

BITS Pilani, Pilani Campus  
2<sup>nd</sup> Sem. 2020-21  
CS F211 Data Structures & Algorithms

=====

Lab X

=====

**Topics:** Binary Search Trees and AVL Trees

Exercise 1: [Expected Time: 5 + 40 + 15 + 15 + 25 = 105 minutes]

a) Define a tree node with four fields:

- i. a value,
- ii. a pointer to the left sub-tree
- iii. a pointer to the right sub-tree and
- iv. height balance information, which is:
  - a) negative if the left subtree is taller,
  - b) positive if the right subtree is taller, and
  - c) zero if the subtrees are of the same height.

[Hint: Use bit-fields in struct so that minimum number of bits can be stored.  
E.g. struct { int x : 2; int y; } will direct a C compiler to assign two bits of storage for integer x. End of Hint.]

b) Implement the binary search tree operations without balancing the height:

- a. *add*
- b. *find* and
- c. *delete* [Hint: If the value to be deleted is in a leaf node it can be freed; if it is not in a leaf node, then find the in-order successor, say *s*, and copy the value of *s* into this internal node. Then *s* is available for deletion and the same procedure can be applied recursively. End of Hint]

c) Implement the rotate operation of AVL tree such that **rotate(bt, X, Y, Z)**:

Please read and understand the contents on the following URL before moving ahead:  
<http://people.cs.ksu.edu/~schmidt/300s05/Lectures/Week13.html>

- a. orders **X**, **Y**, and **Z** as **a**, **b**, and **c**,
  - b. identifies the other children of **X**, **Y**, and **Z** as **T0**, **T1**, **T2**, and **T3** in left-to-right order
- and then balances **bt** by
- a. replacing **Z** with **b** [Hint: This would require Z's parent. You may use an additional parameter to the procedure if necessary passing the parent. End of Hint.]
  - b. setting **a** and **c** as the left and right children – respectively – of **b**
  - c. setting **T0** and **T1** as the left and right children – respectively – of **a**
  - d. setting **T2** and **T3** as the left and right children – respectively – of **c**
- and returns the modified tree.

d) Modify the *add* operation such that it:

- a. identifies the point of imbalance and
- b. invokes *rotate* with right parameters for height-balancing the binary tree.

- e) Modify the *delete* operation such that it:
- Identifies the first point of imbalance
  - invokes rotate with right parameter for height-balancing that sub-tree
  - and repeats a. and b until the root of the binary tree is balanced.

Exercise 2: [Expected Time: 15+30 = 45 minutes]

- [Rank Query]: Implement an inorder traversal operation on binary search trees such that **inorder(bt, K)** returns the Kth smallest element in **bt**.
- [Range Query]: Implement a rangeSearch procedure that given a binary search tree **bt**, and range **K1..K2** of values, finds all the records in **bt** with keys in the given range. The algorithm would be based on deciding where the key of root value, say r.k, falls with respect to the range:
  - If  $r.k > K2$  then (recursively) search for the same range in the left subtree
  - If  $r.k < K1$  then (recursively) search for the same range in the right subtree.
  - If  $K1 \leq r.k \leq K2$  then:
    - Search for  $K1..(r.k)$  in the left subtree
    - Include r.k in the result
    - Search for  $(r.k)..K2$  in the right subtree

The result must be either accumulated in a non-local data structure or accumulated in a local data structure and returned from the procedure.

Assignment to be uploaded:

Solve Exercise-1 and upload its solution. Your uploaded file must contain a file “complete.c” which includes all the functions implemented by you.