COP 5536 – Advanced Data Structures – Assignment 1

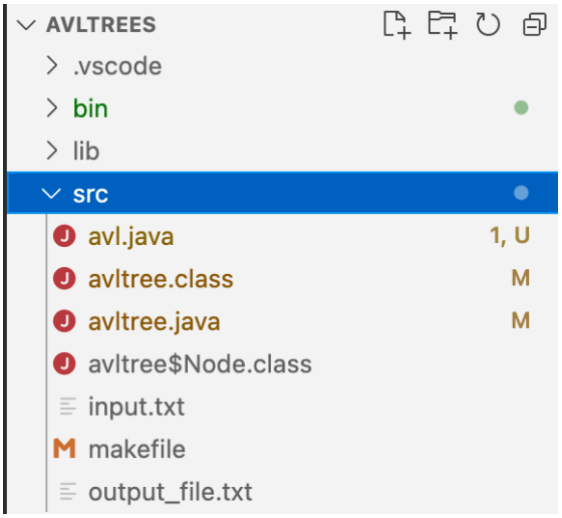
Name: Dhanush Pakanati, /*
UFID: 28079405,
UF mail: dpakanati@ufl.edu.

Steps to run the code:

1. Unzip the compressed file
2. Open the directory and run the make command: $make
3. Run the executable file generated with input file name: $java avltree input
4. Output will be written into the output_file.txt

File Structure of the project directory:

1. avltree.java: Contains class for node declaration, insertion, deletion, rotations, balancing, find and search in a range.
2. MakeFile: File describing the set of tasks that are to be executed to run the files in this directory. Compiles the avltree.java file and saves the class file in same directory.
3. Input.txt: A text file consisting of all the operations that are to be performed on the AVL tree.



The Classes and Methods used in this code are:

avltree.java:

Node Definition:

```java
//1. Write class to initialize node definition
public static class Node{
    //node with left and right child
    Node leftNode;
    Node rightNode;
    int key;
    //each node has height parameter - Distance to leaf
    int height;
    Node(int val){
        this.key  = val;
    }
}
```

Methods: Steps followed while coding:

```java
/*
Author: Dhanush Pakanati
*/
import java.io.*;
import java.util.ArrayList;
public class avltree {
    //1. Write class to initialize node definition
    public static class Node{…
    //we should always have access to root - even after balancing.
    //so having global variable accessible only in this class
    private static Node root;
    //2. write function for insert
    public static void insert(int key){…
    public static Node insert(int key, Node node){…
    //3. write function for rebalance
    //need not be accessible to other files - declaring private
    //will return the root node reference
    private static Node balanceTree(Node node){…
    //4. write function for right rotation(RR)
    private static Node rRight(Node node){…
    //5. write function for left rotation(LL)
    private static  Node rLeft(Node node){…
    //6. get height
    private static void calculateHeight(Node node){…
    private static int getHeight(Node node){…
    //7. calculate the balance factor of that node
    private static int balanceFactor(Node node){…
    //8. write function to search node
    public static Node find(int val){…
    //9. write function to delete
    public static void delete(int key){…
    private static Node delete(Node node, int key){…
    //10. Write function to find left most child
    private static Node leftMostChild(Node node){…
    //11. write function to search in a range
    private static void search(int k1, int k2, ArrayList<Integer> list){…
    private static void searchRange(Node node, int key1, int key2, ArrayList<Integer> list){…
    public static void initialize(){…
    Run | Debug
    public static void main(String[] args) throws IOException{…
}
```

The main operations needed in the assignment requirement:

1. Insert(int element)
2. Delete(int element)
3. Search (int element1, int element2)
4. Search (int element)

Methods Used to achieve the above functionality:

1. Insert: Find the right position to insert the node and then perform rebalance operations if necessary based on AVL tree rules.
2. balanceTree: balances the tree after each insertion and deletion
3. rRight, rLeft : Rotate right and rotate left functions.
4. calculateHeight : Returns the distance from node to farthest leaf.
5. GetHeight: Returns the height of the node.
6. balanceFactor: Calculates the balance factor at that node based on the height of left and right subtrees.
7. Find: returns the node if found in the tree
8. Delete:
9. leftMostChild : Finds the left most child from a given node. Used for replacement in delete method.
10. Search: returns the list of all nodes present in that interval.
11. Search Range: Finds all the nodes in that given range through in order traversal. The results include the boundary values.
12. Initialize: Initialize the node class.

Running the program:

Run $make clean to remove the older output_file.txt and .class files.
Run $make to compile avltree.java
Run the executable avltree.class file with input file name as parameter - $java avltree input
Output will be written to output.txt. To read output use the command - $vim output_file.txt

```
thunder:~/src> make clean
rm -f *.class output_file.txt
thunder:~/src> ls
avl.java  avltree.java  input.txt  makefile
thunder:~/src> make
javac -g avltree.java
lsthunder:~/src> ls
 avl.java              avltree.class   input.txt
'avltree$Node.class'   avltree.java    makefile
thunder:~/src> java avltree input
```

```
dhanush — ssh dpakanati@thu...
1897
23, 65
NULL
NULL
32
NULL
~
~
~
~
~
                    1,1           All
```