COP5536 - Adv Data Structures Project 1

Name: Namita Namita

UFID: 48479313

email: namita.namita@ufl.edu

Problem Description

AVL tree is a self-balancing Binary Search Tree (BST) where the difference between heights of left and right subtrees cannot be more than one for all nodes. In this project, you're asked to develop and test a small AVL Tree (i.e. the entire tree resides in the main memory). The data is given in the form (key) with no duplicates, you are required to implement an AVL Tree to store the key values. Your implementation should support the following operations:

1. Initialize (): create a new AVL tree
2. Insert (key)
3. Delete (key)
4. Search (key): returns the key if present in the tree else NULL
5. Search (key1, key2): returns keys that are in the range key1 ≤key ≤key2

Run the code

- Please unzip the file
- Run the command below in the directory of my project
- $make
- $java avltree input_file_name
- Then, the input file will be passed into my program.
- Then the output.txt file will be generated in the same directory.

Program Structure and Function prototypes:

AvlNode class:

AvlNode class is the class for creating an AVL tree with integer info, left child, and right child pointers.

- **Data fields:**
    - info: Data component of the tree.
    - height: Holds the height of the tree relative to a specific node.
    - left: Holds a pointer to the left child of a specific node.
    - right: Holds a pointer to the right child of a specific node.
- **Functions:**
    - **public AvlNode():** Default constructor to initialize the object of AvlNode.
    - **public AvlNode(int d):** Parameterized constructor for setting up info of the object node.

avltree class:

avltree class is the class for creating, inserting, deleting, and searching elements in the avl tree.

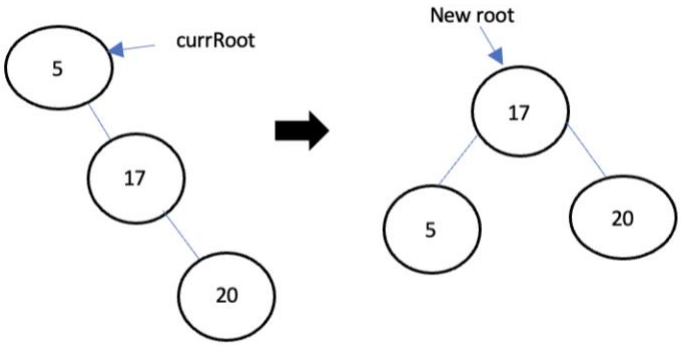
- **Data fields:**
    - root: Holds the pointer to the root of the tree.
    - List<Integer> nodeList: Holds the list of keys found within a range.
- **Functions:**
    - **public static void main (String[] args):** Main function takes "input.txt" as the input while executing and parses the entire file till EOF. Based on the string present in the "input.txt" file, the main function calls "initialize, insert, delete or find" functions and does the designated operations on the AVL tree.

      The output is written in the "output.txt" file and is appended every time there is an execution of the main function.

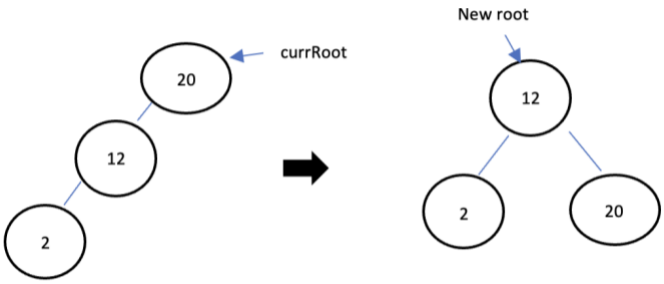
        - Functions called from main:
            - avltree(); - To perform Initialize function.
            - avlTree.insertingNode(avlTree.root, val); - To perform insert operation.
            - avlTree.deletingNode(avlTree.root, val); - To perform delete operation.
            - avlTree.findNodeInRange(avlTree.root, val1, val2); - Search nodes within a certain range.

- avlTree.findNode(avlTree.root, val); - Search for a node with info “val” in the tree.

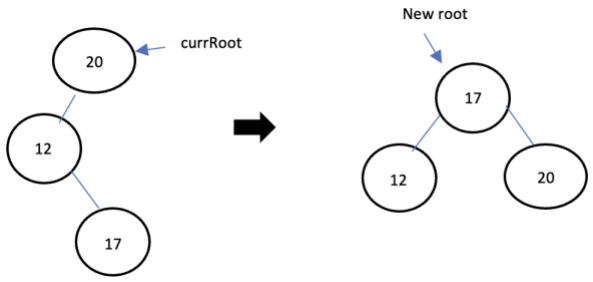
- **<u>AvlNode insertingNode(AvlNode currRoot, int info)</u>:** A method for adding a new node with value "info" to the Avl tree rooted at "currRoot". It inserts the new node at the left or right subtree of the “currRoot” based on the value present in the info field of the new node to be added and modifies the height of the tree. Then, if the "balanceFactor" of the nodes is determined to be different from -1,0,1, it performs one of the following cases:
  - ***RR case*** **–** if (balanceFactor<- 1 && info > currRoot.right.info)
    - Rotate left w.r.t currRoot
    - RR demonstration:



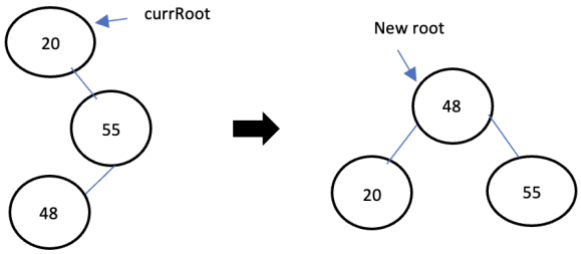
  - ***LL case*** - if (balanceFactor >1 && info < currRoot.left.info)
    - Rotate right w.r.t currRoot
    - LL demonstration:



  - ***LR case*** **-** if (balanceFactor > 1 && info > currRoot.left.info)
    - Rotate left followed by right rotation w.r.t currRoot
    - LR demonstration:



- ***RL case*** - if (balanceFactor < -1 && info < currRoot.left.info)
  - Rotate right followed by left rotation w.r.t currRoot
  - RL demonstration:



- Functions called from insertingNode:
  - AvlNode(info)- To add a node with value as info when the tree is empty.
  - findMaximum(findHeightOfTree(currRoot.left),findHeightOfTree(currRoot.right));- To initially find the height of the left and right subtree and then find max height.
  - findBalanceFactor(currRoot);- To find the balance factor.
  - rightRotate(currRoot);- For RR, LR, and RL cases.
  - leftRotate(currRoot);- For LL, RL, and LR cases.

- **AvlNode deletingNode(AvlNode currRoot, int info):** A method for deleting an existing node with the value "info" to the Avl tree rooted at "currRoot". It deletes the node by comparing its “info” field with the left and right subtree rooted at “currRoot”. Then it searches for the node with a single child or no children, if found, follows that path or takes the minimum from both of the children, deletes the in-order successor, updates the tree's height, and looks for any imbalances.
  - ***LL case*** - if (balanceFactor > 1 && getBalanceFactor(currRoot.leftChild) >= 0)
    - Rotate right w.r.t currRoot
  - ***RR case*** - if (balanceFactor<-1 && getBalanceFactor(currRoot.leftChild) < =0)

    - Rotate left w.r.t currRoot
  - ***LR case*** **-** if (balanceFactor > 1 && getBalanceFactor(currRoot.leftChild) < 0)
    - Rotate left followed by right rotation w.r.t currRoot
  - ***RL case*** **-** if (balanceFactor <-1 && getBalanceFactor(currRoot.leftChild) > 0)
    - Rotate right followed by left rotation w.r.t currRoot
- Functions called from deletingNode:
  - findMinValueNode(currRoot.right);- Find node with least value.
  - findMaximum(findHeightOfTree(currRoot.left),findHeightOfTree(currRoot.right));- To initially find the height of the left and right subtree and then find max height.
  - findBalanceFactor(currRoot);- To find the balance factor.
  - rightRotate(currRoot);- For RR, LR, and RL cases.
  - leftRotate(currRoot);- For LL, RL, and LR cases.

- **int findBalanceFactor (AvlNode avlNode):** Function to find out the balance factor of the "avlNode". It is done by calculating the difference between the height of its left and right subtree. A node is not balanced if the value of its balance factor is anything except -1,0,1.
  - Functions called from findBalanceFactor:
    - findHeightOfTree(currRoot.left);- To find the height of the left and right subtree.
- **int findHeightOfTree(AvlNode currRoot):** Function to find the height of the tree when it is rooted at "currRoot".
- **AvlNode leftRotate(AvlNode avlCurrNode):** A function to change the tree by undergoing left rotation on the tree rooted at "avlCurrNode" by making the rightchild "right" as new root and root "avlCurrNode" as the left child "left" of the new root. Returns the new root.
  - Functions called from leftRotate:
    - findMaximum(findHeightOfTree(currRoot.left),findHeightOfTree(currRoot.right));- To initially find the height of the left and right subtree and then find max height.
- **AvlNode rotateRight(AvlNode avlCurrNode):** A function to change the tree by undergoing left rotation on the tree rooted at "avlCurrNode" by making the left child "left" as new root and root "avlCurrNode" as the right child "right" of the new root. Returns the new root.

- Functions called from leftRotate:
    - findMaximum(findHeightOfTree(currRoot.left),findHeightOfTree(currRoot.right));- To initially find the height of the left and right subtree and then find max height.

- **AvlNode findMinValueNode (AvlNode avlNode):** Function to find out the node with minimum value by traversing downwards to the left subtree rooted at "avlNode".
- **int findMaximum(int x, int y):** Function to find out the maximum between two numbers "x" & "y".
- **AvlNode findNode(AvlNode avlNode, int info):** A function to search a node with the value "info" in the tree rooted at "avlNode". The flow goes to the left or right subtree by comparing info with the info field of the root. If info > "avlNode.info" then right subtree else left subtree. It returns the key found else null.
- **List<Integer> findNodeInRange(AvlNode currRoot, int r1, int r2):** A function to search all the nodes whose values lie between the range [r1,r2]. It determines whether to take the left subtree or right subtree path by comparing the "info" with the root. If info< "currRoot.info" then left subtree else right subtree. Nodes found within the range are added to a list and that list is returned to the calling function.