CS 202, Summer 2023

Homework 4 – Balanced Search Trees

Due: 23:59, August 8, 2023

Important Notes

Please do not start the assignment before reading these notes.

1. Before 23:59, August 8, upload your solutions in a single **ZIP** archive using Moodle submission form. Name the file as studentID_secNo_hw4.zip.
2. Your ZIP archive should contain the following files:
    - Your solution to the first part as a **pdf**. You should prepare and upload **handwritten** answers for Question 1 (in other words, do not submit answers prepared using a word processor). Use the exact algorithms shown in lectures.
    a. Code files (only the "**.cpp**" and "**.h**" files that you write for the homework) of the second part and a "**Makefile**" for the compilation of your code that produces the executable.
    b. In the end, your zip file should contain ONLY **code files**, "**Makefile**" and a **pdf**; any violation of these causes a significant loss from your grade.
3. You can use the codes provided in your textbook or the lecture slides. However, you cannot use any external data structure implementations such as STL's in your code.
4. Do not forget to put your name, student id, and section number in all of these files. Well comment your implementation. Add a header as given below to the beginning of each file:

```
/*
 * Title: Balanced Search Trees
 * Author: Name Surname
 * ID: 21000000
 * Section: 1
 * Assignment: 4
 * Description: description of your code
 */
```

5. If you do not know how to write a Makefile, you can find lots of tutorials on the Internet. Basically they are files that contain a list of rules for building an executable that is used by "make" utility of Unix/Linux environments. "make" command should build an executable called "**avlfreq**", so write your Makefile accordingly (i.e. at the end, when you type "make" in terminal on your working directory, it should produce "**avlfreq**" executable)
6. The test file is available in Moodle next to this document. It is James Joyce's *Dubliners* (https://en.wikipedia.org/wiki/Dubliners). Please note that your program will be tested with additional data.
7. This homework will be graded by your TA, Saeed Karimi. Thus, please contact him directly (saeed.karimi at bilkent edu tr) for any homework related questions. There will also be a forum on Moodle for questions.

- Although you may use any platform or any operating system to implement your algorithms and obtain your experimental results, your code should work on the **dijkstra** server (dijkstra.ug.bcc.bilkent.edu.tr). We will compile and test your programs on that server. If your C++ code does not compile or execute in that server, you will lose points.

Attention: For this assignment, you are allowed to use the codes given in our textbook and/or our lecture slides. However, you ARE NOT ALLOWED to use any codes from other sources (including the codes given in other textbooks, found on the Internet, belonging to your classmates, etc.). Furthermore, you ARE NOT ALLOWED to use any data structure or algorithm related function from the C++ standard template library (STL).

Do not forget that plagiarism and cheating will be heavily punished. Please do the homework yourself.

1) Question Part (40 points)

Assume that we have the following balanced-searched tree implementations:

a) AVL tree
b) 2-3 tree
c) 2-3-4 tree
d) Red-Black tree

Starting with an empty balanced search tree, we would like to insert the following keys into the tree in the given order:

30, 40, 60, 90, 65, 70, 50, 95, 75, 80, 110

and then, delete the keys 65, 95, 75 (in the given order) from the tree. Note: while deleting an internal node, its inorder successor should be used as the substitute if needed.

Show the structure for the AVL, 2-3, 2-3-4, and Red-Black trees after insertions and deletions with sufficient detail. For Red-Black tree, show only insertions. For other tree types above other than Red-Black trees, show both insertions and deletions.

2) Programming Part (60 points)

You are to write a C++ program to count the frequency (number of occurrences) of unique words in a text file. You may ignore any capitalizations and assume that words in the text file are separated with whitespaces and punctuations. For example; the word "you'll " will be counted as two different words "you" and "ll". Your program should construct an AVL tree and consider each word as a key.

You are to use a pointer based implementation of an AVL tree to store the words and their counts. Each node object is to maintain the associated unique word as a string, its current count

as an integer, and left and right child pointers. On top of the regular operations that a AVL has, you must implement the following functions (you don't need to implement delete procedure of AVL tree):

- **`addWord:`** adds the specified word in the AVL if not already there; otherwise, it simply increments its count.

- **`generateTree:`** reads the input text and generates a AVL tree of words. In this function you should detect all of the words in the input text and add them to the tree by using the **`addWord`** function.

- **`printHeight`**: computes and prints the height of the AVL tree

- **`printTotalWordCount`**: recursively computes and prints the total number of unique words currently stored in the tree (number of elements in the tree).

- **`printWordFrequencies`**: recursively prints each unique word in the tree in alphabetical order along with their frequencies in the text file.

- **`printMostFrequent`**: finds and prints the most frequent word with its frequency in the text file.

- **`printLeastFrequent:`** finds and prints the least frequent word with its frequency in the text file.

- **`printStandartDeviation:`** computes and prints the standart deviation of word frequencies in the text file.

The program must be compiled using a Makefile you provided and it should run with the following command. Whenever its computation is finished it should produce two output files, named as "**wordfreqs**" and "**statistics**". Sample execution will be as follow (As you can see input file name will be given as parameter in command line):

./avlfreq <input_filename>

Output files will be produced in the directory your program executed and contain the followings:

wordfreqs: This file will contain each unique word with its frequency on each line. Words must be printed in alphabetical order.

Sample:

```
at 4
....
hello 22
...
yellow: 40
```

statistics: This file will contain statistics about input text file and the avl tree in following format:

Sample:

```
Total Word Count: 50
Tree Height: 8
Most Frequent: yellow 40
Least Frequent: at 4
Standard Deviation: 5,76
```