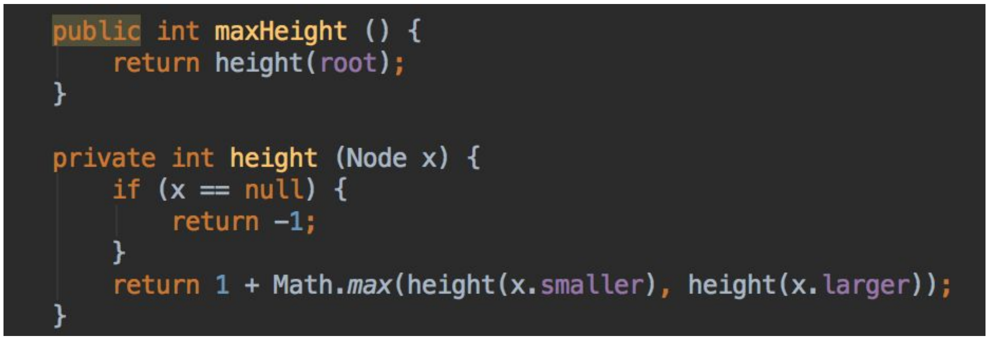
INFO6205 Assignment 5

Program Structures & Algorithms

2018 Fall Jingxuan Li

- Instructions for code testing

1. To test the max depth of the BST, I implemented a method to calculate the max depth of the BST in BSTSimple.java as following:

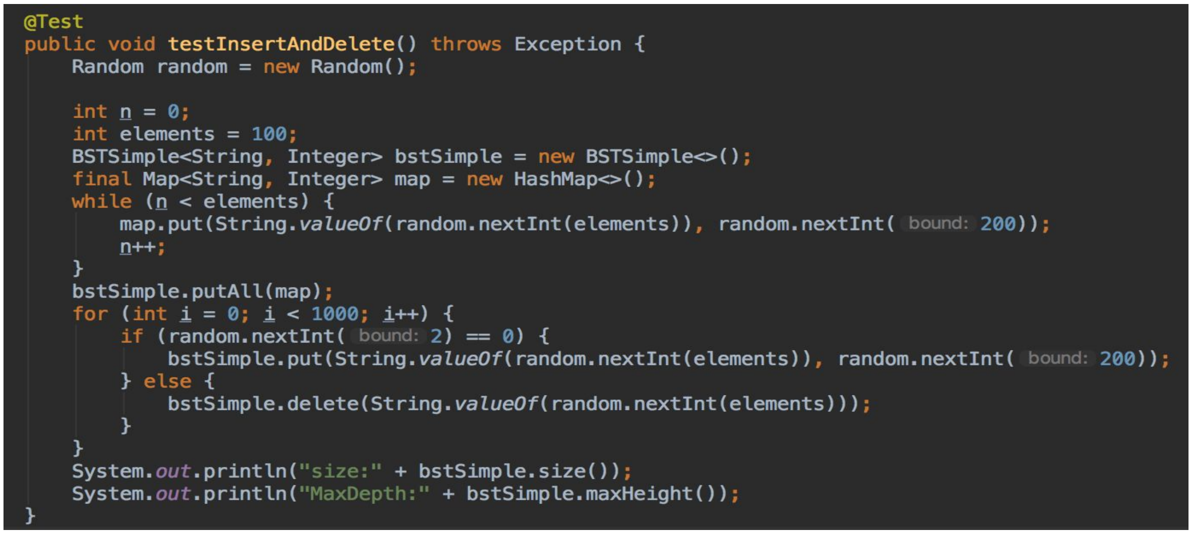


```
public int maxHeight () {
    return height(root);
}

private int height (Node x) {
    if (x == null) {
        return -1;
    }
    return 1 + Math.max(height(x.smaller), height(x.larger));
}
```

2. To generate elements and add them to the Binary Search Tree, I set up a map and set the number of elements. And then I generated keys with random number in the range of 0 to elements-1 and values with random number in the range of 0 to 199. Then I put all of the key value pairs in the map to the Binary Search Tree using putAll() method.

3. To control the number of operation (including insertion and deletion), in my for loop, I can change the number of operator manually.



```
@Test
public void testInsertAndDelete() throws Exception {
    Random random = new Random();

    int n = 0;
    int elements = 100;
    BSTSimple<String, Integer> bstSimple = new BSTSimple<>();
    final Map<String, Integer> map = new HashMap<>();
    while (n < elements) {
        map.put(String.valueOf(random.nextInt(elements)), random.nextInt( bound: 200));
        n++;
    }
    bstSimple.putAll(map);
    for (int i = 0; i < 1000; i++) {
        if (random.nextInt( bound: 2) == 0) {
            bstSimple.put(String.valueOf(random.nextInt(elements)), random.nextInt( bound: 200));
        } else {
            bstSimple.delete(String.valueOf(random.nextInt(elements)));
        }
    }
    System.out.println("size:" + bstSimple.size());
    System.out.println("MaxDepth:" + bstSimple.maxHeight());
}
```

- Conclusion:

  Assuming that M is the number of operations, and N is the number of elements in the BST, we can conclude that when M is large enough, the depth of Binary Search Tree will end up being ***O***(*N^1/2*) instead of ***O***(*lg N*).

- *Proof:*

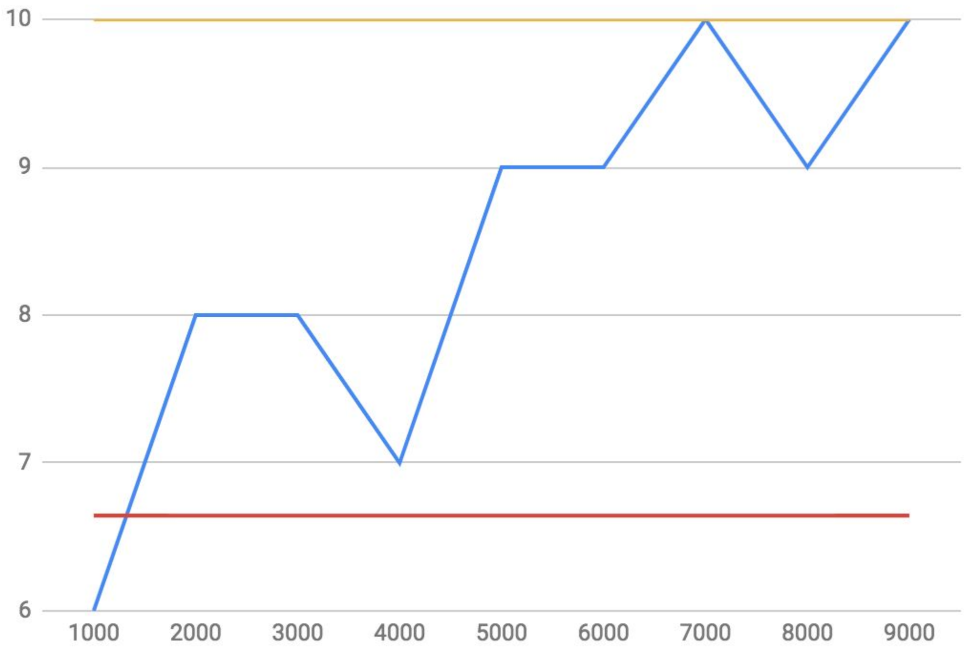
  Here is the data that I collected. I just set up 100 elements to the Binary Search Tree initially.

  logN = *log100* = 6.64385619;

  √n = 10.

operation	*max depth*
1000	*6*
2000	*8*

3000	*8*
4000	*7*
5000	*9*
6000	*9*
7000	*10*
8000	*9*
9000	*10*



The red line stands for lgN; the yellow line stands for √n; The blue line stands for the actual max depth of the BST. We can see that if we increase M, the max depth of BST will end up being ***O****(N^1/2)* instead of ***O****(lg N)*.