IIT Ropar
CSL201 Data Structures
Semester 1, AY 2018/19

Lab Assignment 5 - 40 marks Due on November 5, 11:59 PM

Objective

To understand and implement an AVL tree and to learn to customize a basic AVL tree for special application requirements.

Instructions

1. You are to use C++ programming language to complete the assignment.
2. Provide a Makefile to compile your final code.
3. All function and class declarations should be in ".hpp" files while the definitions should be in ".cpp" files. It is recommended that you have one header file and one cpp file for each class. If the class is templated, declare it in the hpp file itself.
4. This is an individual assignment. You can have high level discussions with other students, though.
5. Include a "Readme.txt" file on how to compile and run the code.
6. Upload your submission to moodle by the due date and time.
7. Late submission is not allowed. No assignment will be accepted through email after submission system closes. I suggest you to upload the assignment at least 1 hour in advance.
8. No doubts will in entertained on the last two days of submission, i.e. November 4 and November 5.

Program Description

In this assignment you need to implement the dictionary you implemented in the assignment 4 using an AVL tree. Define alphabetical order on the numbers. To define order on keys, assume the absent character to be "a". For example, to compare "chom" and “chomu”, you can assume “chom” to be “choma”. Hence, if a word is subset of another work, the shorter word will smaller. See below the input format description taken from assignment 4:

“The program should take an input file name as argument and add all the words in the file to the dictionary. The file format is as follows:
Word1 meaning1
Word2 meaning2
…..

In other words, the first word in each line is the “word” and the remaining text is its meaning. You have to use the “word” as the key of an entry. See files words.txt for example. The “entry” structure/class should store word and meaning separately, i.e., entry = (w,m). The hash table should store entry pointers.”

Your program should support the following tasks:

- add(s): extract the key from s (the first word), create an entry, and add that entry to the tree, if it is not already there; else update word’s meaning.
- remove(w): remove the entry with key k from the table, if it's there.
- search(): Search and display the meaning of word w.
- size(): return the number of strings in the tree.
- countAllInRange(k1,k2): Compute and return the number of entries in map with key k such that k1 ≤ k ≤ k2; k1 and k2 may not be present in the tree. For example, if k1=a and k2=d, the function should return all the words starting with a,b,c, and d. Keys can be words also.

Program Interface
The program should give the option to the user to load from input file (words.txt) or enter keys manually. The interface should allow the user to add/remove keys one by one and display the level order traversal of the tree where the keys in the same level appear in the same line. It should also allow use to read from the file and search meaning of any word.