GTU Department of Computer Engineering

CSE 222/505 – Spring 2021

Homework 7 Report

ŞEYMA NUR CANBAZ

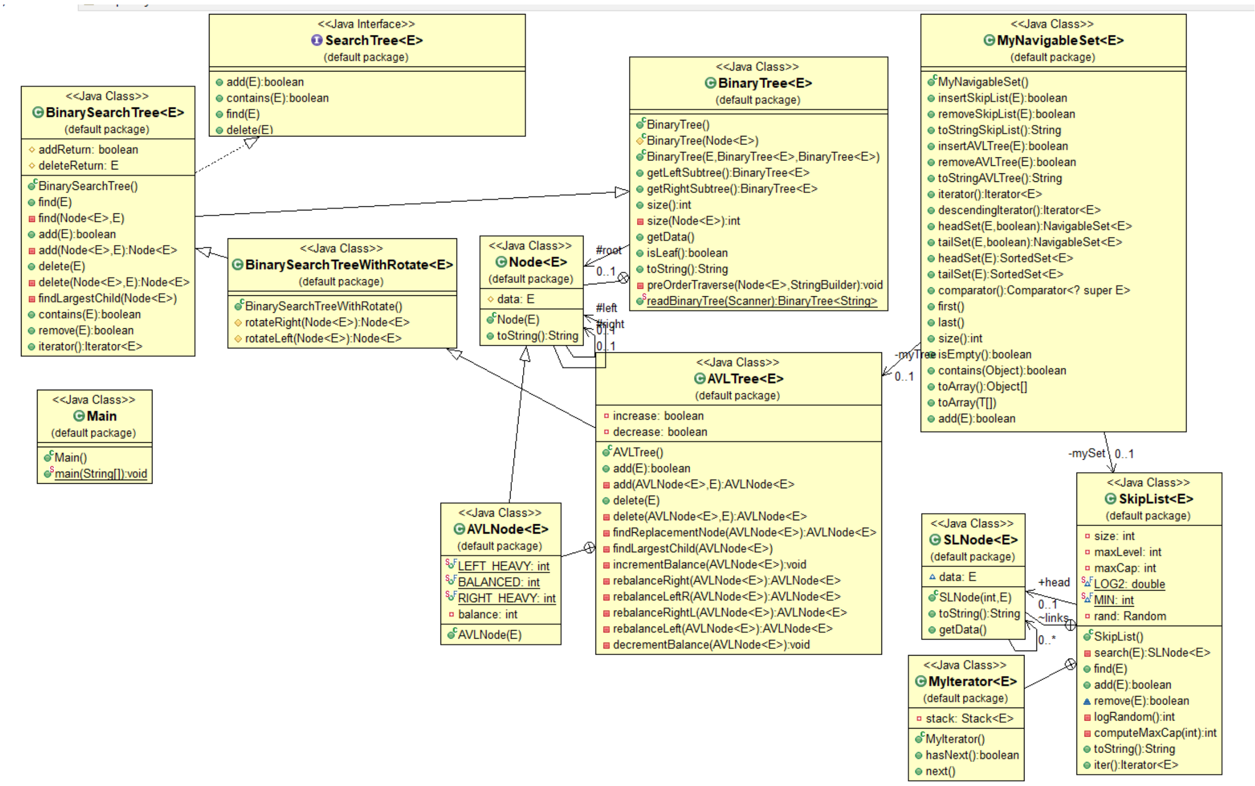
171044076

PART1

1. **System Requirement:**

Firstly, I created MyNavigableSet class. This class implemets the NavigableSet interface. Since we wanted to implement using SkipList and AVL tree, I kept SkipList (mySet) and AVLTree (as myTree) objects inside this class. In this class, I used the insert and delete method of SkipList. I needed to create an iteraor for the descendingIterator() method. Therefore I created 'MyIterator' class in SkipList class. Here I created a stack to iterate the data as descending and kept the data here.
For AVL tree, I override the headSet and tailSet methods of the navigableSet interface. I used AVLTree's iterator in these methods. I used the methods in the AVL tree class for insertion and iterator.

2. **Class Dİagram:**



3. Problem Solution Approach:

First of all, I created MyNavigableSet class. This class implements the NavigableSet interface. I kept the two object inside this class. One of them, mySet object. It is a skipList object. I wrote the insertSkipList and deleteSkipList methods for the NavigableSet using the skip list object in here. For this, I used the insert and remove methods of skip list. I created my own iterator class to write the descendingIterator() method. In order to access the data more easily as descending, I kept the data in the skip list as a stack here. In the Next() method, I returned the elements in the stack by popping them. With the hasNext() method, I returned true until the stack was empty.

I kept myList object for AVL tree. I used AVLTree's own insert method for insert. I override the headSet and tailSet method. For the headSet method, if the tree is not empty, I returned the subset from the beginning to the given element. If the flag is true, I returned the subset includes the elements given data. In the same way, I wrote the tailSet method. This time I returned the subset from the given element to the last element. I used AVL Tree's own iterator method for iterator() method. I also implemented the NavigableSet interface for OOP. However, I did not implement the other methods in the NavigableSet interface. I just implemented what was requested in the assignment. That's why I returned null, false, 0 in other methods.

4. Test Cases:

- **Skip list insert:** Inserts the given data to the skip list.
- **Skip list remove:** Removes the given item from the skip list.
- **Skip list remove (with non exist element):** If given data does not exist in the skip list, it throws NoSuchElementException.
- **Skip list descending iterator:** Iterates through elements in the skiplist as descending order.
- **AVL tree insert:** Inserts the given data to the AVL tree.
- **AVL tree iterator:** Iterates through elements in the AVL tree.
- **AVL tree headset:** Returns a view of the subset of this set whose elements are less than given element.
- **AVL tree headset (include given data):** Returns a view of the subset of this set whose elements are less than given element. If flag is true, the subset includes the elements toData if it exists.
- **AVL tree tailset:** Returns a view of the subset of this set whose elements are greater than given element.
- **AVL tree tailset (include given data):** Returns a view of the subset of this set whose elements are greater than given element. If flag is true, the subset includes the elements toData if it exists.
- **AVL tree remove:** Removes the given item from the AVL tree.

5. Running command and results:

-Skip list Insert 1 – 7 – 10 15 – 2

```
Skip list inserting:
Head: 3 --> 1 |1| --> 2 |1| --> 7 |1| --> 10 |2| --> 15 |3|
```

-Skip list remove 1 – 7

```
Skip list removing:
Head: 3 --> 2 |1| --> 10 |2| --> 15 |3|
```

-Skip list remove 3 (non – exist)

```
Skip list removing (non exist element):
java.util.NoSuchElementException
```

-Skip list descendingIterator

```
Skip list descending iterator: 15 10 2
```

-AVL tree insert 7 – 1 – 2 – 10 – 15

```
AVL Tree inserting:
7
1
2
10
15
```

-AVL tree iterator

```
AVL Tree iterator: 1 2 7 10 15
```

-AVL tree headSet (15)

```
AVL Tree headset:
1 2 7 10
```

-AVL tree headSet (15, true)

```
AVL Tree headset:
1 2 7 10 15
```

-AVL tree tailSet (2)

```
AVL Tree tailset:

7 10 15
```

-AVL tree tailSet (2, true)

```
AVL Tree tailset:

2 7 10 15
```

-AVL tree remove 1 – 7

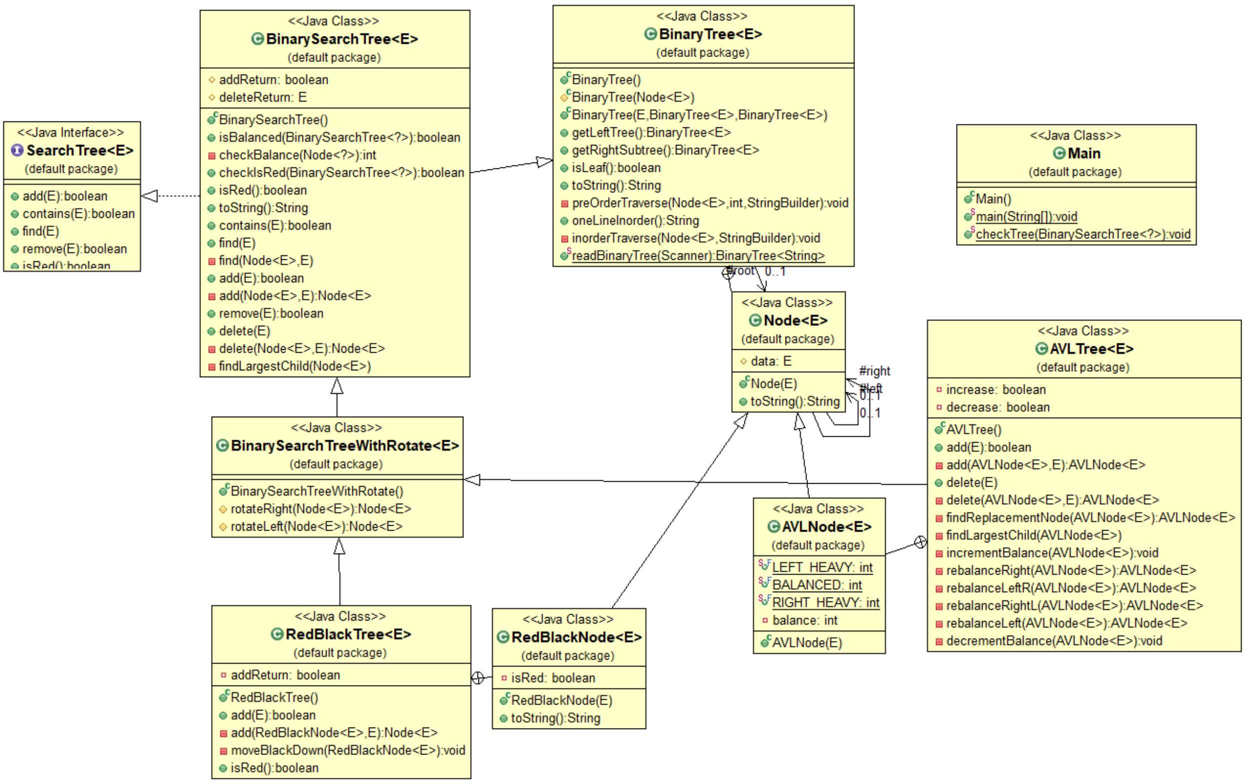
```
AVL Tree removing:
2
10
15
```

PART2

1. SYSTEM REQUIREMENTS:

I added the BinarySearchTree class to hold a Binary Search Tree object. Then I added these classes for AVLTree and RedBLackTree. This is how I created the class hierarchy, since all classes derive from the BinarySearchTree. The code works on Java 8.

2. CLASS DIAGRAMS:



3. PROBLEM SOLUTION APPROACH:

First, I created a method called checkTree() in the main class. This method takes binary search tree object as parameter. Then I wrote methods called isBalanced() and isRed() inside this method. If the given tree is an AVL tree, isBalanced() returns true and isRed() returns false. The isRed() method returns true if the given tree is a red black tree. If there is a tree that does not satisfy these conditions, it is a different tree. Based on these results, I printed which tree it was. In order to use these methods, I created isBalanced() and isRed() method in BinarySearchTree class. I wrote a helper method 'checkBalance()' for the isBalanced() method. This method takes a node as a parameter and calculates the height difference of the tree's left subtree and right subtree. Returns -1 if the height difference is greater than 1. Thus, it is understood that it is not balanced. Otherwise it is balance. I wrote the isRed() method on the searchTree interface. So I override this method in AVLTree and RedBlackTree classes. This method in AVLTree always returns false. In the RedblackTree class, if the given tree is not single-element, it will return true because there will be a red node. If it is a single-element tree, we cannot understand whether it is an AVL tree or a red black tree. Starting from here, I made the necessary distinction with the isRed() method in both classes.

4. TEST CASES:

- If given tree is AVL tree
- If given tree is Red Black Tree (balanced)
- If given tree is Red Black Tree (not balanced)
- I given tree is another (for example Binary Search Tree)

5. RUNNING COMMAND AND RESULTS:

- If given tree is AVL tree (insert 4, 3, 2)

```
AVLTree<Integer> avl = new AVLTree<Integer>();
avl.add(4);
avl.add(3);
avl.add(2);

checkTree(avl);
```

```
Balanced: true
isRed: false
This tree is an AVL Tree
```

- If given tree is Red Black Tree(insert 0, 1, 2, 6, 9, 10) - not balanced

```
RedBlackTree<Integer> rbt = new RedBlackTree<Integer>();
rbt.add(0);
rbt.add(1);
rbt.add(2);
rbt.add(6);
rbt.add(9);
rbt.add(10);
```

```
Balanced: false
isRed: true
This tree is a Red Black Tree
```

- If given tree is Red Black Tree (insert 4, 3, 2) – balanced

```
System.out.println("\n");
rbt.add(4);
rbt.add(3);
rbt.add(2);
```

```
Balanced: true
isRed: true
This tree is a Red Black Tree
```

- I given tree is another (Binary Search Tree – insert 6, 2, 1, 3)

```
BinarySearchTree<Integer> obj = new BinarySearchTree<Integer>();
obj.add(6);
obj.add(2);
obj.add(1);
obj.add(3);
checkTree(obj);
```

```
Balanced: false
isRed: false
This tree is not AVL Tree or Red Black Tree
```