CS 202 Fundamental Structures of Computer Science II

Assignment 4

Date Assigned: November 23, 2015
Due Date: December 9, 2015 - 23:55 (sharp)

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

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

40, 55, 80, 105, 85, 90, 79, 115, 95, 100, 130

and then, delete the keys 85 115 95 (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, and 2-3-4 trees after additions and deletions with sufficient detail.

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 seperated 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 a 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 dont 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 following informations:

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
```

Code Format and Notifications

You have to follow the following instructions about the format, programming style and general layout of your program.

- You can use the codes provided in your text book or the lecture slides. However, you cannot use any external data structure implementation such as STL's in your code.
- Don't forget to write down your id, name, section, assignment number or any other information relevant to your program in the beginning of every file that you are submitting.
- Don't forget to write comments at important parts of your code.
- You are free to write your programs in any environment (you may use either Linux or Windows). On the other hand, we will test your programs in a Linux environment and we will expect your programs to compile and run on Linux. If we cannot get your program work properly on Linux, you will lose a considerable amount of points. Therefore, we recommend you to make sure that your program compiles and properly works on Linux before submitting your assignment.
- You should upload your solutions to Moodle. You should upload a **single zip** file that contains:
    a. Your solution to first part as **pdf** (do NOT send photos of your solution).
    b. 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.
    c. 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. Name of the pdf should be "***Section_ID_SurnameName.pdf***".
- 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)
- The test file is available on the course website 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.
- Late submissions will not be graded.
- This homework will be graded by your TA Gündüz Vehbi Demirci (gunduz.demirci@bilkent.edu.tr). You may contact him for further questions.

DO THE HOMEWORK YOURSELF. PLAGIARISM AND CHEATING ARE HEAVILY PUNISHED!!!