Data Structures

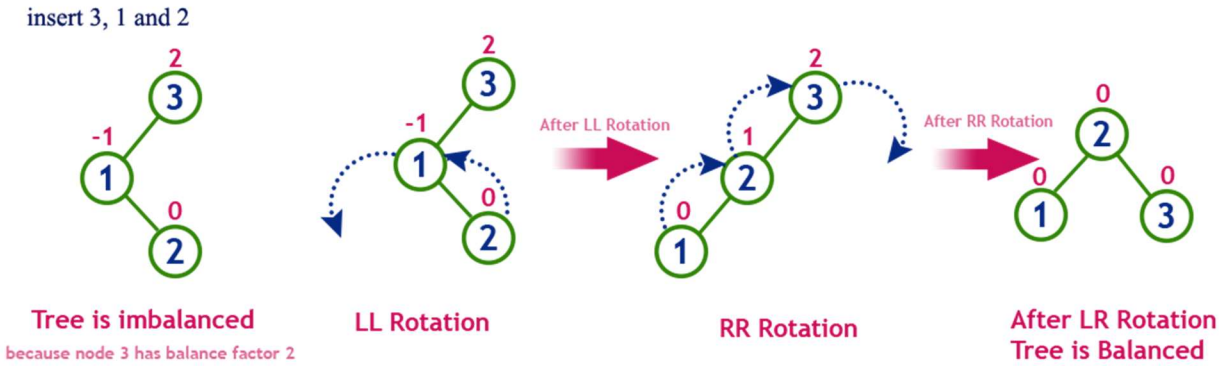
CS202

Assignment 3

Deadline: 20th October 2019

11:55 pm

In case of ambiguity, contact Anusheh Zohair Mustafeez
at 21100072@lums.edu.pk



Q1. In this part you'll implement an AVL tree. Since this part is a continuation of part 2 of the previous assignment, if you have attempted that part you just have to add balancing to your BST in this part. You need to implement the following additional functions:

- **node* rotateleft(node* p):** p is root of the tree before rotation, the function should return pointer to the new root of the tree, around which the subtrees were rotated. Remember to fix the heights of the subtrees.
- **node* rotateright(node* p):** Same as above.
- **node* balance(node* p):** Calls appropriate rotations depending upon the four cases discussed in the class.
- **int balanceFactor(node* p):** Calculates and returns the balance factor of the node p based upon the heights of left and right subtrees.
- **void fixHeight(node* p):** Calculates new height of the tree rooted at p

Note: For the other functions, you may use the code you wrote for implementing a bst in the previous assignment. Feel free to add any helper functions you like.

Please ensure your code compiles properly on a **Linux** environment. You may use test1.cpp to verify the correctness of your code.

Q2. In this part, you'll implement a dictionary using BST & AVL trees and compare the time taken by both to search words. We have provided you a file "words.txt" containing dictionary words in random order. All the words are in lower-case. You have to load (insert) these words in BST & AVL trees. The words themselves will act as keys. You'll have to implement the following functions:

- **Dictionary():** Creates an empty (i.e. without words) Dictionary object.
- **~Dictionary():** Frees the memory occupied by nodes containing words.
- **void initialize(string wordsFile):** Populates words read from wordsFile into wordsTree, wordsFile is the name of the file.
- **node* findWord(string word):** Corresponds to node* search(T k).
- **bool deleteWord(string word):** Corresponds to void delete_node(T k).
- **bool editWord(string oldWord, string newWord):** Changes the key. Therefore, the tree may need to be adjusted to maintain BST property.
- **bool insertWord(string word):** Corresponds to void insert(string val, T k).

Note: The skeleton code has been provided to you in the folder part-2. You may replace bst.h and bst.cpp in this folder with your implementation of BST and copy your AVL implementation from part 1 to the part 2 folder. Feel free to add any helper functions you like.

Please ensure your code compiles properly on a **Linux** environment. You may use test1.cpp to verify the correctness of your code.

Comparison:

Select 26 words such that each one starts from a distinct alphabet.
Report you results in a pdf file in the following table format:

#	Word	Time(sec) take by BST (t1)	Time(sec) take by AVL (t2)	Difference (dt=t1-t2)	Percentage decrease ((dt/t1)*100)

Submission Guidelines:

Submit a **zip** folder containing the following files on your lms Assignment 3 tab:

- bst.h
- bst.cpp
- avl.h
- avl.cpp
- dictionary.cpp
- dictionary.h
- words.txt
- Comparison.pdf

Naming Convention: A3_*your roll number*.zip