AVL Tree

2014.1.23

Iris Jiaonghong Shi

AVL tree operations

- **AVL `find`**:
  - Same as BST **`find`**

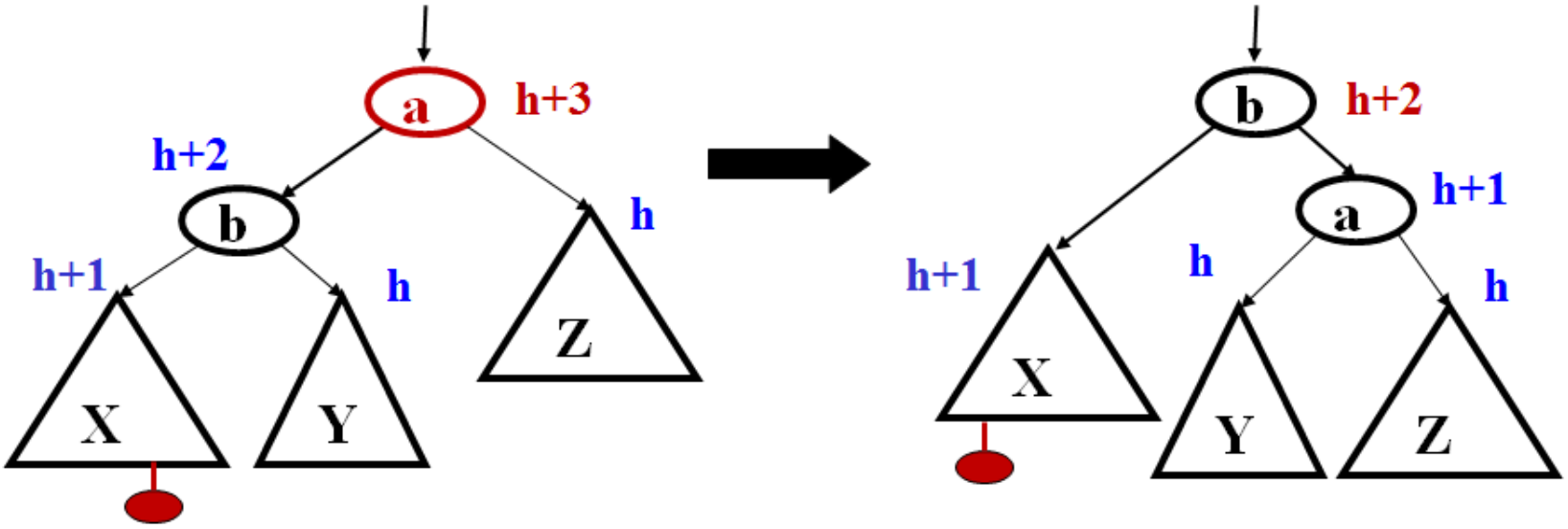
- **AVL `insert`**:
  - First BST **`insert`**, *then* check balance and potentially "fix" the AVL tree
  - Four different imbalance cases

- **AVL `delete`**:
  - The "easy way" is lazy deletion
  - Otherwise, do the deletion as binary search tree and check balance.

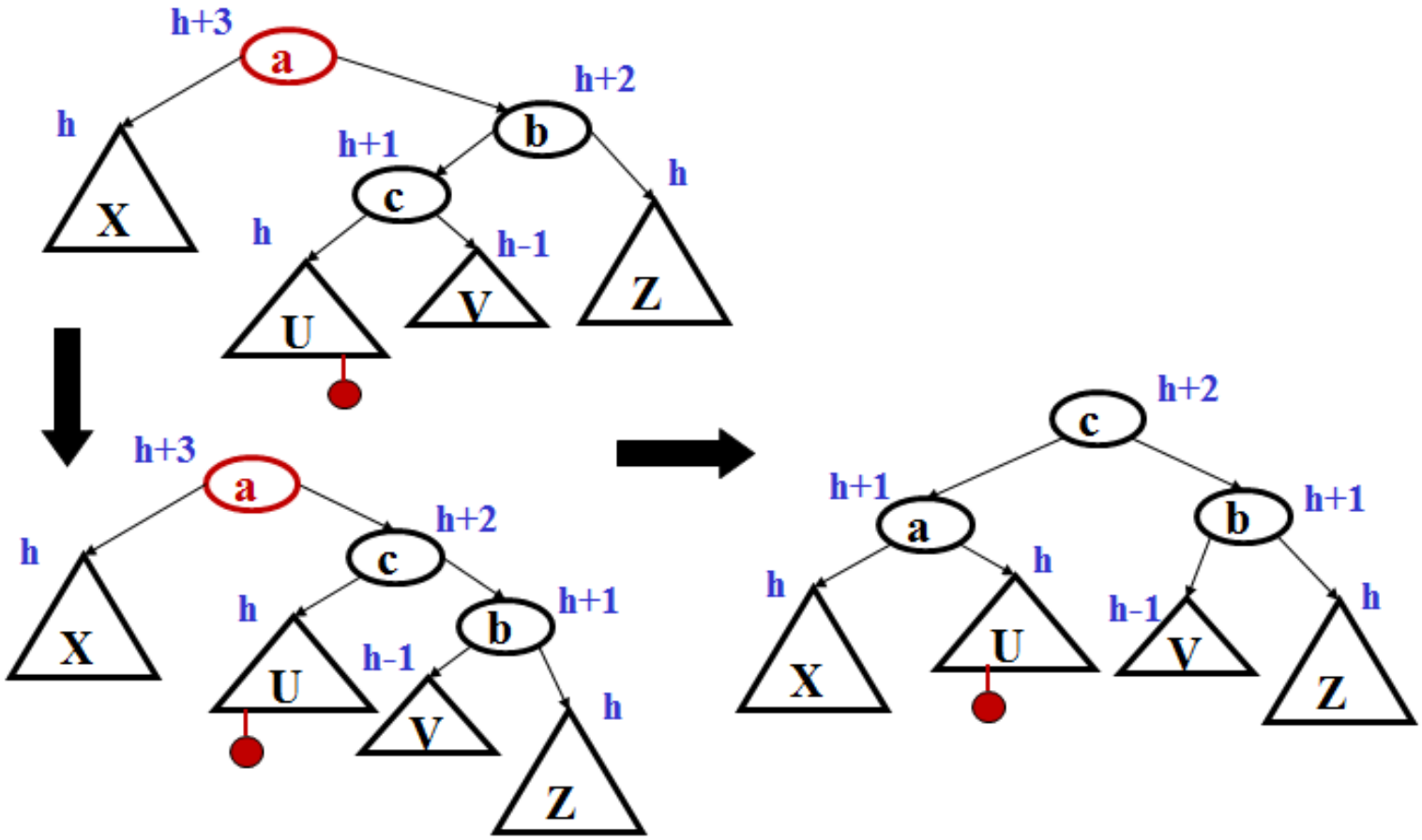
Insert, summarized

- Insert as in a BST
- Check back up path for imbalance, which will be 1 of 4 cases:
  - Node's left-left grandchild is too tall
  - Node's left-right grandchild is too tall
  - Node's right-left grandchild is too tall
  - Node's right-right grandchild is too tall
- Only one case occurs because tree was balanced before insert
- After the appropriate single or double rotation, the smallest-unbalanced subtree has the same height as before the insertion
  - So all ancestors are now balanced

Node's left-left grandchild is too tall



Node's right-left grandchild is too tall



Example: Insert

- Warm up: 2, 5, 6, 8, 9, 7
- Example in the book:
  - 3,2,1,4,5,6,7
  - 16,15,14,13,12,11,10,8,9
- An avl applet to play with:
  - http://www.site.uottawa.ca/~stan/csi2514/applets/avl/BT.html

Insert

```
/**
 * Internal method to insert into a subtree.
 * @param x the item to insert.
 * @param t the node that roots the subtree.
 * @return the new root of the subtree.
 */
private AvlNode<AnyType> insert( AnyType x, AvlNode<AnyType> t )
{
    if( t == null )
        return new AvlNode<>( x, null, null );

    int compareResult = x.compareTo( t.element );

    if( compareResult < 0 )
        t.left = insert( x, t.left );
    else if( compareResult > 0 )
        t.right = insert( x, t.right );
    else
        ;  // Duplicate; do nothing
    return balance( t );
}
```

Balance

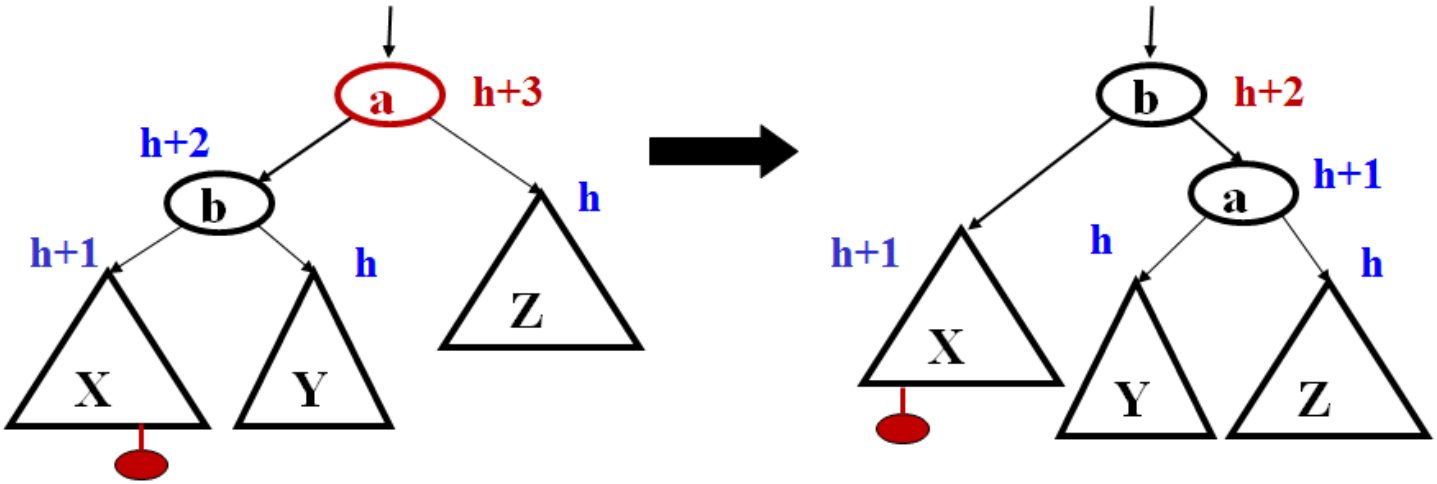
```
// Assume t is either balanced or within one of being balanced
private AvlNode<AnyType> balance( AvlNode<AnyType> t )
{
    if( t == null )
        return t;

    if( height( t.left ) - height( t.right ) > ALLOWED_IMBALANCE )
        if( height( t.left.left ) >= height( t.left.right ) )
            t = rotateWithLeftChild( t );
        else
            t = doubleWithLeftChild( t );
    else
    if( height( t.right ) - height( t.left ) > ALLOWED_IMBALANCE )
        if( height( t.right.right ) >= height( t.right.left ) )
            t = rotateWithRightChild( t );
        else
            t = doubleWithRightChild( t );

    t.height = Math.max( height( t.left ), height( t.right ) ) + 1;
    return t;
}
```

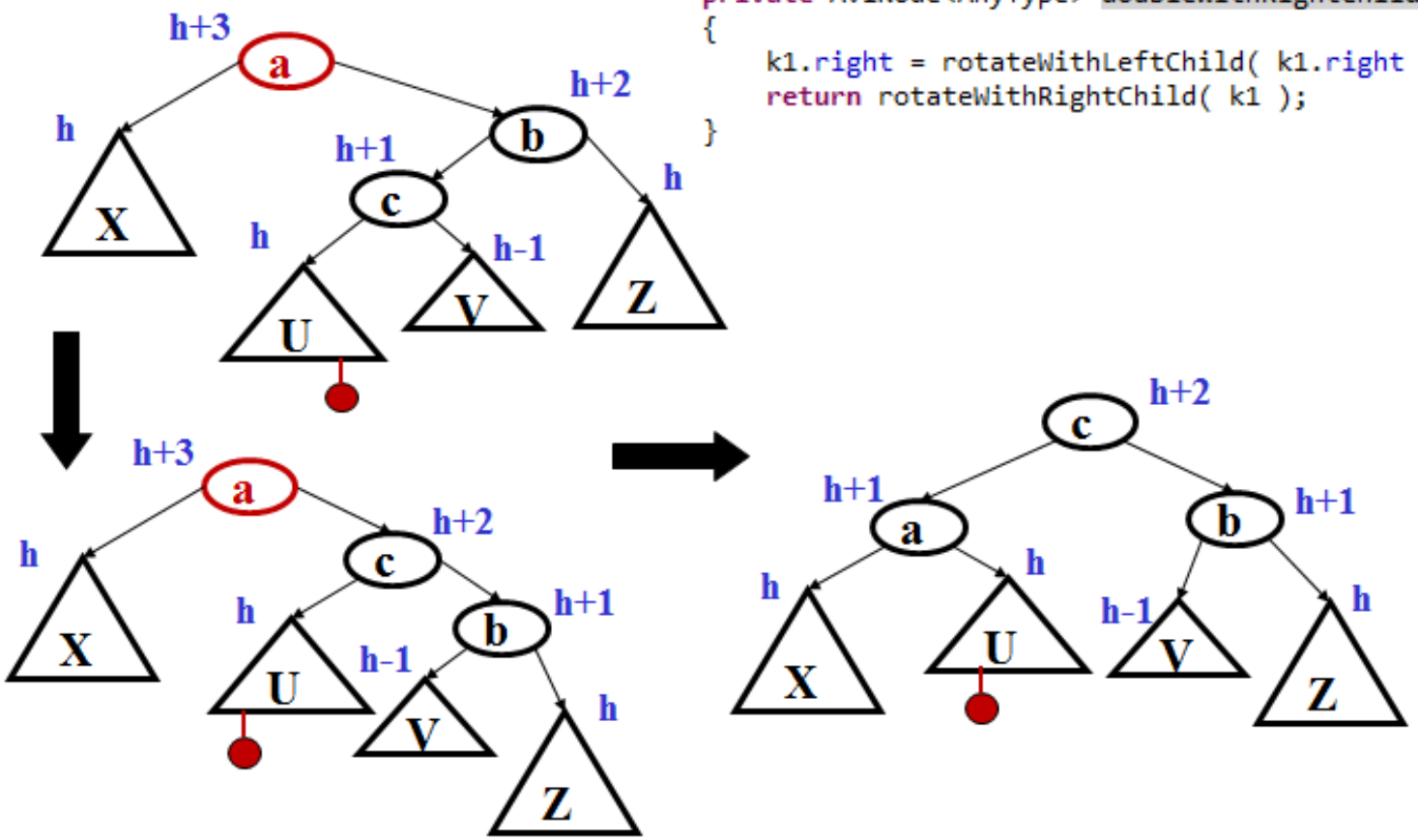
Node's left-left grandchild is too tall: rotateWithLeftChild

```
/**
 * Rotate binary tree node with left child.
 * For AVL trees, this is a single rotation for case 1.
 * Update heights, then return new root.
 */
private static AvlNode rotateWithLeftChild( AvlNode k2 )
{
    AvlNode k1 = k2.left;
    k2.left = k1.right;
    k1.right = k2;
    k2.height = max( height( k2.left ), height( k2.right ) ) + 1;
    k1.height = max( height( k1.left ), k2.height ) + 1;
    return k1;
}
```



Node's right-left grandchild is too tall: doubleWithLeftChild

```
/**
 * Double rotate binary tree node: first right child
 * with its left child; then node k1 with new right child.
 * For AVL trees, this is a double rotation for case 3.
 * Update heights, then return new root.
 */
private AvlNode<AnyType> doubleWithRightChild( AvlNode<AnyType> k1 )
{
    k1.right = rotateWithLeftChild( k1.right );
    return rotateWithRightChild( k1 );
}
```



Remove: BST+balance

```
/**
 * Internal method to remove from a subtree.
 * @param x the item to remove.
 * @param t the node that roots the subtree.
 * @return the new root of the subtree.
 */
private AvlNode<AnyType> remove( AnyType x, AvlNode<AnyType> t )
{
    if( t == null )
        return t;   // Item not found; do nothing

    int compareResult = x.compareTo( t.element );

    if( compareResult < 0 )
        t.left = remove( x, t.left );
    else if( compareResult > 0 )
        t.right = remove( x, t.right );
    else if( t.left != null && t.right != null ) // Two children
    {
        t.element = findMin( t.right ).element;
        t.right = remove( t.element, t.right );
    }
    else
        t = ( t.left != null ) ? t.left : t.right;
    return balance( t );
}
```

Example(Cont.)

- Insert:
  - 3,2,1,4,5,6,7
  - 16,15,14,13,12,11,10,8,9
- Delete(assume replace by leftMax)
  - 7,5

Q&A

- Thank you!