COP5536: Advanced Data Structures Programming Project Report: AVL Tree

Tre' R. Jeter
UFID: 1946-9876
Email: t.jeter@ufl.edu
November 9, 2022

Project Description

An 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, we're asked to develop and test a small AVL Tree (i.e., the entire tree resides in main memory). The data is given in the form *key* with no duplicates and we we're 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 *key*1 ≤ key ≤ *key*2

Compiling Instructions

The project has been compiled and tested in Eclipse and on thunder.cise.ufl.edu with multiple versions of my own test files, including the one given as an example with the **`javac`** compiler. After unzipping my zip folder **`Jeter_Tre`**, there will be a folder called **`Jeter_Tre`** containing all the files needed to run the program. Below is how I executed the program on thunder.

Executing the Program:

1.) Remotely access the thunder server using **`ssh [username]@thunder.cise.ufl.edu`**
2.) Under **`AVLTree`**, direct yourself to the **`src`** folder where the following files should be:
    a. avltree.java
    b. Makefile
    c. avl.txt
    d. test1.txt
    e. test2.txt
    f. test3.txt
3.) To compile, simply run **`make`** (Makefile is made of a generic ***`javac -g`*** command)
    a. Class files will be generated and can be seen with a simple **`ls`**
4.) After successful compilation, run **`java avltree [name of file to test]`**
    a. A file named *`output_file.txt`* will be generated with the expected output and can be seen with any command line text editor (vim, nano, emacs, etc.)
5.) To remove the class files generated, simply run **`make clean`**
6.) To remove the output_file.txt, simply run **`rm output_file.txt`**, however if ran again with a new test file it should just be overwritten with the new expected output
7.) Repeat the above process to test more files

Structure of Project

This program is made up of the *avltree* class that encloses the *Node* class.

The function in the *Node* class is defined as follows:

- *getKey()*: gets a specific key value of type int and returns the key value if possible

public int getKey()

The constructor in the *Node* class is defined as follows:

- *Node(int key)*: depicts a Node
    - A key value helps distinguish many attributes about a Node (height, left child, and right child)

public Node(int key)

Utility functions used within the Node class are defined as follows:

- *height()*: returns the height of AVL Tree from specific point

int height(Node point)

- *rightRotate()*: returns the new root of the AVL Tree after a right rotate

Node rightRotate(Node point)

- *leftRotate()*: returns the new root of the AVL Tree after a left rotate

Node leftRotate(Node point)

- *balanceTree()*: returns a balanced version of the previous AVL Tree

int balanceTree(Node point)

- *insert()*: inserts a node with Binary Search Tree insertion
    - updates the AVL Tree's height
    - balances the resulting AVL Tree
    - returns the new AVL Tree with node inserted

Node insert(Node point, int key)

- *minNodeKey()*: returns the node with the smallest key

Node minNodeKey(Node point)

- *delete()*: deletes a node with Binary Search Tree deletion
    - updates the AVL Tree's height
    - balances the resulting AVL Tree
    - returns the new AVL Tree with requested node deleted

Node delete(Node point, int key)

The functions in the *avltree* class are defined as follows:

- *search()*: searching for the existence of a key in the AVL Tree
    - key: any given key to be found (or not)
    - returns a found key, null if the requested key is not found

public String search(int key)

- *rangeSearchHelp()*: helper function for rangeSearch()
    - Parameters: a list of keys, a node, and two key values
        - keys: list of key values
        - node: a specific node
        - key1: any given key value as the starting point of a range search
        - key2: any given key value as the ending point of a range search
    - Keys falling between the given range are recursively added to the list of keys

Public void rangeSearchHelp(List<Integer> keys, Node node, int key1, int key2)

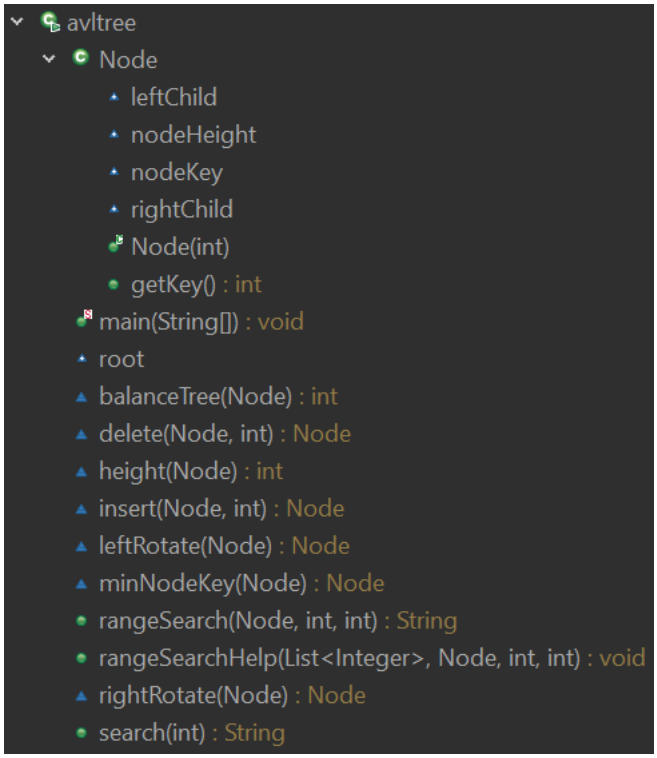
- *rangeSearch()*: given a node and two key values, returns the values (inclusive) between the two key values given
    - Parameters: a node and two key values
        - node: a specific node
        - key1: any given key value as the starting point of a range search
        - key2: any given key value as the ending point of a range search
    - Returns a range of keys if found, null if keys are not found

public String rangeSearch(Node node, int key1, int key2)

- *main()*: function for reading the text file commands
    - FileNotFoundException is thrown if a file isn't found

public static void main(String [] args) throws FileNotFoundException

The image below depicts the structure of the program in the Eclipse IDE.



Running the Program on Thunder

Viewing every file that is stored in AVLTree/src

```
thunder:~/AVLTree/src> ls
avltree.java  avl.txt  Makefile  test1.txt  test2.txt  test3.txt
thunder:~/AVLTree/src>
```

View of the Makefile

```
default:
	javac -g avltree.java

clean:
	$(RM) *.class
~
```

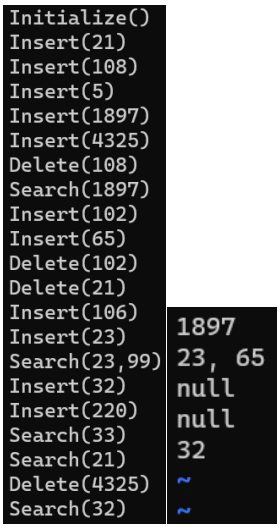
Running the make command to compile the avltree.java file and generate the class files

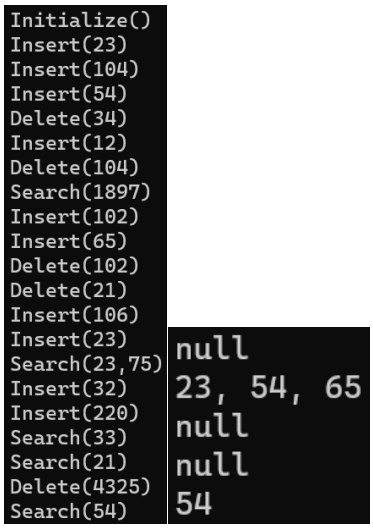
```
thunder:~/AVLTree/src> make
javac -g avltree.java
thunder:~/AVLTree/src> ls
'avltree$Node.class'   avltree.java   Makefile    test2.txt
 avltree.class         avl.txt        test1.txt   test3.txt
thunder:~/AVLTree/src>
```

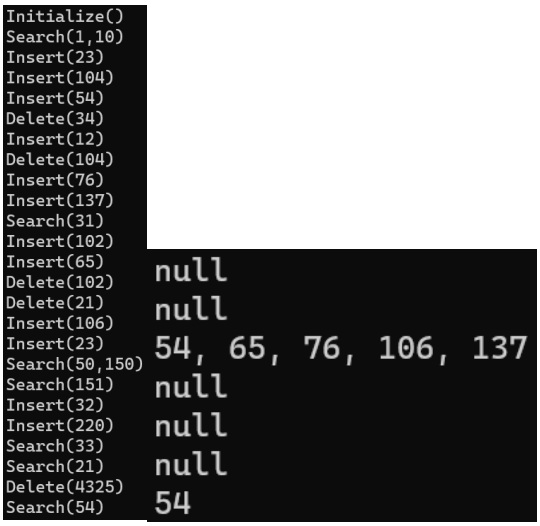
Running the java command to run the avltree program and generate the output_file.txt

```
thunder:~/AVLTree/src> java avltree avl.txt
thunder:~/AVLTree/src> ls
'avltree$Node.class'   avltree.java   Makefile          test1.txt   test3.txt
 avltree.class         avl.txt        output_file.txt   test2.txt
thunder:~/AVLTree/src>
```

Example outputs from the given example within the description of the assignment and self-generated test files. ***From left to right:*** avl.txt, output, test1.txt, output, test2.txt, output, test3.txt, and output.







Conclusion

This program was designed to imitate an Adelson-Velsky and Landis Tree (AVL Tree). It successfully implements functionalities like insertion, deletion, and search. Other functions are thrown in for help such as the rangeSearchHelp function that assists with searching for keys within a range of two given key values. Others like height, minNodeKey, balanceTree, leftRotate, and rightRotate are useful utility functions to properly execute the main AVL Tree functionalities. I can confirm that each method is properly implemented and the AVL Tree data structure within this program functions as expected.