EEM 480 Homework-Bonus-Report

The homework was developed in Apache NetBeans IDE 12.0 using JDK8-301. The total hwBonus_isik package contains 4 Java classes with main class. Which are AVLNode.java, AVLTree.java, ReadandParse.java and main.java.

In the main class there is a "AVLHandler" object declaration from AVLTree. With the "AVLHandler" object I called method which is processFile(string pathname). In the data structure, there is a balanced binary search tree list. AVLTree is a binary tree that holds "idNr, name and surname" values. Construction of the tree is made with idNr. In the below you can find further explanations for the classes I used. In this homework I used lecture slides for understanding the algorithm behind trees. Also, I referenced the websites at the end of this report which are very useful.

1. `public class AVLNode`

In the constructor block AVLNode is created with the parameters idNr, name and surname values. Left and Right values holds AVLNode initialized as null then it will be set in the further process. Height value also holds in the nodes each node has value that is the height of the current node in the tree. setName() method declared but never used.

2. `public class AVLTree`

In the constructor block root is created from AVLNode and initializes as null. String searchpath and Boolean found values declared. Searchpath is used for concatenating the visited nodes to a string and found is for Boolean flag that marks the search is complete and proceed the code.

a) `int getBalance(AVLNode N)`

   This method calculates the balance between left child and right child and returns the value Θ (1).

b) `private int max(int leftheight, int rightheight)`

   This method compares the left node's height and right node's height chooses the highest number and returns the value Θ (1).

c) `private int height(AVLNode t )`

   This method calculates the height of the with the given algorithm given in the slides. Basically, it returns -1 if the node has no child Θ (1).

d) `private AVLNode findMin(AVLNode node)`

This method iterates in the left subtree to find minimum value. AVLTree structure is inorder traversal structure so the leftist node is the minimum node. Loop iterates until finding the min value so it iterates Θ (n).

e) `private AVLNode rotatetotheright(AVLNode A)`

This method does the swap operation between nodes if the tree or subtree is left heavy rightrotate operation balances the tree. Balancing the tree changes height value so that these values get update. Θ (1).

f) `private AVLNode rotatetotheleft(AVLNode B)`

This method does the swap operation between nodes if the tree or subtree is right heavy leftrotate operation balances the tree. Balancing the tree changes height value so that these values get update. Θ (1).

g) `public void insertStudent(Integer idNr, String name, String surname)`

This method is used for sending the root value only for first iteration. Polymorphism used here. This method also called in the ReadandParse java file.

h)
```
public AVLNode insertStudent(Integer idNr,String name, String
surname, AVLNode node)
```

This method inserts a new node and updates the tree to ensure it is balanced. The selected node that will be put in the tree will be compared with the root in the first iteration. Searches the node in the tree that is ordered as left to right in ascending order. When the comparison done then it will be placed in appropriate position. Balance will be checked in every recursive operation and the rotations will be made accordingly with balance. Complexity of the time required is Θ (log n) for lookup, plus a maximum of Θ (log n) retracing levels. So, it is Θ (log n).

i) `public void deleteStudent(int idNr)`

This method is used for sending the root value only for first iteration. Polymorphism used here. This method also called in the ReadandParse java file.

j) `public AVLNode deleteStudent(AVLNode root, int idNr)`

This method deletes a new node and updates the tree to ensure it is balanced. The selected node that will be deleted in the tree will be compared with the root in the first iteration. Searches the node in the tree that is ordered as left to right in ascending order. When the comparison done the desired node with the given idNr will be checked if it has any child. then it will be placed in appropriate position. Balance will be checked in every recursive operation and the rotations will be made accordingly with balance. Complexity of the time required is Θ (log n) for lookup, plus a maximum of Θ (log n) retracing levels. So it is Θ(log n).

k) `private AVLNode searchStudent(AVLNode r, int idNr)`

Searches the node in the tree that is ordered as left to right in ascending order. Finds the asked node with their idNr. Basically, the method is recursive and compares the idNr with the current node every iteration if the asked node's idNr is smaller than the reference node than it will go to the left and updates the search_path. If it is bigger than the reference node it will go right and updates the search_path. If it finds the asked node updates the search_path string with "Found". If it could not find anything function calls itself with the changed reference node. Θ(log n).

l) `public void processFile(String filename)`

This method creates parser object from ReadandParse class. This object used for calling the function named studentParser().Θ (1).

m) `public void printSearchPath(int idNr)`

This method calls the search function with the given index. If found flag is true prints the visiting way otherwise it print "The student with idNr:"+idNr+" is not existed in the system". Θ (1).

n) `public void printwithinorder()`

This method is a handler for recursive function with the same name. Polymorphism used. Θ (1).

o) `private void printwithinorder(AVLNode r)`

This method is used for debug purposes it iterates in the tree according to in order traversal. Prints the value that it visits Θ (n).

3. `public class ReadandParse`

In the constructor block myfile and reader objects are created from File and Scanner with the given file path parameter.

a) `private String get_line()`

Uses the reader object to use nextLine method. It gets the lines one by one and return the data every time the method called. There is no loop in this method so that complexity is Θ(1)

b) `public void studentParser()`

This method creates AVLHandler object for using the avl methods in this class. Regular string splitting operations done. First line reserved for the number of insertions and deletions. Other lines get inside for loop the given numbers in first line. After insertion and deletion operations done, infinite while loop starts if the user enters '0' value in console the code terminates itself otherwise searchs the given integer idNumber in the AVLTree.

References

1. https://www.cs.usfca.edu/~galles/visualization/AVLtree.html For understanding how rotating operations work and how does the deletion operation effect the tree.
2. https://www.tutorialspoint.com/data_structures_algorithms/avl_tree_algorithm.htm For understanding the rotation operations again.
3. https://en.wikipedia.org/wiki/AVL_tree
4. https://www.youtube.com/watch?v=jDM6_TnYIqE Youtube lecture for insertion operations