        - Copy/Paste your code for the functions below with their O(N) (and T(N), if not recursive) time complexities.
            - BinarySearchTree::insert(string s, int count)
            - BinarySearchTree::find(string s)
            - BinarySearchTree::remove(string s)
            - TreeNode::insert()
            - int & BinarySearchTree::operator[] (string s)
    - Part 2: Testing BST under valgrind with no memory leaks
        - Include a screenshot of the valgrind execution (without memory leaks) of your program's test & measure functions.
    - Part 3: Graphing the data
        - Include one graph for the times of the XAllWords functions over the partitioned files, just using random.txt for your test measurement.
    - Part 4: Testing TreeNode and BinarySearchTree as a template
        - Include a screenshot of the valgrind execution (without memory leaks) of your program's functions outputting the number of words for each length.
        - Example output:
            - Words in random.txt of:
                - length 1: 20 words
                - length 2: 50 words

Homeworks that are not formatted nicely will not be accepted. Pay careful attention to memory management. Do not share memory across objects. Obvious code copying, from a textbook, from another student, or within your own program, will be down-graded (write everything once (perhaps by writing auxillary functions), then call the function or method each time that operation is required). Feel free to add additional functions or methods, but please use the data members outlined in the homework.