Design and Analysis of Algorithms

Lab

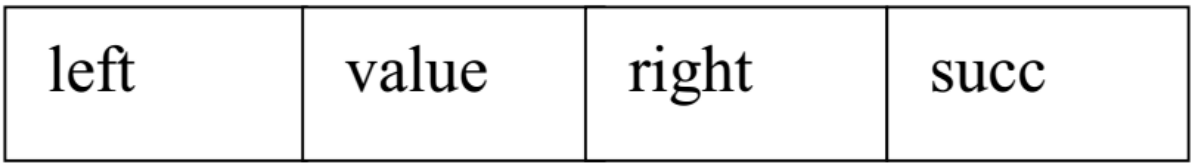
Lab Assignment: 2 Dated: 16/02/2018

Tasks to perform (in a programming language of your choice without using the inbuilt functions):

Problem 1

A Binary Search Tree (**BST**) is a binary tree in which all nodes in the left sub-tree are less than the current node, and all node in the right sub-tree are equal or more than the current node. In this lab, you are expected to create a **BST** tree of integers stored in an input file. For this problem, assume that all the values in the input are unique.

Node structure:



Here, value is the element stored in the **BST, and** left and right are the pointers to left and right subtrees. succ pointer points to the in-order successor of the current node.

Balancing mechanism:

A simple mechanism to balance the height of the tree: delete element from a longer subtree, push an element to smaller subtree, and adjust root accordingly.

Type definitions: (**tree.h**)

The structure **BST** contains the definitions for the node structure indicated above.

List of Functions to implement:

1. Common list of functions like insert, delete, find.

2. Use a balance mechanism given to balance the tree.

3. Perform comparative evaluation of balanced and unbalanced BST

Problem 2

Given a large text, create an index of words appearing in the text, along with the location(s) in which these words appear.

Data source for this problem: you may use any e-book.

Algorithm Outline:

1. Sanitize the text to remove all characters that are not alphabets or white spaces.

2. Maintain a data structure to handle index.

3. Traverse the text word by word.

4. If it is a new word,

- add to index (along with location),

5. else

- search the word in index and add the extra location.

6. Print the Index to a file ***(index.txt)*** indicating word and the location(s) where the word is found.

Note: use man fseek to know how to find position or go to a position

Approach to follow:

a. AVL tree of words.
b. Maintain the index as an AVL tree.
c. You may use the node structure of Problem1 for this.
d. Final index is an in-order traversal. (follow the nodes like a linked list, by using the succ pointer instead of the next pointer of traditional linked lists.)