Homework #7 - 1801042634

Student ID	1801042634
Student Name	Muhammed
Student Surname	Oğuz

Problem solutions approach

Part-1

NavigableSet with SkipList

For this section, I first took SkipList data structure from book. Than I create a class named `NavigableSetSkipList`. It implements navigableSet interface. So I overloaded all necessary methods to being NavgiableSet. After than, I implement expected methods only. `insert` `delete` `descendingIterator`. I Implement descending iterator with using SkipList iterator over ArrayList iterator. Finally reverse it with collections.sort method.

NavigableSet with AVL Tree

Firstly, I took AVLTree implementation from book. After that, I create a class named `NavigableSetAVL` class and implement it.
First overloaded all methods for implementing `NavigableSet` after, implement only expected methods.
`insert` `iterator` `headSet` `tailSet` .

For getting iterator. I changed AVLTree book implementation and added a iterator.
I also used a return value named `NavigableSetAVL` for headSet and tailSet methods

Part-2

Firstly I create a class named `TreType` . Has two static methods called `isRedBlackTree` and `isAVLTree` .

`isRedBlackTree` calls a BinarySearchTree's isRed method. If it returns `true` that means it is not a RedBlackTree, if returns `false` (means have a black node) it means it is a RedBlackTree.

`isAVLTree` cheks a BinarySearchTree if this three balanced like AVL tree, it return `true` . `False` otherwise.

Part-3

Firstly, I create a `ComparePerformance` class and add static methods called `add_X_10K` and `CalculateTime` methods.

`add_X_10K` adds a given $X * 10000$ new random item to given tree.
for example
`add_X_10K(4, tree)` adds new 40000 item to tree.
`calculateTime` calculates passing time with adding 100 new item to tree and return result.

Also, used BTree with order `50` and used BTree to use 2-3 Tree with order `3`

Test Cases

Part-1

TestNavigableSetSkipList

```
public static void testNavigableSetSkipList() {
        System.out.println("\n-----------Testing NavigableSetSkipList class-----------\n");
        System.out.println("This class implemented NavigableSet with skipList data structure");

        System.out.println("Create a NavigableSet with following values");
        System.out.println("5-51-15-4-57-12");
        NavigableSetSkipList<Integer> navigableSetSkipList = new NavigableSetSkipList<>();

        navigableSetSkipList.insert(5);
```

```
        navigableSetSkipList.insert(51);
        navigableSetSkipList.insert(15);
        navigableSetSkipList.insert(4);
        navigableSetSkipList.insert(57);
        navigableSetSkipList.insert(12);

        System.out.println("Print whole list");
        System.out.println(navigableSetSkipList);

        System.out.println("Delete 4 from set. Print removed item");
        System.out.println(navigableSetSkipList.delete(4));

        System.out.println("Delete 40 (non exist). Print removed item (if not exist print null)");
        System.out.println(navigableSetSkipList.delete(40));

        System.out.println("Create descendingIterator and print");
        Iterator<Integer> reverseIt = navigableSetSkipList.descendingIterator();

        while (reverseIt.hasNext())
            System.out.println(reverseIt.next());

        System.out.println("Delete 5 and Insert -10 and reCreate descendingIterator and print");
        navigableSetSkipList.delete(5);
        navigableSetSkipList.insert(-10);

        reverseIt = navigableSetSkipList.descendingIterator();

        while (reverseIt.hasNext())
            System.out.println(reverseIt.next());


        System.out.println("Lets try with some String values");
        System.out.println("Insert following values");
        System.out.println("Basak Hoca - Erdogan Hoca - Muhammed - Burak Hoca - '20' - '10' - HW08 - HW07");
        NavigableSetSkipList<String> navigableSetSkipListString = new NavigableSetSkipList<>();

        navigableSetSkipListString.insert("Basak Hoca");
        navigableSetSkipListString.insert("Erdogan Hoca");
        navigableSetSkipListString.insert("Muhammed");
        navigableSetSkipListString.insert("Burak Hoca");
        navigableSetSkipListString.insert("20");
        navigableSetSkipListString.insert("10");
        navigableSetSkipListString.insert("HW08");
        navigableSetSkipListString.insert("HW07");

        System.out.println("Print new NavigableSet");
        System.out.println(navigableSetSkipListString);

        System.out.println("Delete Muhammed and Print removed item");
        System.out.println(navigableSetSkipListString.delete("Muhammed"));
        System.out.println("Delete CSE222 (not exit) and Print removed item (if not exit print null)");
        System.out.println(navigableSetSkipListString.delete("CSE222"));

        System.out.println("Create descendingIterator and print");
        Iterator<String> reverseStringIterator = navigableSetSkipListString.descendingIterator();

        while (reverseStringIterator.hasNext())
            System.out.println(reverseStringIterator.next());

        System.out.println("End of NavigableSetSkipList. Thanks for Testing :)");

    }
```

TestNavigableSetAVL

This test almost same with TestNavigableSetSkipList but additionally tests headSet and tailSet methods.

```
public static void testNavigableSetAVL() {
        System.out.println("\n---------Testing NavigableSetAVL class---------");
        System.out.println("Insert following values to NavigableSetAVL Class");
        System.out.println("10-102-15-11-20-11 (try to add twice) -12");
        NavigableSetAVL<Integer> avlNavigable = new NavigableSetAVL<>();

        avlNavigable.insert(10);
        avlNavigable.insert(102);
        avlNavigable.insert(15);
        avlNavigable.insert(11);
        avlNavigable.insert(20);
        avlNavigable.insert(11);
        avlNavigable.insert(12);

        System.out.println("Print with AVL toString Style. Balance value with following data");
        System.out.println(avlNavigable);
```

```
        System.out.println("Create a iterator and print all values");

        Iterator<Integer> it = avlNavigable.iterator();

        while (it.hasNext())
            System.out.println(it.next());

        System.out.println("Delete 11 and 399 (not exist) and print their boolean return values");
        System.out.println(avlNavigable.delete(11));
        System.out.println(avlNavigable.delete(399));

        System.out.println("Insert new values (40-45-50-53-56-59-60) and print tailSet(50, true) and headSet(60) with AVLTree's toStri
        avlNavigable.insert(40);
        avlNavigable.insert(45);
        avlNavigable.insert(50);
        avlNavigable.insert(53);
        avlNavigable.insert(56);
        avlNavigable.insert(59);
        avlNavigable.insert(60);

        System.out.println(avlNavigable.tailSet(50, true));
        System.out.println(avlNavigable.headSet(60));

        System.out.println("Lets test with String Values");
        System.out.println("Insert Following Values");
        System.out.println("Basak Hoca - Erdogan Hoca - Burak Hoca - Muhammed - CSE222");
        NavigableSetAVL<String> avlNavigableString = new NavigableSetAVL<>();
        avlNavigableString.insert("Basak Hoca");
        avlNavigableString.insert("Erdogan Hoca");
        avlNavigableString.insert("Burak Hoca");
        avlNavigableString.insert("Muhammed");
        avlNavigableString.insert("CSE222");

        System.out.println("Delete Muhammed and HW07 (not exist) and print boolean results");
        System.out.println(avlNavigableString.delete("Muhammed"));
        System.out.println(avlNavigableString.delete("HW07"));

        System.out.println("Print with iterator");

        Iterator<String> itString = avlNavigableString.iterator();

        while (itString.hasNext())
            System.out.println(itString.next());

        System.out.println("End of NavigableSetAVL. Thanks for Testing :)");

    }
```

Part-2

Part-2 Test

This part shown how tested `TreeType` class's static methods

```
public static void testTreeTypes() {
        System.out.println("\n-----------Testing TypeTree class-----------\n");
        System.out.println("Create a AVL tree class and test it with following values");
        System.out.println("20-30-40-10-50");
        BinarySearchTree<Integer> avl = new AVLTree<>();
        avl.add(20); avl.add(30);
        avl.add(40); avl.add(10);
        avl.add(50);
        part2Helper(avl);

        System.out.println("Create a RedBlackTree and test it with following values");
        System.out.println("4-1-11-7-5-9");
        BinarySearchTree<Integer> rdb = new RedBlackTree<>();
        rdb.add(4); rdb.add(1);
        rdb.add(11); rdb.add(7);
        rdb.add(5); rdb.add(9);
        part2Helper(rdb);

        System.out.println("Lets add 10 and 11 to this redBlackTree and see what says our test");
        rdb.add(10); rdb.add(11);
        part2Helper(rdb);

        System.out.println("This shows, this tree is in same balance level of AVLTrees and a red black tree");

        System.out.println("\nLets try with an unbalanced BinarySearchTree");
        System.out.println("Initialize with following -> 10-20-30-40-5-3");
        BinarySearchTree<Integer> unbalanced = new BinarySearchTree<>();
```

```
        unbalanced.add(10); unbalanced.add(20);
        unbalanced.add(30); unbalanced.add(40);
        unbalanced.add(5); unbalanced.add(3);
        part2Helper(unbalanced);

        System.out.println("Lastly, send a BinarySearchTree but balanced with our entries");
        System.out.println("8-5-9-3-11");
        BinarySearchTree<Integer> balanced = new BinarySearchTree<>();
        balanced.add(8); balanced.add(5);
        balanced.add(9); balanced.add(3);
        balanced.add(11);
        part2Helper(balanced);

        System.out.println("End of TypeTree class. Thanks for Testing :)");
    }
```

Helper for Part-2 Test

```
@SuppressWarnings("rawtypes")
    public static void part2Helper(BinarySearchTree tree) {
        System.out.println("Printing tree");
        System.out.println(tree);
        boolean AVL = TreeType.isAVLTree(tree);
        boolean RDB = TreeType.isRedBlackTree(tree);

        if (AVL && RDB)
            System.out.println("This three both AVLTree (in balance like AVL) and RDB Tree\n");
        else if (AVL)
            System.out.println("This tree is AVLTree (in balance like AVL)\n");
        else if (RDB)
            System.out.println("This tree is RedBlackTree\n");
        else
            System.out.println("This tree neither AVL or RedBlackTree\n");
    }
```

Part-3

Part-3 code section a little bet in repeat. But all functions have different tree type, so it necessary.

Part-3 Test

```
public static void part3() {
        System.out.println("\nBinarySearchTree results");
        System.out.println("------------------------");
        part3Helper_BST(1);
        part3Helper_BST(2);
        part3Helper_BST(4);
        part3Helper_BST(8);

        System.out.println("\nRedBlackTree Results");
        System.out.println("------------------------");
        part3Helper_RDB(1);
        part3Helper_RDB(2);
        part3Helper_RDB(4);
        part3Helper_RDB(8);

        System.out.println("\n2-3 Tree Results - Used BTree as a Order = 3");
        System.out.println("------------------------");
        part3Helper_B(1, 3);
        part3Helper_B(2, 3);
        part3Helper_B(4, 3);
        part3Helper_B(8, 3);

        System.out.println("\nBTree results - Order = 50");
        System.out.println("------------------------");
        part3Helper_B(1, 50);
        part3Helper_B(2, 50);
        part3Helper_B(4, 50);
        part3Helper_B(8, 50);

        System.out.println("\nSkipList Results");
        System.out.println("------------------------");
        part3Helper_S(1);
        part3Helper_S(2);
        part3Helper_S(4);
        part3Helper_S(8);
    }
```

Part-3 Helper Function

All helper functions are likely very same. I will show only one of them

```
public static void part3Helper_BST(int k) {
        BinarySearchTree<Integer> bTree;
        long t = 0;
        for (int i = 0; i < 10; i++)
        {
            bTree = new BinarySearchTree<Integer>();
            ComparePerformance.add_X_10K(k, bTree);
            t += ComparePerformance.calculateTime(bTree);
        }
        System.out.println(k+"0k Items Performance " + t/10 + "ms");
    }
```

Running Command and Results

Part-1

NavigableSetSkipList Test Results

```
----------Testing NavigableSetSkipList class----------

This class implemented NavigableSet with skipList data structure
Create a NavigableSet with following values
5-51-15-4-57-12
Print whole list
[4, 5, 12, 15, 51, 57]
Delete 4 from set. Print removed item
4
Delete 40 (non exist). Print removed item (if not exist print null)
null
Create descendingIterator and print
57
51
15
12
5
Delete 5 and Insert -10 and reCreate descendingIterator and print
57
51
15
12
-10
```

```
Lets try with some String values
Insert following values
Basak Hoca - Erdogan Hoca - Muhammed - Burak Hoca - '20' - '10' - HW08 - HW07
Print new NavigableSet
[10, 20, Basak Hoca, Burak Hoca, Erdogan Hoca, HW07, HW08, Muhammed]
Delete Muhammed and Print removed item
Muhammed
Delete CSE222 (not exit) and Print removed item (if not exit print null)
null
Create descendingIterator and print
HW08
HW07
Erdogan Hoca
Burak Hoca
Basak Hoca
20
10
End of NavigableSetSkipList. Thanks for Testing :)
```

NavigableSetAVLTree Test Results

```
---------Testing NavigableSetAVL class---------
Insert following values to NavigableSetAVL Class
10-102-15-11-20-11 (try to add twice) -12
Print with AVL toString Style. Balance value with following data
0: 15
  0: 11
    0: 10
      null
      null
    0: 12
      null
      null
  -1: 102
    0: 20
      null
      null
    null

Create a iterator and print all values
10
11
12
15
20
102
Delete 11 and 399 (not exist) and print their boolean return values
true
false
```

```
Insert new values (40-45-50-53-56-59-60) and print tailSet(50, true) and headSet(60) with AVLTree's toString style
0: 59
  0: 53
    0: 50
      null
      null
    0: 56
      null
      null
  1: 60
    null
    0: 102
      null
      null

1: 20
  0: 12
    0: 10
      null
      null
    0: 15
      null
      null
  0: 53
    0: 45
      0: 40
        null
        null
      0: 50
        null
        null
    1: 56
      null
      0: 59
        null
        null
```

```
Lets test with String Values
Insert Following Values
Basak Hoca - Erdogan Hoca - Burak Hoca - Muhammed - CSE222
Delete Muhammed and HW07 (not exist) and print boolean results
true
false
Print with iterator
Basak Hoca
Burak Hoca
CSE222
Erdogan Hoca
End of NavigableSetAVL. Thanks for Testing :)
```

Part-2

Test a Tree created as an AVLTree class

```
Create a AVL tree class and test it with following values
20-30-40-10-50
Printing tree
0: 30
  -1: 20
    0: 10
      null
      null
    null
  1: 40
    null
    0: 50
      null
      null

This tree is AVLTree (in balance like AVL)
```

Test a tree created as a RedBlackTree class

```
Create a RedBlackTree and test it with following values
4-1-11-7-5-9
Printing tree
Black: 4
  Black: 1
    null
    null
  Red  : 7
    Black: 5
      null
      null
    Black: 11
      Red  : 9
        null
        null
      null

This tree is RedBlackTree
```

Add new values to this RDB tree to make this balanced like AVL

```
Lets add 10 and 11 to this redBlackTree and see what says our test
Printing tree
Black: 7
  Red  : 4
    Black: 1
      null
      null
    Black: 5
      null
      null
  Red  : 10
    Black: 9
      null
      null
    Black: 11
      null
      null

This three both AVLTree (in balance like AVL) and RDB Tree

This shows, this tree is in same balance level of AVLTrees and a red black tree
```

Test a Tree created as BST (inserted values to make unbalanced)

```
Lets try with an unbalanced BinarySearchTree
Initialize with following --> 10-20-30-40-5-3
Printing tree
10
  5
    3
      null
      null
    null
  20
    null
    30
      null
      40
        null
        null

This tree neither AVL or RedBlackTree
```

Test a BST (balanced with our entries)

```
Lastly, send a BinarySearchTree but balanced with our entries
8-5-9-3-11
Printing tree
8
  5
    3
      null
      null
    null
  9
    null
    11
      null
      null

This tree is AVLTree (in balance like AVL)

End of TypeTree class. Thanks for Testing :)
```

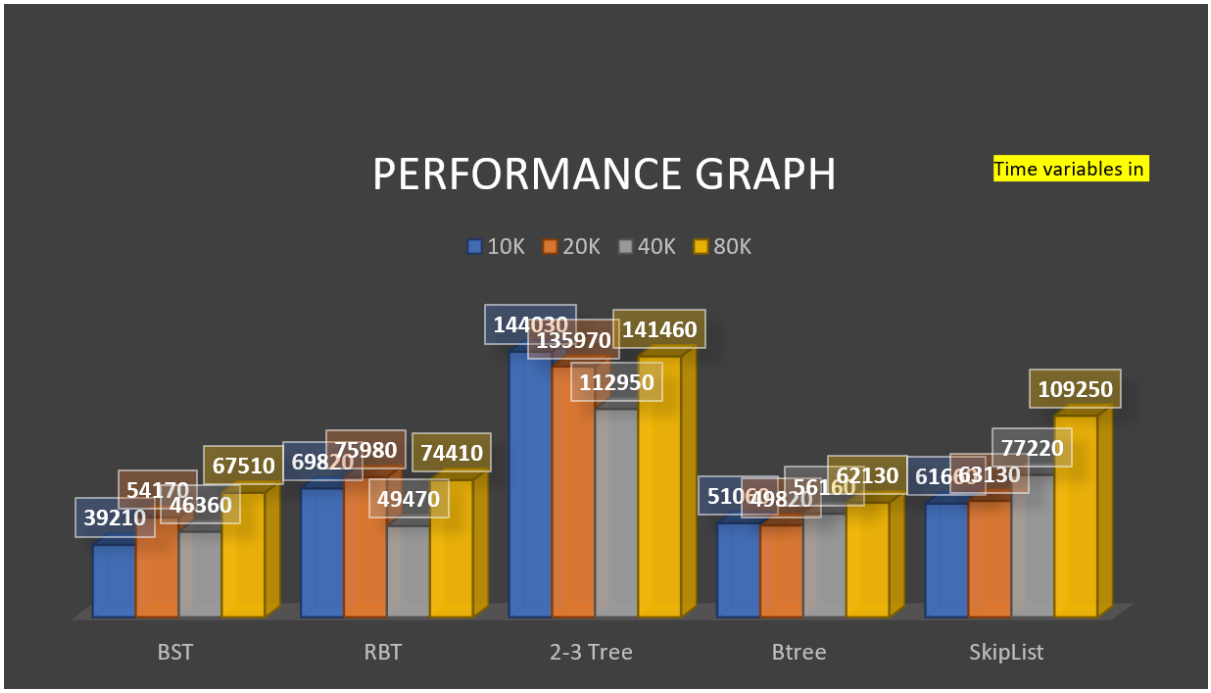
Part-3

Results in Terminal

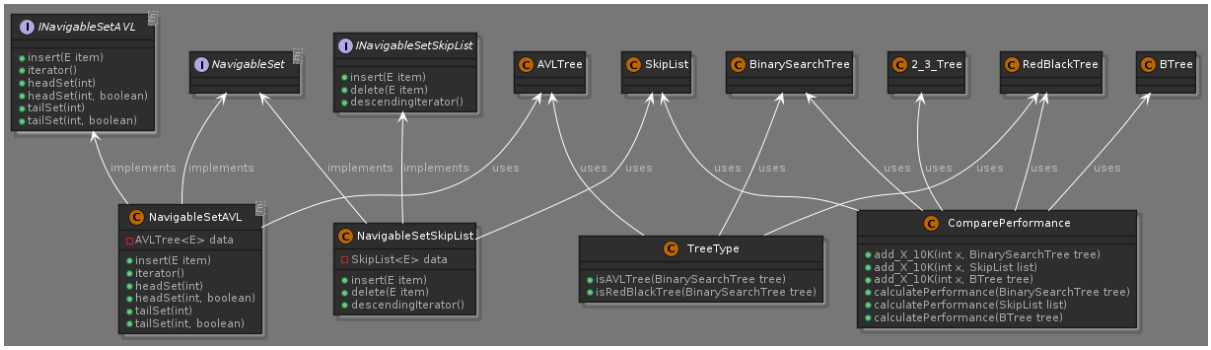
```
>>>>Test For PART-3<<<<


BinarySearchTree results
------------------------
10k Items Performance 48130ms
20k Items Performance 34510ms
40k Items Performance 40880ms
80k Items Performance 65730ms

RedBlackTree Results
------------------------
10k Items Performance 75740ms
20k Items Performance 48090ms
40k Items Performance 50680ms
80k Items Performance 60050ms

2-3 Tree Results - Used BTree as a Order = 3
------------------------
10k Items Performance 113610ms
20k Items Performance 98030ms
40k Items Performance 128210ms
80k Items Performance 171320ms
```

```
BTree results - Order = 50
_____________________________

10k Items Performance 45560ms
20k Items Performance 58370ms
40k Items Performance 55280ms
80k Items Performance 66660ms

SkipList Results
_____________________________

10k Items Performance 69810ms
20k Items Performance 72720ms
40k Items Performance 105060ms
80k Items Performance 128690ms

<<<<End Of Part-3 Tests>>>>
```

Performance Graph



Class Diagrams



HD version in folder