GEBZE TECHNICAL UNIVERSITY

COMPUTER ENGINEERING

CSE222-2021

HOMEWORK 07 REPORT

AHMET FURKAN KURBAN

1801042674

1.1 Problem Solution Approach

First of all for part1 i create navigableset avl and navigableset skiplist interface then i create class for it and using avl and skiplist object i implement necessary functions then i test it at main.

Part2-)

AvlCheck- I check avl tree balanced or not using height function.

RbtCheck-I check root is black or not then red node has black child then i check black node is same from given node to leafe node .If root is null i return true.

Part3-)

ı get implementations from book above

Binary search tree implementation
- Red-Black tree implementation
- 2-3 tree implementation
- B-tree implementation
- Skip list implementation

I Perform this operation 10 times for 10.000, 20.000, 40.000 and 80.000 random numbers .

I ınserted 100 extra random numbers into the structures I built and measure the running time
I Calculated the average running time for each data structure and problem size

1.2 Detailed System Requirements

AVL TREE (NavigableSet)

@Override

public boolean insert(E e) {

return sl.insert(e);

}

Insert element to the skiplist obje navigable set is interface and this functions are implemented.

@Override

public E delete(E e) {

return sl.delete(e);

}

Delete element from skiplist.

```
@Override
public Iterator<E> descendingIterator() {
	return sl.descendingIterator();
}
```

Print element from biggest element to lowest element using iterator.

```
@Override
public String toString() {
	return sl.toString();
}
```

Print skiplist elements.

AVL TREE (NavigableSet)

```
public boolean insert(E e) {
	return sl.add(e);
}
```

Insert elements to the avl tree using avl tree add function.

```
@Override
public Iterator<E> iterator() {
	return  sl.iterator();
}
```

Reach elements of avl tree using iterator.

```
@Override
```

```
public AVLTree<E> headSet(E e) {
        return sl.headSet(e);
}
```

Return head set of avl tree.

```
@Override
public AVLTree<E> tailSet(E e) {
        return sl.tailSet(e);
}
```

return tail set of avl tree.

```
public void print(){
        sl.print();
}
```

Print avl tree elements using special class.

---

public boolean checkAvlTree(BinarySearchTree<E> tree)

Check Tree is avl or not . Here we cast avl tree to the bst tree and

then i checked it is avl or not.

---

public boolean checkRBTTree(BinarySearchTree<E> tree)

Check Tree is rbt or not . Here we cast avl tree to the bst tree and

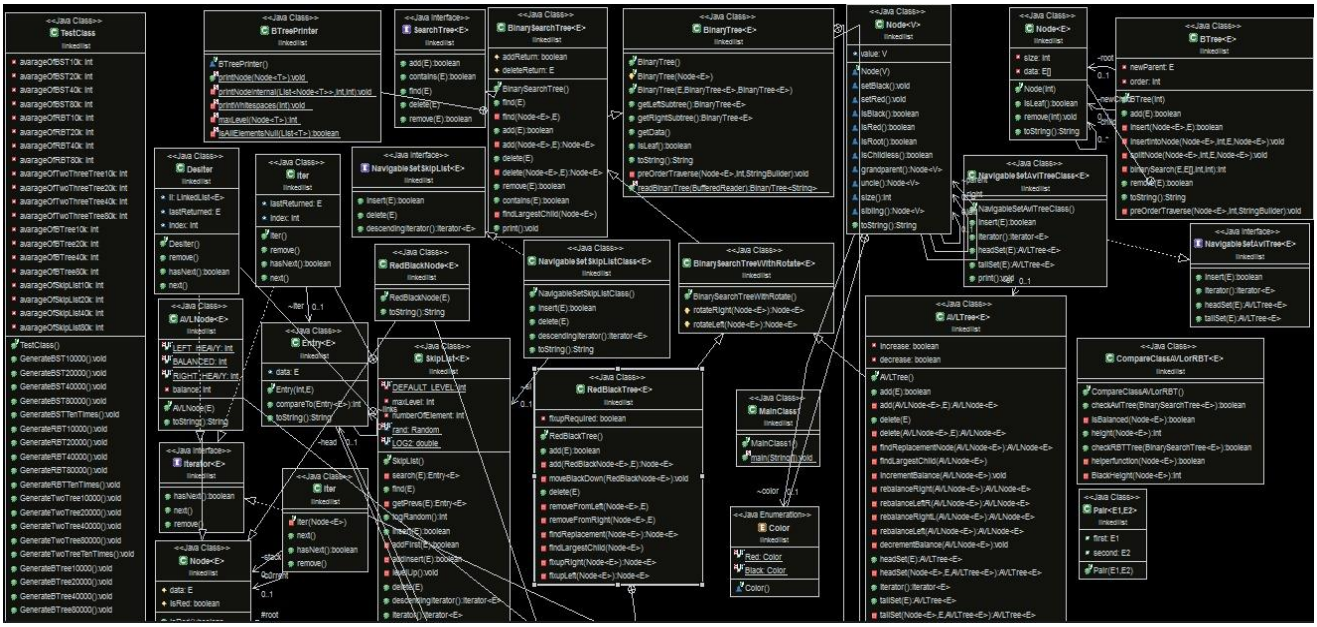
then i checked it is rbt or not.

---

public void StartTest()

This function start adding element for part3.

2.1 Class Diagram



3.1 Test Cases

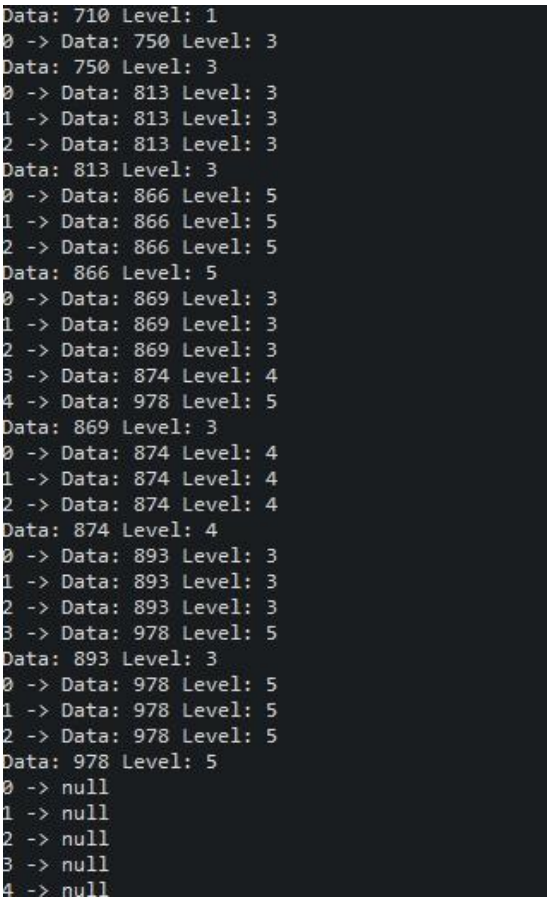
Part_1->

SkipList

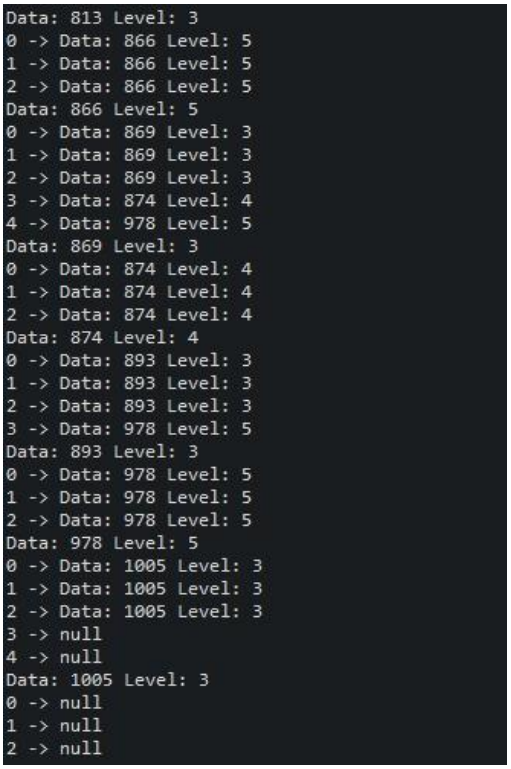
Insert 1:

```
Before Delete
Data: null Level: 5
0 -> Data: 5 Level: 5
1 -> Data: 5 Level: 5
2 -> Data: 5 Level: 5
3 -> Data: 5 Level: 5
4 -> Data: 5 Level: 5
Data: 5 Level: 5
0 -> Data: 23 Level: 3
1 -> Data: 23 Level: 3
2 -> Data: 23 Level: 3
3 -> Data: 116 Level: 4
4 -> Data: 234 Level: 5
Data: 23 Level: 3
0 -> Data: 24 Level: 2
1 -> Data: 24 Level: 2
2 -> Data: 116 Level: 4
Data: 24 Level: 2
0 -> Data: 100 Level: 2
1 -> Data: 100 Level: 2
Data: 100 Level: 2
0 -> Data: 116 Level: 4
1 -> Data: 116 Level: 4
Data: 116 Level: 4
0 -> Data: 226 Level: 4
1 -> Data: 226 Level: 4
2 -> Data: 226 Level: 4
3 -> Data: 226 Level: 4
Data: 226 Level: 4
0 -> Data: 234 Level: 5
1 -> Data: 234 Level: 5
2 -> Data: 234 Level: 5
3 -> Data: 234 Level: 5
Data: 234 Level: 5
0 -> Data: 475 Level: 3
```

Insert 2: (To see delete easily (1005))



Delete:

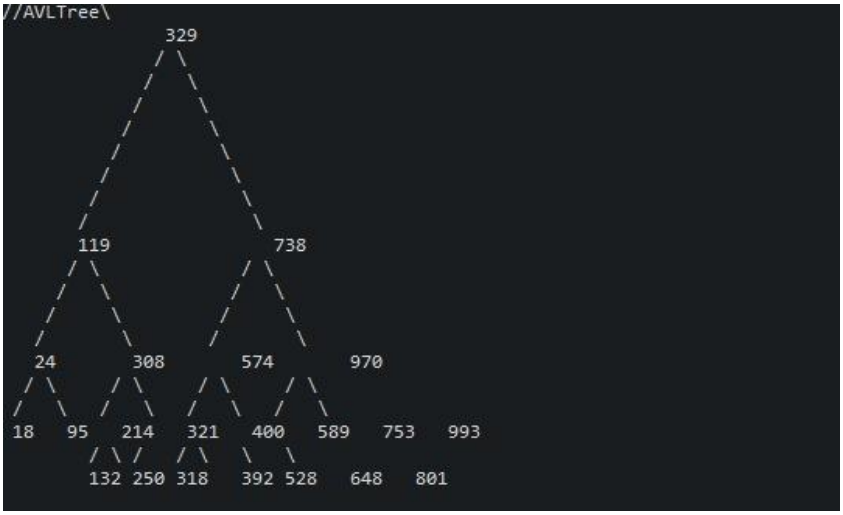


Descending Iterator

```
Descending Iterator
978 --> 893 --> 874 --> 869 --> 866 --> 813 --> 750 --> 710 --> 693 --> 662 --> 645 --> 627 --> 611 --> 475 --> 234 --> 226 --> 116 --> 100 --> 24 --> 23 -->
```

AvlTree

Insert



Iterator

```
Iterator
18 --> 24 --> 95 --> 119 --> 132 --> 214 --> 250 --> 308 --> 318 --> 321 --> 329 --> 392 --> 400 --> 528 --> 574 --> 589 --> 648 --> 738 --> 753 --> 801 --> 970 --> 993
```

HeadSet

```
HeadSet
0: 308
  0: 119
    0: 24
      0: 18
        null
        null
      0: 95
        null
        null
    0: 214
      0: 132
        null
        null
      0: 250
        null
        null
  1: 321
    0: 318
      null
      null
    1: 329
      null
      0: 392
        null
        null
```

TailSet

```
TailSet
0: 574
  0: 329
    0: 318
      0: 308
        null
        null
      0: 321
        null
        null
    0: 400
      0: 392
        null
        null
      0: 528
        null
        null
  0: 753
    0: 648
      0: 589
        null
        null
      0: 738
        null
        null
    0: 970
      0: 801
        null
        null
      0: 993
        null
        null
```

Part_2->

Check AVL Or RBT

```
Avl Tree check (Tree is -> AVL) : true
RBT tree check (Tree is -> AVL) : false

RBT tree check (Tree is -> RBT) : true
AVL tree check (Tree is -> RBT) : false
```

Part_3->

```
For BST items
 For 10000 items ->-> 135129, For 10000 items ->-> 71841, For 10000 items ->-> 41052, For 10000 items ->-> 64998, For 10000 items ->-> 55163, For 10000 items ->-> 49605, For 10000 items ->->
Avarage of 10000 items -> 63160

 For 20000 items ->-> 42762, For 20000 items ->-> 36348, For 20000 items ->-> 50887, For 20000 items ->-> 90229, For 20000 items ->-> 56874, For 20000 items ->-> 59013, For 20000 items ->->
Avarage of 20000 items -> 56403

 For 40000 items ->-> 46184, For 40000 items ->-> 44900, For 40000 items ->-> 66709, For 40000 items ->-> 77400, For 40000 items ->-> 61150, For 40000 items ->-> 67137, For 40000 items ->->
Avarage of 40000 items -> 56660

 For 80000 items ->-> 45328, For 80000 items ->-> 62005, For 80000 items ->-> 48321, For 80000 items ->-> 72269, For 80000 items ->-> 38914, For 80000 items ->-> 33783, For 80000 items ->->
Avarage of 80000 items -> 47209


 For RBT items
 For 10000 items ->-> 96215, For 10000 items ->-> 84242, For 10000 items ->-> 83387, For 10000 items ->-> 86807, For 10000 items ->-> 44473, For 10000 items ->-> 59012, For 10000 items ->->
Avarage of 10000 items -> 62860

 For 20000 items ->-> 57729, For 20000 items ->-> 50888, For 20000 items ->-> 53453, For 20000 items ->-> 60295, For 20000 items ->-> 40197, For 20000 items ->-> 55591, For 20000 items ->->
Avarage of 20000 items -> 51186

 For 40000 items ->-> 60295, For 40000 items ->-> 81676, For 40000 items ->-> 79110, For 40000 items ->-> 64571, For 40000 items ->-> 68420, For 40000 items ->-> 106478, For 40000 items ->->
Avarage of 40000 items -> 71498

 For 80000 items ->-> 96643, For 80000 items ->-> 90656, For 80000 items ->-> 141544, For 80000 items ->-> 59439, For 80000 items ->-> 97071, For 80000 items ->-> 71413, For 80000 items ->->
Avarage of 80000 items -> 80649
```

```
For Btree items
 For 10000 items ->-> 48321, For 10000 items ->-> 48749, For 10000 items ->-> 49177, For 10000 items ->-> 49177, For 10000 items ->-> 49177, For 10000 items ->-> 52170, For 10000 items ->->
Avarage of 10000 items -> 49005

 For 20000 items ->-> 52598, For 20000 items ->-> 66281, For 20000 items ->-> 58157, For 20000 items ->-> 75690, For 20000 items ->-> 51742, For 20000 items ->-> 56019, For 20000 items ->->
Avarage of 20000 items -> 58584

 For 40000 items ->-> 56874, For 40000 items ->-> 74834, For 40000 items ->-> 97498, For 40000 items ->-> 73552, For 40000 items ->-> 106051, For 40000 items ->-> 56019, For 40000 items ->->
Avarage of 40000 items -> 70130

 For 80000 items ->-> 86380, For 80000 items ->-> 106051, For 80000 items ->-> 54308, For 80000 items ->-> 81676, For 80000 items ->-> 58156, For 80000 items ->-> 54736, For 80000 items ->->
Avarage of 80000 items -> 69617


 For SkipList items
 For 10000 items ->-> 162069, For 10000 items ->-> 156510, For 10000 items ->-> 109471, For 10000 items ->-> 167201, For 10000 items ->-> 145392, For 10000 items ->-> 200983, For 10000 items
Avarage of 10000 items -> 139405

 For 20000 items ->-> 165918, For 20000 items ->-> 217660, For 20000 items ->-> 270258, For 20000 items ->-> 207826, For 20000 items ->-> 193286, For 20000 items ->-> 248449, For 20000 items
Avarage of 20000 items -> 212700

 For 40000 items ->-> 348513, For 40000 items ->-> 372461, For 40000 items ->-> 359631, For 40000 items ->-> 382296, For 40000 items ->-> 301474, For 40000 items ->-> 305751, For 40000 items
Avarage of 40000 items -> 348085

 For 80000 items ->-> 598673, For 80000 items ->-> 621765, For 80000 items ->-> 709000, For 80000 items ->-> 605088, For 80000 items ->-> 551207, For 80000 items ->-> 544792, For 80000 items
Avarage of 80000 items -> 592857
```

```
[66538, 87577, 89330, 185160, 580883]
BST <BTree <RBT <TwoThreeTree <SkipList
```

Compare