Assignment 5
INFO 6205 section 1
Qixiang Zhou

Idea

This assignment is to proof BST is not a balanced data structure. Which means when we randomly add and delete nodes, the tree will end up to imbalanced. Therefore, the height of tree (both max height and average height) will increase.

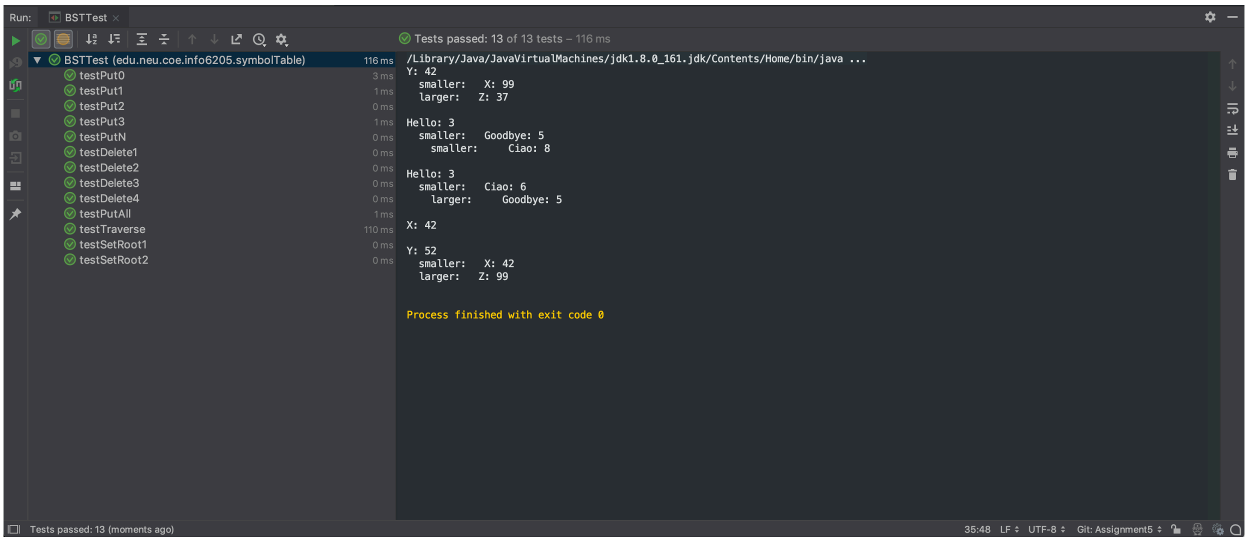
Theoretically, the height, or say depth, of the BST will be log by 2 of N (which is the number of nodes), however, when we do the process above many times, the BST will imbalance and goes to square root of N.

In order to proof this theory, we need to first create a BST with some nodes, in other words, create a BST with size N. Than we randomly choose a node to delete, and randomly add a node into BST. We should try to maintain the size of BST. For let's say 10000 or 50000 times deletion and insertion, we can see how height of BST goes. Than we can proof our thought.

Experiment

Since we want to keep the size of BST stable, therefore, I choose to create a full BST. Since the node of BST doesn't contain same key node. Thus, we can random generate key from 0 – 1999, and initialize a BST with size 2000. After this, we random choose a key from 0 -1999, delete the node, and add it again. Below is the step:

1. Initialize a BST with size 2000, and key range with 0 -1999.
2. Random chose a key from 0 -1999
3. Delete that key node.
4. Insert that key node.
5. Show the maxHeight, and aveHeight of BST, and write into .csv file
6. Loop 1 -5 steps 50000 times.
7. Draw pilot to show how height goes.



Tests passed: 13 of 13 tests – 116 ms

```
BSTTest (edu.neu.coe.info6205.symbolTable)   116 ms
    testPut0        3 ms
    testPut1        1 ms
    testPut2        0 ms
    testPut3        1 ms
    testPutN        0 ms
    testDelete1     0 ms
    testDelete2     0 ms
    testDelete3     0 ms
    testDelete4     0 ms
    testPutAll      1 ms
    testTraverse  110 ms
    testSetRoot1    0 ms
    testSetRoot2    0 ms
```

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_161.jdk/Contents/Home/bin/java ...
Y: 42
  smaller:   X: 99
  larger:   Z: 37

Hello: 3
  smaller:   Goodbye: 5
    smaller:     Ciao: 8

Hello: 3
  smaller:   Ciao: 6
    larger:     Goodbye: 5

X: 42

Y: 52
  smaller:   X: 42
  larger:   Z: 99


Process finished with exit code 0
```

Tests passed: 13 (moments ago)

The unit test of BSTSimple has all passed, however, there is a bug in BSTSimple.java's put() method, when we put same key again and again, the size of the BST will increase, which should not in BST's definition. So, I write my own BST file which is BSTmine.java. I use Hibbard deletion.

In order to count average height and max height, I write two new methods:

1. This one count max height buy using recursive method.

```
/**
* this method is used to calculate the max height of BST
 * @return int, recursive find the max of left subtree and right subtree. Finally
find the larger one is the max height of BST*/
```

```java
public int height() {
    return height(root);
}
private int height(Node x) {
    if (x == null) return -1;
    return 1 + Math.max(height(x.left), height(x.right));
}
```

2. This one count the average height of BST by using DFS.

```java
/**
 * Calculate the aveheight of the BST
 * Use BFS to implement this function
 * I use two queues, one store the node will be visited next, and one store the
height this node have.
 * @return float, after BFS it will return totalHeight / # of leafs
 * */
private float aveheight(Node node) {
    LinkedList<Node> queue = new LinkedList<>();
    LinkedList<Integer> len = new LinkedList<>();
    int totalLen = 0;
    int leafnum = 0;
    queue.add(node);
    len.add(0);

    while (queue.size() != 0) {
        Node n = queue.poll();
        int l = len.poll();
        if (!isLeaf(n)) {
            if (n.left != null) {
                queue.add(n.left);
                len.add(l+1);
            }
            if (n.right != null) {
                queue.add(n.right);
                len.add(l+1);
            }
        } else {
            totalLen += l;
            leafnum += 1;
        }
    }
    return totalLen / leafnum;
}

boolean isLeaf(Node node) {
    return node.left == null && node.right == null;
}
```

In the BSTmain.java, which will do the experiment. I generate a full BST which size is 2000 and key range from 0 – 1999.

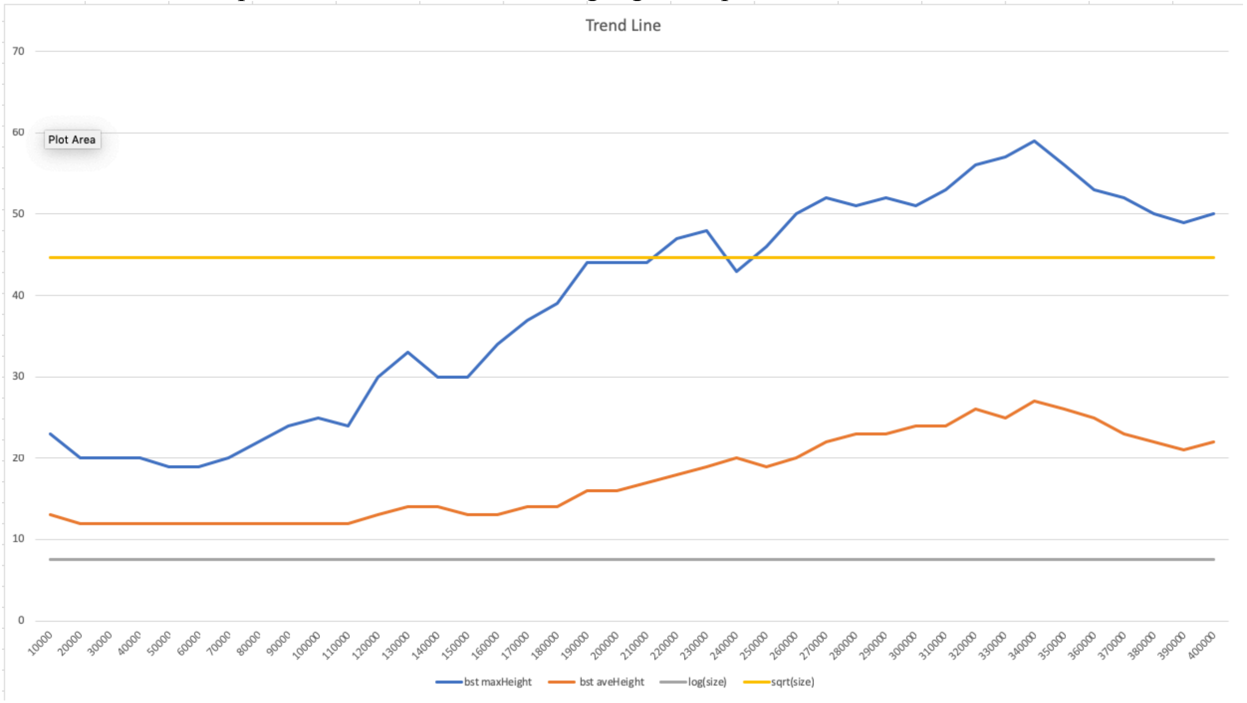
```java
/*
    Here I used my own BST implementation, since the original one have some problem
when I add same key again and again.
    It will ultimately increase the size of BST, which should not be.
*/
BSTmine<Integer, Integer> bst = new BSTmine<>();
/*
    create a bst with 2000 nodes, the keys is from 0 - 1999.
    In other words, we create a full bst.
*/
while (bst.size() < 2000) {
    bst.put(getRandom(2000), 1);
}
System.out.println("Origin height: " + bst.aveheight());
System.out.println(bst.size());
```

After initialization of BST, I choose a node delete first then insert it.

```
for (int N = 10000; N <= 500000; N += 10000) {
     for (int j = 0; j < N; j++) {
     /*we simply get a random key, first delete it, than insert it again.
     Since we generate keys from 0 to 1999. Therefore, the size of the binary tree
will be constant.*/
         Integer k = getRandom(2000);
         bst.delete(k);
         bst.put(k, 1);
     /*Normally, if we delete the node, and add a node with same key again, the
height of BST will not change so much.
      What we expect to see is the balance of BST will be broken via this
modification.*/
            }
System.out.println(N+","+bst.size()+","+bst.height()+","+bst.aveheight()+","+Math.log
(bst.size())+","+Math.sqrt(bst.size()) + ","+Math.log(N)+","+Math.sqrt(N));
        }
```

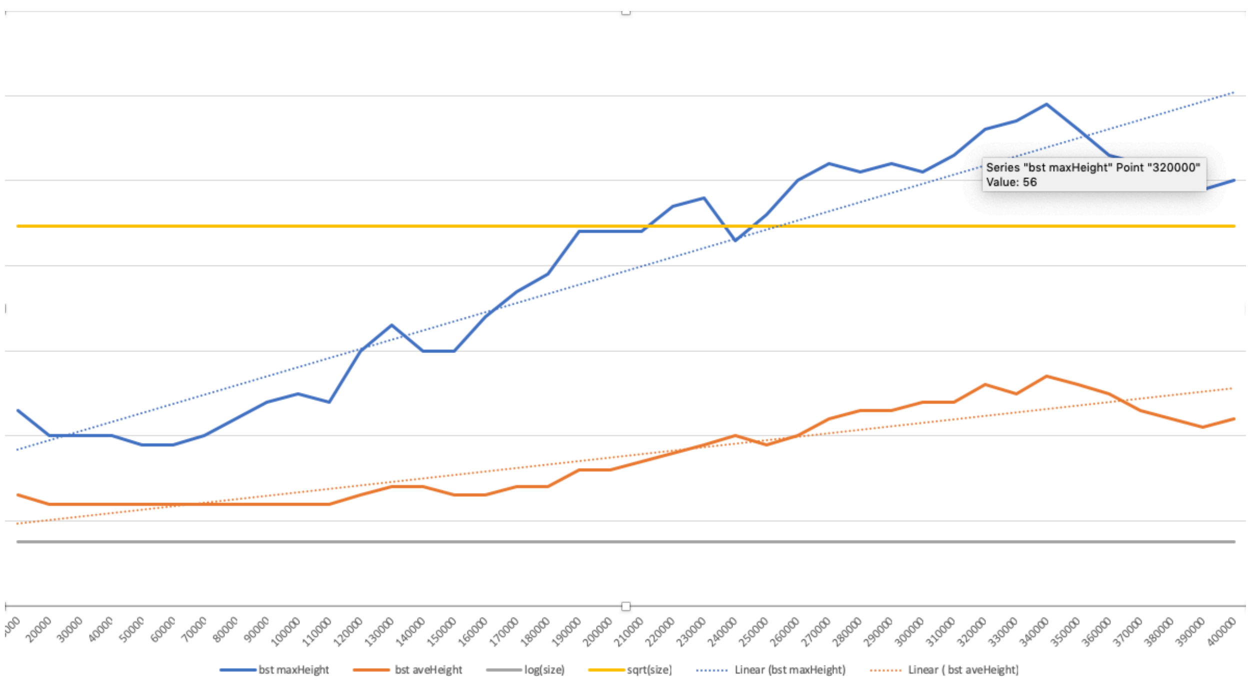
Results

After the experiment, we can see how height goes in pilot:



The grey line represents the log of BST size. And the light orange line represents the square root of BST size. In the beginning, the average height (dark orange line) and max height (blue line) close to the log size until 100000-time deletion and insertion. After, the balance of the BST has been broken, the max height get huge increase, and the average height increase too. And both will reach the light orange line (square root of size).

When I add a trend line by linear regression, the trend will be clearer.



We can see both max height and average height of BST goes up. The worst case will be square root of BST size.

Conclusion

In the experiment, I create a full BST, and delete, insert together. Therefore, the size of BST is fixed. Also, since the key range is same as BST size, we will not add new key into BST. Ideally, the BST should maintain what it is, by having constant size, height, and structure.

However, after the experience, we found the max height and average height both increase when we do the operation many times. Thus, I can have a conclusion that, the BST is not balanced, and when we modify BST so many times, it will end up to very imbalance between left subtree and right subtree.

Theoretically, the height of BST should be logN, which means the deletion will be logN. But since the BST will be imbalance, the performance of deletion will be bad as square root of N (N^1/2).