자료구조 실습 보고서

숙제명: [제13주]이진탐색트리

제출일: 2018.05.29

학번/이름: 201404377 / 진승언

<프로그램 설명서>

이번 과제는 이진탐색트리(BinarySearchTree)와 AVL Tree를 구현하는 것이었다. 이진 탐색 트리란 각각의 노드가 다음과 같은 BST 특성을 만족하는 키 주소 쌍을 가지고 있는 이진트리이다. 트리에 있는 각각의 키에 대해, 왼쪽 서브 트리에 있는 모든 키는 이것보다 작고, 오른쪽 서브트리에 있는 모든 키는 이것보다 크다는 성질을 갖고 있다.

AVL Tree란 어떤 노드에서도 두 서브트리가 거의 같은 높이를 갖도록 강제하여 균형을 유지하는 이진 탐색 트리이다. 만약 불균형이 생긴 경우, 서브트리를 회전하여 균형을 맞추어준다.

1. 이진 탐색 트리

insertKey 메소드와 isBST() 메소드를 구현해야했다.

```
public void insertKey(Object data) {
    if (data.hashCode() > root.hashCode()) {
        if (right != null) {
            right.insertKey(data);
        } else if (right == null) {
            BinaryTree tmp = new BinaryTree(data);
            right = tmp;
        }
    } else if (data.hashCode() < root.hashCode()) {
        if (left != null) {
            left.insertKey(data);
        } else if (left == null) {
            BinaryTree tmp = new BinaryTree(data);
            left = tmp;
        }
    }
}
```

<insertKey>

=> Comparable을 이용해 비교를 할 수 도 있었지만 hashCode로도 되기 때문에 hashCode로 비교를 해주었다. 삽입할려는 data가 root값 보다 크고 right에 값이 이미 있는 경우는 right로 재귀를 root보다 값이 작고 left에 이미 값이 이미 있는 left로 재귀를 돌려주었다. 만약 비교를 해서 left나 right에 값이 null이라면 그 곳에 해당 data를 삽입해주었다.

```
public Boolean isBST() {
    Comparable key = (Comparable) root;
    if (left != null && key.compareTo(this.left.getRoot()) < 0)
        return false;
    if (right != null && key.compareTo(this.right.getRoot()) > 0)
        return false;
    if (left != null) {
        left.isBST();
    }
    if (right != null) {
        right.isBST();
    }
    return true;

}
```

<isBST()>

=> 먼저 가장 꼭대기에 있는 값 root를 비교를 가능하게 하도록 하기위해 Comparable로 형변환을 해준다. isBST는 해당 트리가 이진탐색트리인지 확인하는 것이니깐 왼쪽은 root보다 작은 값 오른쪽은 root보다 큰 값이 나와야 한다. 만약 그렇지 않다면 false를 반환시켜주었다. 그리고 left와 right가 존재한다면 재귀를 돌려주었다. 이렇게 다 탐색을 하고 이상이 없으면 true를 반환시켜주게 코딩을 하였다.

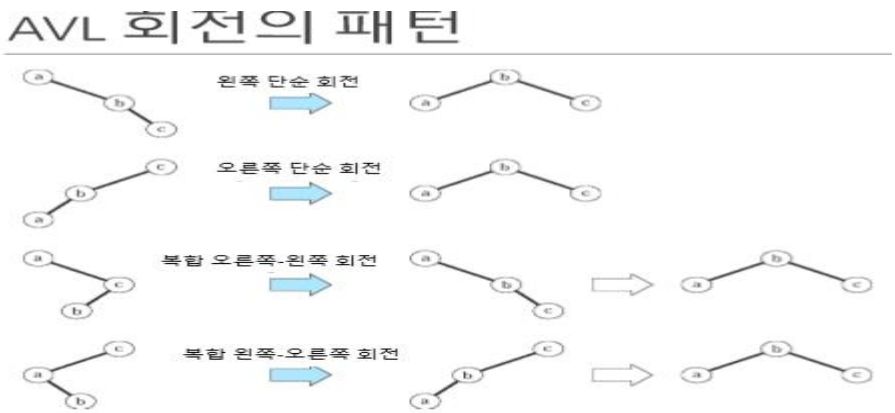
<실행 결과>

```
<terminated> sub_main (1) [Java Application] C:₩P
treeA: [[[A],B],C,[D,[[E,[F]],G]]]
treeA.isBSt(): true
```

결과가 잘 출력됨을 볼 수 있었고 isBST를 디버깅 해본 결과 모든 값들을 탐색함을 볼 수 있었다.

2. AVL Tree

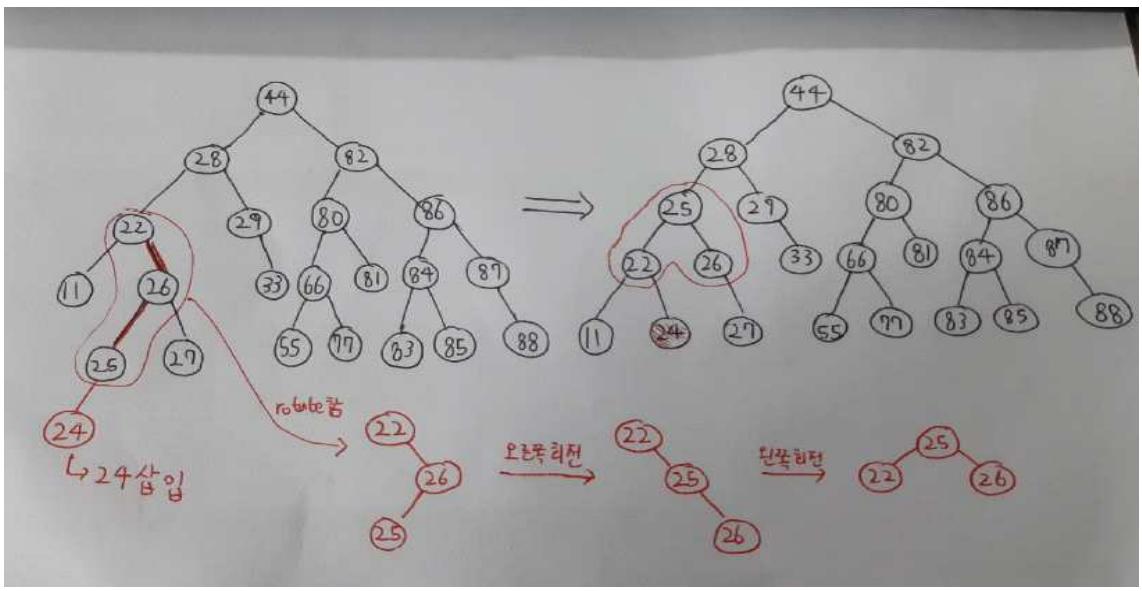
어떤 데이터가 삽입되든지 두 서브트리가 거의 같은 높이를 갖도록 강제하여 균형을 유지하게 해야한다. 그래서 다음 그림과 같은 AVL 회전패턴을 이용해서 균형을 맞춰준다.



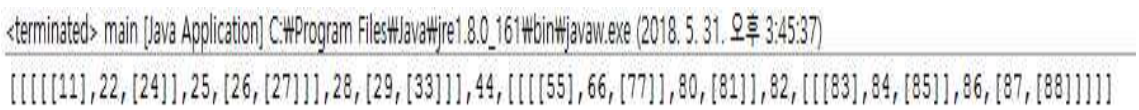
그리고 이를 AVL회전 패턴을 구현한 코드가 다음과 같다.

```
private void rebalance() {
    if(right.height > left.height+1) {
        if(right.left.height > right.right.height) right.rotateRight();
        rotateLeft();
    }
    else if(left.height > right.height+1) {
        if(left.right.height > left.left.height) left.rotateLeft();
        rotateRight();
    }
}
```

"24"가add 될 때 rotate 과정을 그림으로 표현하면 다음과 같다.



<실행 결과>


<terminated> main [Java Application] C:₩Program Files₩Java₩jre1.8.0_161₩bin₩javaw.exe (2018. 5. 31. 오후 3:45:37)
[[[[[11],22,[24]],25,[26,[27]]],28,[29,[33]]],44,[[[[55],66,[77]],80,[81]],82,[[[83],84,[85]],86,[87,[88]]]]]