Binary Trees

Introduction

The problem to be solved in this assignment is to design and implement a program which uses a binary search tree data structure albeit without using any if the built in Java collection classes. The implementation of this program should include some use of recursion within it as well. Similar to the “linked list” assignment, each node in the Binary Search Tree must hold the details of a student(namely their ID and say an exam mark) which will be stored as nodes in the binary tree. The program should also manage to allow users to enter information for each node, also getting inputs to let them execute specific tasks within the Binary Tree such as deleting/searching for nodes.

Requirements

1. **Allow the user to enter details for a student – the program will add this to a binary tree.** The requirement was met and the program allows for the user to choose the values for each node as required in the binary tree.
2. **Allow the user to request that the whole tree be printed in numerical order of student id.** This requirement was tackled and a function was made to allow users to print the Binary Tree In-order, Which worked as expected and would run if the user wanted.
3. **3. Allow the user to enter a student id to look for in the tree and show the exam mark for that student (or display a message if the student does not appear in the tree).** The search/query requirement was met and program allowed for the user to enter an ID, referencing that with the binary tree and checking to see if a node matched with the same ID as given.
4. **Allow the user to request that a particular student is removed from the tree (or display a message if the student does not appear in the tree) .** This requirement was met so that both options are achieved. The program allows for users to input an ID referencing that to the Binary Tree, deleting if found. If not, informing the user that it was not found
5. **[Optional] Write out your tree to a text file, and then read it in again at the start of the program .** I did not manage to get into implementing/doing this requirement.
6. **[Optional] Implement other forms of traversal for the tree, e.g. do a preorder and postorder traversal to display the tree contents.**  This requirement was tackled and met as on top of the Inorder traversal for the display function. It was implemented so that users can decide from 3 different ways of traversing specifically Preorder and Postorder as well.
7. **[Optional] Read up on how to balance a binary tree and implement a practical solution to improve the balance of your binary tree. Note that you should design and write up the algorithm you intend to implement first before coding it! Be aware that this is a very complex feature to implement.** This requirement was not tackled and implemented into the program.

Design

Class: TreeNode

Fields: left, right, ID, mark

Method: setid,getid,setleft,setright,setmark,getmark, getright,getleft.

Class: Tree

Fields: TreeNode

Method: add, getroot, setroot, findintree, delete, setroot, printin,printpost,printpre

DEF printInorder(Treenode):

IF node == null:

RETURN

END IF

printInorder(Treenode.left)

printInorder(Treenode.right)

DEF add(TreeNode, ID):

Found = null

IF ID < Treenode.ID:

IF TreeNode.left != null:

add(Treenode.left, ID)

END IF

ELSE:

TreeNode.left = new TreeNode(ID)

END ELSE

END IF

ELSE IF ID > TreeNode.ID:

IF TreeNode.right != null:

Add(TreeNode.right, ID)

END IF

ELSE

TreeNode.right = new TreeNode(ID)

END ELSE

END ELSE IF

DEF find(TreeNode,ID):

TreeNode found = null

IF TreeNode == null:

Return null

END IF

IF TreeNode.ID == ID:

Return root

END IF

IF TreeNode.right != null:

Found = find(TreeNode.right, ID)

END IF

IF Found == null:

Found = find(TreeNode.left, ID)

END IF

Return Found

DEF delete(root, ID):

IF root == null:

Return null

END IF

IF ID < root.ID:

root.left = delete(root.left, ID)

END IF

ELSE IF ID > root.ID:

root.right = delete(root.right, ID)

END ELSE IF

ELSE

IF root.left == null:

Return root.right

END IF

ELSE

IF root.right == null:

Return root.left

END IF

END ELSE

Root.ID = SETNEWROOT

Root.right = delete(root.right, root.ID)

END ELSE

Return root

Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Description | Test Data Set | Reason | Expected output | Work as expected |
| Add multiple nodes to a tree. Print in order. | ID - 5,3,7,5,8 | normal | Nodes are added to binary tree then printed inorder | Y |
| Add multiple nodes to a tree. Print postorder. | ID - 5,3,7,5,8 | normal | Nodes are added to binary tree then printed postorder | Y |
| Add multiple nodes to a tree. Print preorder. | ID - 5,3,7,5,8 | normal | Nodes are added to binary tree then printed preorder | Y |
| Add multiple nodes to a tree.  Delete 5 | ID - 5,3,7,5,8 | normal | Nodes are added. 5 is deleted. New root is set. | Y |
| Add multiple nodes to a tree. Search for item | ID - 5,3,7,5,8  Searchitem - 3 | normal | Nodes are added. 3 is searched for. Outputs that 3 is in binary tree, | Y |
| Binary tree main Menu option. | Menu - k | exceptional | Incorrect input, ask for re-input | Y |
| Add node | ID = h,1 | Exceptional | Incorrect input, asks for re-input, asks for new node details | Y |
| Add nodes to a tree.  Delete 9 | ID - 5 | Extreme | Nodes are added. 9 isnt in tree so is not deleted. Message saying item isn’t in tree is outputted | Y |

Evaluation

I found nothing hard to program in terms of getting things to work/make sense, the thing I found difficult was more figuring out how to get the classes to work together.

The biggest issue I had was identifying why a function worked in one class and not another. In this case, the delete function would work in the tree class but not in the tester class. After thinking through for a while I noticed I made a mistake making a field static so one thing which may have been an issue is concerning the visibility of functions/fields.

The main challenges for me were identifying/designing appropriate algorithms to work in java recursively to achieve what is needed. I was having difficulty in figuring how I was to implement the recursion in this function specifically. With the add function, I implemented a way to add nodes manually but was unsure as to how I would do it recursively, given the parent and child nodes and direction. After looking over my old python recursive functions such as an old recursive quick sort I implemented, I made sense of how I would implement the recursive functions. I noticed that similarly to how a Quick sort partitions, the same pattern is needed to happen in this program which made implementing it a lot easier to do and understand. After the add function was implemented, this made the delete function much easier to program as they more or less used the same concept/way of utilising recursion.

The strengths of this project is that the basic required parts of the assignment are programmed well and run efficiently and well. The weakness is that the extra parts of the program such as the balancing tress requirement and the files part has not been able to be implemented.