

Objective(s):

- To understand the BST form characteristic which depends on the order of the input.
- To understand the process of BST deletion

Task 1:

As mentioned during the class, BST shape depends on the order of the input.

Implement /* your code 9 */ so that your BST can return its height.

```
class BST {
...
    public int height() {
        return root == null ? 0 : height(root);
    }
    public int height(TreeNode node) {
        if (node == null)
            return 0;
        return 999 /* your code 9 */;
    }
}
```

```
public static void demo1() {
    int [] data = {2,1,3,4,5,6,7,8,9};
    bst = new BST();
    for (int j = 0; j < data.length; j++)
        bst.insert(data[j]);

    bst.printInOrder();
    println("Tree height = " + bst.height());

    int[] dat = { 15, 20, 10, 18, 16, 12, 8, 25, 19, 30};
    bst = new BST();
    for (int j = 0; j < dat.length; j++)
        bst.insert(dat[j]);

    bst.printInOrder();
    println("Tree height = " + bst.height());
}
```

Instruction: Capture your int height(TreeNode node) and demo1()'s output

```

public int height() {
    return root == null ? 0 : height(root);
}
public int height(TreeNode node) {
    if (node == null)
        return 0;
    return 1 + Math.max(height(node.left), height(node.right));
}

```

```

1 2 3 4 5 6 7 8 9 Tree height = 8
8 10 12 15 16 18 19 20 25 30 Tree height = 4
Done!

```

```

//          15          1
//    10          20    2
//  8 12      18  25    3
//        16 19  30    4

```

```

//      2          1
//    1 3          2
//      4          3
//      5          4
//      6          5
//      7          6
//      8          7
//      9          8

```

Task 2:

To delete a node on a BST, it must know the node with the maximum value to replace the deleted node content.

```
public static void demo2() {
    println("node with max value " +
           bst.findMaxFrom(bst.getRoot()));
}
```

```
class BST {
...
    public TreeNode findMaxFrom(TreeNode subtreeHead) {
        /* your code 10 */
        return current;
    }
}
```

Implement /* your code 10 */

Instruction: Capture your int height(TreeNode node) and demo1()'s output

```
public TreeNode findMaxFrom(TreeNode subtreeHead) {
    TreeNode current = subtreeHead;
    while (current.right != null) {
        current = current.right;
    }

    return current;
}
```

```
node with max value null<-30->null
Done!
```

```
//      15
//    10      20
//  8 12    18 25
//      16 19    30
```

30 is the rightmost
value in the tree, so
it's the max

Task 3:

Implement /* your
code 11 */

```
public static void demo3() {
    bst.delete(12, bst.getRoot());
    println(bst.search(20)); // 18<-20->25
    println(bst.search(25)); // null<-25->30
    println(bst.search(16)); // null<-16->null
    println(bst.search(10)); // 8<-10->null
    println(bst.search(12)); // not found
}
```

```
class BST {
...
    public void delete(int d, TreeNode current) {
        if (current == null) return; //not found
        if (d < current.data)
            delete(d, current.left);
        else if (d > current.data)
            delete(d, current.right);
        else { //found ... time to delete
            if (current.left == null || current.right == null) { // 0 or 1 child
                TreeNode q = (current.left == null) ? current.right : current.left;
                if (current.parent.left == current)
                    current.parent.left = q; //this node is left child
                else
                    current.parent.right = q;
                if (q != null) q.parent = current.parent;
            } else { // two children
                TreeNode q = findMaxFrom(current.left);
                /* your code 11 */
            } // two children
        } //found
    }
}
```

Instruction: Capture your int height(TreeNode node) and demo1()'s output

Submission: this pdf

Due date: TBA

```

public void delete(int d, TreeNode current) {
    if (current == null) return; //not found
    if (d < current.data) delete(d, current.left);
    else if (d > current.data) delete(d, current.right);
    else { //found ... time to delete
        if (current.left == null || current.right == null) { // 0 or 1 child
            TreeNode q = (current.left == null) ? current.right : current.left;
            if (current.parent.left == current)
                current.parent.left = q; //this node is left child
            else current.parent.right = q;
            if (q != null) q.parent = current.parent;
            System.out.println("Deleted " + d);
        }
        else { // two children
            TreeNode q = findMaxFrom(current.left);
            // * your code 11 * //
            current.data = q.data; //copy the data
            delete(q.data, current.left); //delete the node that had the max value
        } // two children } //found
    }
}
}

```

```

Deleted 12
18<-20->25
null<-25->30
null<-16->null
8<-10->null
null
Done!

```