GTU Department of Computer Engineering
CSE 222/505 - Spring 2021
Homework # Report

Sena Nur Ulukaya
1901042622

1. SYSTEM REQUIREMENTS

a. Non-Functional Requirements

- Java Runtime Environment to run code.
- Some memory to test the program because it will test with different data sizes. (100-200 kb)
- Min 4 GB RAM because it will load some data and can be difficult with lower RAM.
- Java as programming language.

b. Functional Requirements

- SkipList with insert, delete and descending iterator. Elements must be unique because it is implementing NavigableSet.
    *Insert add unique elements.
    *Descending iterator iterate through elements in decreasing order.

- AVLTree with insert, iterator, headset, tailset which implementing NavigableSet.
    *HeadSet returns elements until the given element without including it.
    *TailSet returns elements from given element with including it.
    * There's a function to include or exclude the given element.
    * Iterator iterates through elements.

- Function to check if given Binary Search Tree is Red Black Tree or AVLTree.
    * Function to count black nodes of tree.
    * Function to count heigh of the tree.

- Binary Search Tree, Red Black Tree, 23Tree, B Tree implementations and MySkipList implementation with not unique elements to compare them.
- Function to add unique elements in the given size to the data structures.
- Function to calculate average running time of each data structure in each size.

2. CLASS DIAGRAMS

(I added full version on the file also, it didn't fit.)

3. PROBLEM SOLUTION APPROACH

Part1:

I used algorithms we discussed in class from the book and used available helper methods from book. (such as rebalanceLeft etc.)

- MySkipList

I implemented insert and delete method but only with unique elements because SkipList is implementing NavigableSet.

For descending iterator, I added elements in the first level of the skipList to an arrayList and iterate through the end of the arrayList to the beginning.

- MyAVLTree

I used inorder traversal of the tree to implement iterator. Again, I keep the data in an arrayList.

For headset and tailSet I used recursive helper methods.

In headSet if given element is smaller than or equal to root, I traverse only left, otherwise I traverse both left and right. I included given element if given Boolean parameter is true, otherwise excluded it.

In tailSet if given element is greater than or equal to root, I traverse only left, otherwise I traverse both left and right. I excluded given element if given Boolean parameter is false, otherwise included it.

Part2:

AVLTree nodes are red always, to indicate it.

I implemented a method to check height difference because it is not an AVLTree if difference is more than 2.

I implemented a method to count black nodes of the Red Black Tree because it must be equal in every path from the root. If it is not it is not RBT.

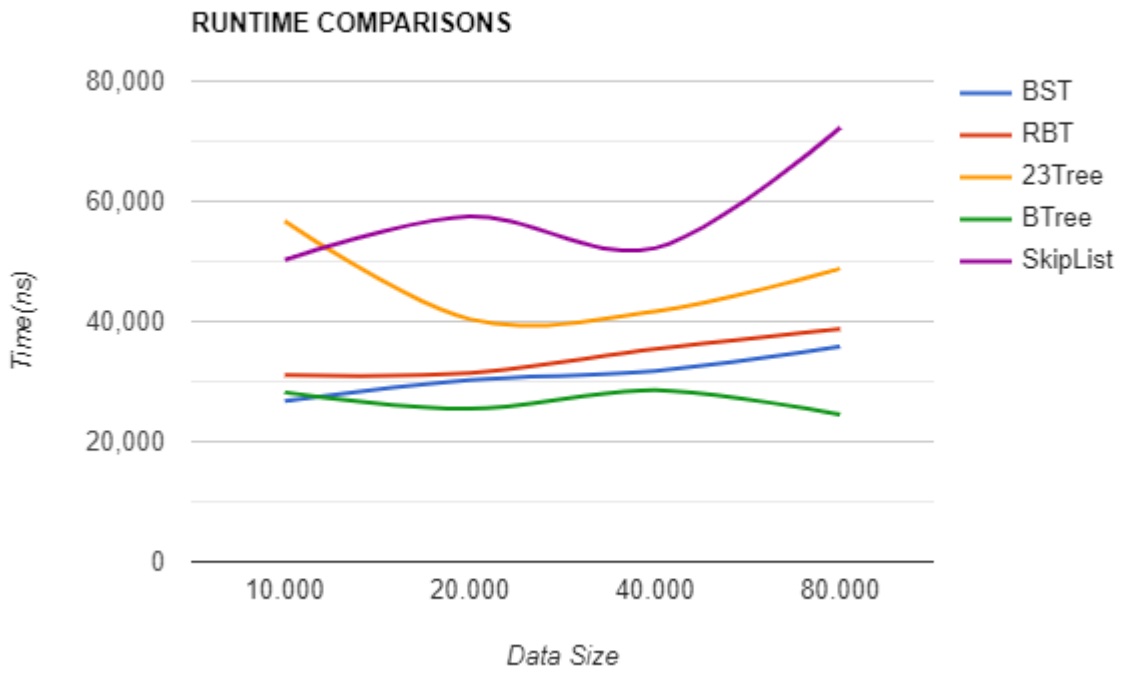
Also, if root is not black and red roots have red children than it is not an RBT.

However, a tree can be balanced like AVLTree even though it is not an instance of it. So we can't say it is AVLTree for sure, but because we will only check RedBlackTree and AVLTree it is enough to compare. Also it can be neither of them and I checked that too.

Part3:

I used the implementation from to book and MySkipList implementation without the NavigableSet because it can have element more than once.

I added elements in the given data sizes and calculated average running time for each.



This is the graph for my test case. SkipList has worst performance with bigger data sizes.

B-Tree is best with bigger data sizes. Generally all of them run times increased when data size is bigger.

4. TEST CASES

Part 1:

I used the example from the book and the class to check if my AVL tree working fine.

I added 10k elements for each. Removed non-existing elements, try to add same element more than once, added null element and displayed the success situations and sizes.

Also, I showed the iterators and iterator functions.

HeadSet and TailSet with include and exclude.

Part2:

I added elements to create right-right tree. Of course AVL and RBT balanced theirselves but BST didn't. I created BST array with them and tried with the function for the each array element.

Then I showed the results with my function and with the instanceof to test if it is working well.

Part 3:

I displayed the average running time in nanoseconds for each data size and each data structure.

5. RUNNING AND RESULTS

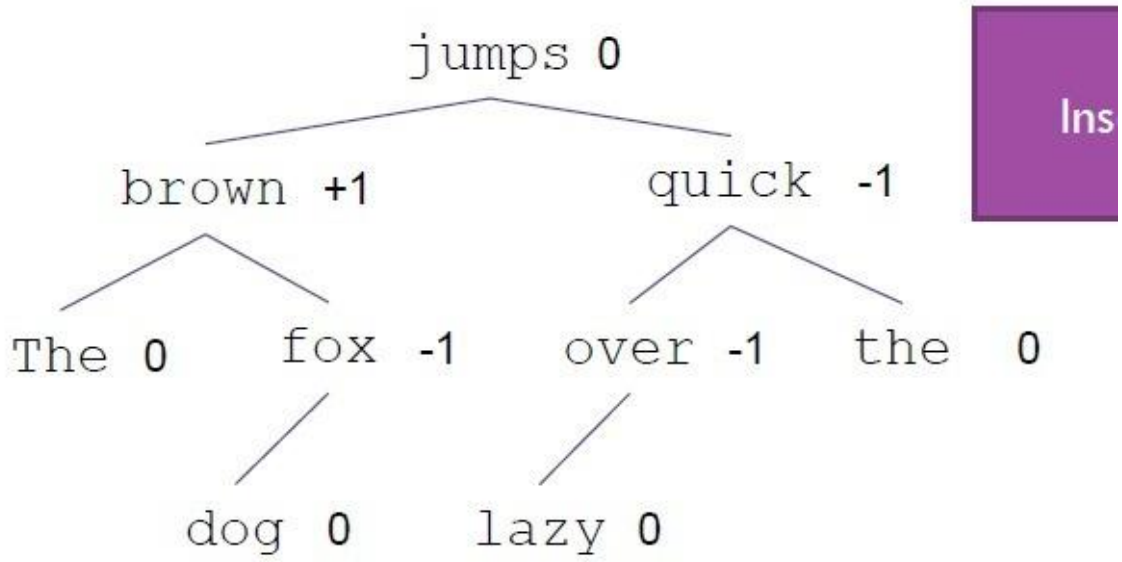
```
------ PART1: SKIPLIST ------



ADD TEST:


size: 0
Adding 10.000 unique elements to MySkipList:(There will be error message if unsuccesful)

size: 10000
Adding same number 5 times to MySkipList:

Adding is succesfull.
Adding is unsuccesfull.
Adding is unsuccesfull.
Adding is unsuccesfull.
Adding is unsuccesfull.
size: 10001

Adding null element to MySkipList:

You can't add null elements.


REMOVE TEST:


size: 10001
Removing non-existing elements from MySkipList:

Removing is unsuccesfull.
Removing is unsuccesfull.
Removing is unsuccesfull.
Removing is unsuccesfull.
Removing is unsuccesfull.
size: 10001

Removing all numbers from MySkipList:(There will be error message if unsuccesful)

size: 0
```

```
DESCENDING ITERATOR TEST:


Numbers between 0-30 added.
SkipList Elements with ToString: 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 19 -> 20 -> 21 -> 22 -> 23 -> 24 -> 25 -> 26 -> 27 -> 28 -> 29 -> 30
SkipList Elements with descendingIterator:
30
29
28
27
26
25
24
23
22
21
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0

Removing number 12 with descendingIterator:
SkipList Elements with ToString: 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 19 -> 20 -> 21 -> 22 -> 23 -> 24 -> 25 -> 26 -> 27 -> 28 -> 29 -> 30
```

```
------ PART1: AVLTREE



ADDING DEMONSTRATION


0: The
  null
  null

1: The
  null
  0: quick
    null
    null

0: brown
  0: The
    null
    null
  0: quick
    null
    null

1: brown
  0: The
    null
    null
  -1: quick
    0: fox
      null
      null
    null

1: brown
  0: The
    null
    null
  0: jumps
    0: fox
      null
      null
    0: quick
      null
      null
```

```
0: jumps
  0: brown
    0: The
      null
      null
    0: fox
      null
      null
  -1: quick
    0: over
      null
      null
    null

0: jumps
  0: brown
    0: The
      null
      null
    0: fox
      null
      null
  0: quick
    0: over
      null
      null
    0: the
      null
      null
```

```
1: jumps
  0: brown
    0: The
      null
      null
    0: fox
      null
      null
  -1: quick
    -1: over
      0: lazy
        null
        null
      null
    0: the
      null
      null

0: jumps
  1: brown
    0: The
      null
      null
    -1: fox
      0: dog
        null
        null
      null
  -1: quick
    -1: over
      0: lazy
        null
        null
      null
    0: the
      null
      null
```



Book result, it is same.

```
ADD TEST:


Adding 10.000 unique elements to MyAVLTree:(There will be error message if unsuccesful)

Adding same number 5 times to MyAVLTree:

Adding is succesfull.
Adding is unsuccesfull.
Adding is unsuccesfull.
Adding is unsuccesfull.
Adding is unsuccesfull.

Adding null element to MyAVLTree:

You can't add null elements.


HEAD SET AND TAIL SET:


Added numbers between 0-30.
Head Set with the given element 12 excluded:
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

Head Set with the given element 12 included:
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

Tail Set with the given element 17 excluded:
[18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]

Tail Set with the given element 17 included:
[17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]
```

```
------ PART2: AVLTREE OR REDBLACK TREE ------



Adding 5-10-15-17-25-42 in order to the AVLTree, BinarySearchTree and RedBlackTree:


Checking whether AVLtree or RedBlackTree with my function:

It is a AVL Tree.
It is neither of them
It is a Red Black Tree.

Checking with instance of function to compare:

It is a AVL Tree.
It is neither of them
It is a Red Black Tree.
```

```
RUNNING TIME COMPARISONS (nanosecond)

Adding 100 elements with data size of 10000:

Binary Search Tree:26767
Red-Black Tree:31059
2-3 Tree:56687
B Tree:28208
Skip List:50249

Adding 100 elements with data size of 20000:

Binary Search Tree:30249
Red-Black Tree:31428
2-3 Tree:40368
B Tree:25459
Skip List:57487

Adding 100 elements with data size of 40000:

Binary Search Tree:31766
Red-Black Tree:35419
2-3 Tree:41659
B Tree:28546
Skip List:52208

Adding 100 elements with data size of 80000:

Binary Search Tree:35837
Red-Black Tree:38748
2-3 Tree:48790
B Tree:24490
Skip List:72268
```