AVL Search Trees

- Inserting in an AVL tree
- Insertion implementation
- Deleting from an AVL tree
Insertion

• Insert using a BST insertion algorithm.
• Rebalance the tree if an imbalance occurs.
• An imbalance occurs if a node's balance factor changes from -1 to -2 or from +1 to +2.
• Rebalancing is done at the deepest or lowest unbalanced ancestor of the inserted node.

• There are three insertion cases:
  1. Insertion that does not cause an imbalance.
  2. Same side (left-left or right-right) insertion that causes an imbalance.
     • Requires a single rotation to rebalance.
  3. Opposite side (left-right or right-left) insertion that causes an imbalance.
     • Requires a double rotation to rebalance.
Insertion: case 1

- Example: An insertion that does not cause an imbalance.
Insertion: case 2

- Case 2a: The lowest node (with a balance factor of -2) had a taller left-subtree and the insertion was on the left-subtree of its left child.
- Requires single right rotation to rebalance.
Insertion: case 2 (contd)

- Case 2b: The lowest node (with a balance factor of +2) had a taller right-subtree and the insertion was on the right-subtree of its right child.
- Requires single left rotation to rebalance.
Insertion: case 3

- Case 3a: The lowest node (with a balance factor of -2) had a taller left-subtree and the insertion was on the right-subtree of its left child.
- Requires a double left-right rotation to rebalance.

Insert 7

1. Left rotation, with node 5 as the pivot
2. Right rotation, with node 10 as the pivot

left rotation, with node 5 as the pivot
green arrow

right rotation, with node 10 as the pivot
red arrow
Insertion: case 3 (contd)

- Case 3b: The lowest node (with a balance factor of +2) had a taller right-subtree and the insertion was on the left-subtree of its right child.
- Requires a double right-left rotation to rebalance.
AVL Rotation Summary

- Single right rotation
- Double left-right rotation
- Single left rotation
- Double right-left rotation
Insertion Implementation

- The insert method of the AVLTree class is:

```java
public void insert(Comparable comparable){
    super.insert(comparable);
    balance();
}
```

Recall that the insert method of the BinarySearchTree class is:

```java
public void insert(Comparable comparable){
    if(isEmpty()) attachKey(comparable);
    else {
        Comparable key = (Comparable) getKey();
        if(comparable.compareTo(key)==0)
            throw new IllegalArgumentException("duplicate key");
        else if (comparable.compareTo(key)<0)
            getLeftBST().insert(comparable);
        else
            getRightBST().insert(comparable);
    }
}
```
Insertion Implementation (contd)

• The AVLTree class overrides the attachKey method of the BinarySearchTree class:

```java
public void attachKey(Object obj)
{
    if(!isEmpty())
        throw new InvalidOperationException();
    else
    {
        key = obj;
        left = new AVLTree();
        right = new AVLTree();
        height = 0;
    }
}
```
protected void balance(){
    adjustHeight();
    int balanceFactor = getBalanceFactor();
    if(balanceFactor == -2){
        if(getLeftAVL().getBalanceFactor() < 0)
            rotateRight();
        else
            rotateLeftRight();
    }
    else if(balanceFactor == 2){
        if(getRightAVL().getBalanceFactor() > 0)
            rotateLeft();
        else
            rotateRightLeft();
    }
}
Deletion

- Delete by a BST deletion by copying algorithm.
- Rebalance the tree if an imbalance occurs.
- There are three deletion cases:
  1. Deletion that does not cause an imbalance.
  2. Deletion that requires a single rotation to rebalance.
  3. Deletion that requires two or more rotations to rebalance.
- Deletion case 1 example:
Deletion: case 2 examples

Delete 40

right rotation, with node 35 as the pivot
Deletion: case 2 examples (contd)
Deletion: case 3 examples

Delete 40

right rotation, with node 35 as the pivot

right rotation, with node 30 as the pivot