Assignment 3 - Binary Trees

(Shop)

Introduction

This report presents the development of a Java shop program that utilises a binary search tree (BST) data structure for efficient management of a shop's inventory. The aim of this assignment was to create a user-friendly application that allows users to add, delete, search, and print items within the shop, while also enabling the storage of item data in a text file for persistence and easy retrieval.

The shop program leverages the power of a binary search tree to organise and retrieve item information effectively. With the binary search tree's hierarchical structure, operations such as searching for specific items based on name or attributes, adding new items, and deleting existing ones are performed efficiently. Furthermore, the program offers flexibility by allowing users to print the item list using pre-order, post-order, or in-order traversal methods, enabling different perspectives on inventory organization.

The program's user interface is designed to be intuitive, providing menu-driven options for seamless interaction with the shop's inventory. The incorporation of a text file-saving feature ensures that item data can be stored and retrieved easily, eliminating the need to recreate the stock from scratch with each program launch.

In summary, this report showcases the development of a Java shop program that employs a binary search tree for efficient management of a shop's inventory. The program's user-friendly interface facilitates various operations on the inventory, including addition, deletion, searching, and printing. Integrating a text file-saving mechanism enhances data persistence and retrieval capabilities.

The subsequent sections of this report will delve into the implementation details, algorithms utilized, encountered challenges, and potential future enhancements to further improve the shop program's functionality and performance.

Requirements

Functional

- The program should allow users to enter details of an item and then add it to a binary tree.
- The program should be able to print the item list in numerical order of item id.
- The program should be able to search for an item in the list.
- The program should be able to calculate the total cost of all the items in the shop.
- The program should allow users to remove a particular item.

Non-functional:

- The program should save the item list in a text file and be able to read it again while running.
- The program should implement preorder and postorder traversals.
- The program should keep track of stocks
- The binary tree should be self-balancing

My program includes all of the above functional and non-functional requirements. I did implement the balance binary tree but it is not integrated in the shop itself. I have a method in the binary search tree class which allows you to balance any binary tree when you pass its root node in the method.

Design

Working of the program:

The program first prints a menu which allows the user to perform the tasks. When the user chooses to add an item they can enter the details of the item and add it. When the user chooses to delete or search they have to put in the item id of the item and then it gets deleted or searched. When the user chooses to print the item list they can choose in which way they want to print it (preorder, postorder, in order). The user can see the total cost of the items in their shop from the calculate cost option and the user can exit and save the program by exit option.

Pseudo code for adding in the binary tree:

```
if the root is null:
      create a new node with the given data and make it the root of the tree
   else if data is less than the root's data:
         recursively call add with the left child of the root and the given data
   else if data is greater than the root's data:
         recursively call add with the right child of the root and the given data
   end if
```

Pseudo code for deleting in the binary tree:

```
if the root is null:
    return root (nothing to delete)
else if data is less than the root.data:
    set root's left child as the result of recursively calling deleteNode with root's left child and the data
else if data is greater than the root.data:
    set root's right child as the result of recursively calling deleteNode with root's right child and the data
else:
    if the root has no left child:
        set root's right child as the new root
        delete the current root
    else if root has no right child:
        set root's left child as the new root
        delete the current root
    else:
        find the minimum value node in the right subtree of the root
        replace the root's data with the minimum value
        recursively call deleteNode with the root's right child and the minimum value
end if
return root
```

Pseudo code for checking if a node exists in the binary tree:

```
if root is null:
    return false (the tree is empty)
else if data is equal to root.data:
    return true (found a match)
else if data is less than root.data:
    recursively call containsNode with root's left child and the data
else if data is greater than root.data:
    recursively call containsNode with root's right child and the data
```

Pseudo code for preorder traversal in the binary tree:

```
if root is null:
    return (base case, empty tree)
else:
    visit root (process the data of the current node)
    recursively call preOrderTraversal with root's left child
    recursively call preOrderTraversal with root's right child
end if
```

Pseudo code for postorder traversal in the binary tree:

```
if root is null:
    return (base case, empty tree)
else:
    recursively call postOrderTraversal with root's left child
    recursively call postOrderTraversal with root's right child
    visit root (process the data of the current node)
end if
```

Pseudo code for in-order traversal in the binary tree:

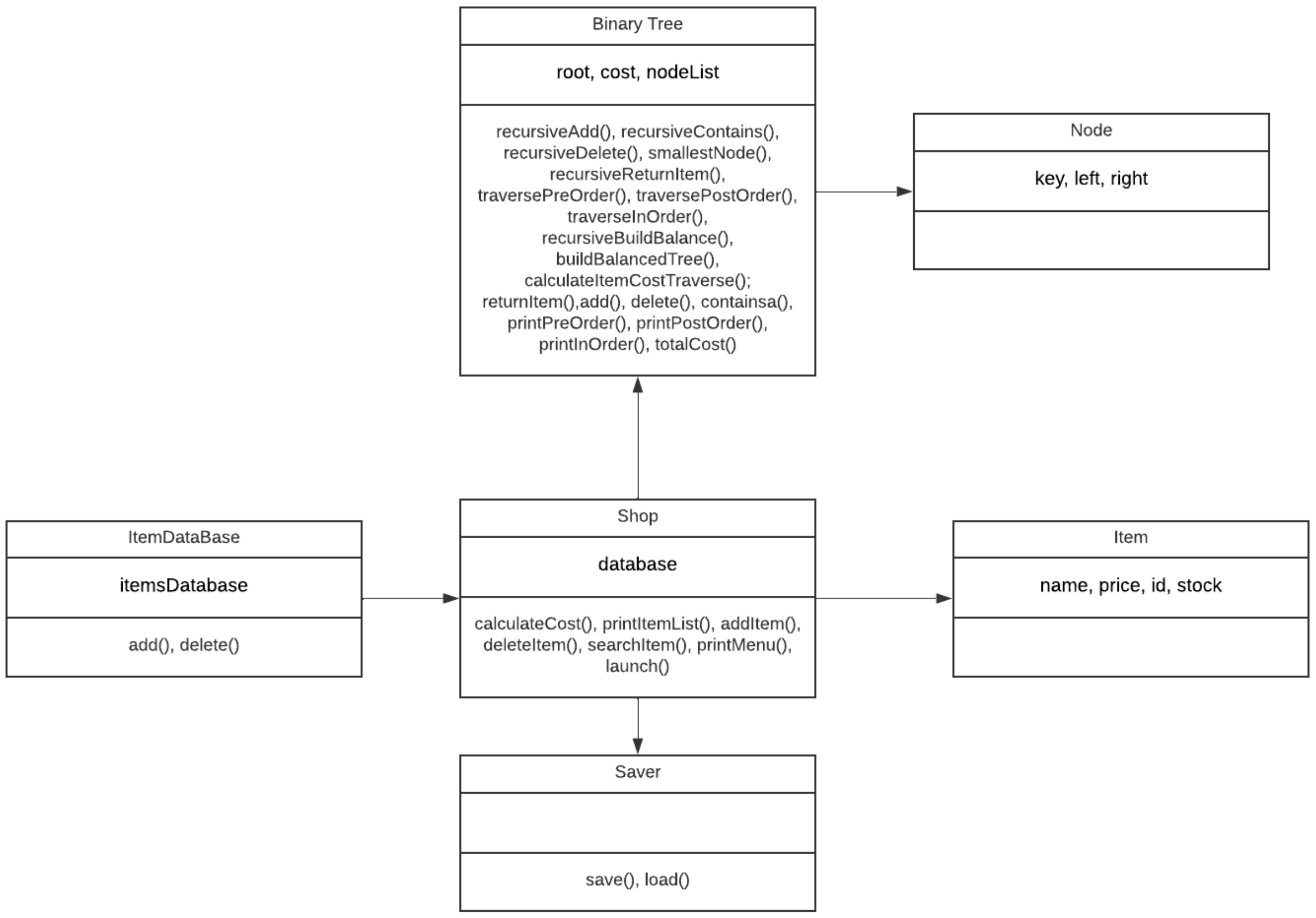
```
if root is null:
    return (base case, empty tree)
else:
    recursively call inOrderTraversal with root's left child
    visit root (process the data of the current node)
    recursively call inOrderTraversal with root's right child
end if
```

Pseudo code for balancing the binary tree:

1. Perform an in-order traversal of the original binary tree.
2. Store the values of the nodes in an ArrayList.
3. Create a new binary tree by recursively constructing balanced subtrees using the sorted list of nodes.
4. Find the middle node of the sorted list and set it as the root of the new binary tree.
5. Recursively repeat steps 3 and 4 for the left and right halves of the sorted list, respectively.
6. Set the left subtree of the root as the result of constructing a balanced tree from the left half of the sorted list.

7. Set the right subtree of the root as the result of constructing a balanced tree from the right half of the sorted list.
8. Return the root node of the newly constructed balanced binary tree.

Class diagram


Binary Tree
root, cost, nodeList
recursiveAdd(), recursiveContains(), recursiveDelete(), smallestNode(), recursiveReturnItem(), traversePreOrder(), traversePostOrder(), traverseInOrder(), recursiveBuildBalance(), buildBalancedTree(), calculateItemCostTraverse(); returnItem(),add(), delete(), containsa(), printPreOrder(), printPostOrder(), printInOrder(), totalCost()
Node
key, left, right
ItemDataBase
itemsDatabase
add(), delete()
Shop
database
calculateCost(), printItemList(), addItem(), deleteItem(), searchItem(), printMenu(), launch()
Item
name, price, id, stock
Saver
save(), load()

Test Plans and Results

All the test cases were made manually by me. I initialised a binary tree which is used in most of the test cases. Some of the tests like testInorder, testPreorder, and testPostorder were tested manually by printing the binary tree as I was not successful to test it with the built-in tester. And also the function responsible for the balance of the binary tree was tested by printing the preorder of its balanced form. The tester class includes all the test cases. The test sheet for the program is given below.

IC10039

Name: Sanskar Basnet
Matric number: 2540525
Lab Title: Assignment 3: Binary Tree
Test number/date/version: 26/05/2023
Test Notes: Tests run on the submitted assignment

The default BinaryTree bt = 6, 4, 8, 3, 5, 7, 9

Test Description	**Test Data**	**Expected result**	**Worked?**
Test add	6, 4	True	Y
Test Delete	6	False	Y
Test contains	6 4 2 9	True True True True	Y
Test InOrder	bt	3, 4, 5, 6, 7, 8, 9	Y
Test preOrder	bt	6, 4, 3, 5, 8, 7, 9	Y
Test postOrder	bt	3, 5, 4, 7, 9, 8, 6	Y
Test balanceTree	 10, 8, 7, 6, 5 4, 3, 2, 1, 5, 6, 7 4, 3, 2, 1	(in pre-order) 7, 5, 6, 8, 10 4, 2, 1, 3, 6, 5, 7 2, 1, 3, 4	Y

Self Evaluation

This was an enjoyable and challenging assignment. I encountered some bugs while developing the binary tree but when I looked at the lecture slides and did some research by myself I was able to fix the problems. The most challenging part of this assignment was developing the binary tree other than that the other functionalities were not hard to implement.

To Conclude, the assignment was fun to work on and the program met all the basic requirements in the assignment brief. All the test cases are passed and the program performs all the operations such as adding, deleting, searching, and printing items and it also saves the item list in a text file.