VOLUNTEER AT EOH 2026!

For EOH 2026, we're holding workshops to teach visitors important skills. We're planning on holding sessions to teach TinkerCAD, Python, and HTML & CSS. We're looking for volunteers to help teach these workshops. Furthermore, we also have many other volunteering requiring no pervious experience!

Workshops Volunteering

General Volunteering

ILLINOIS

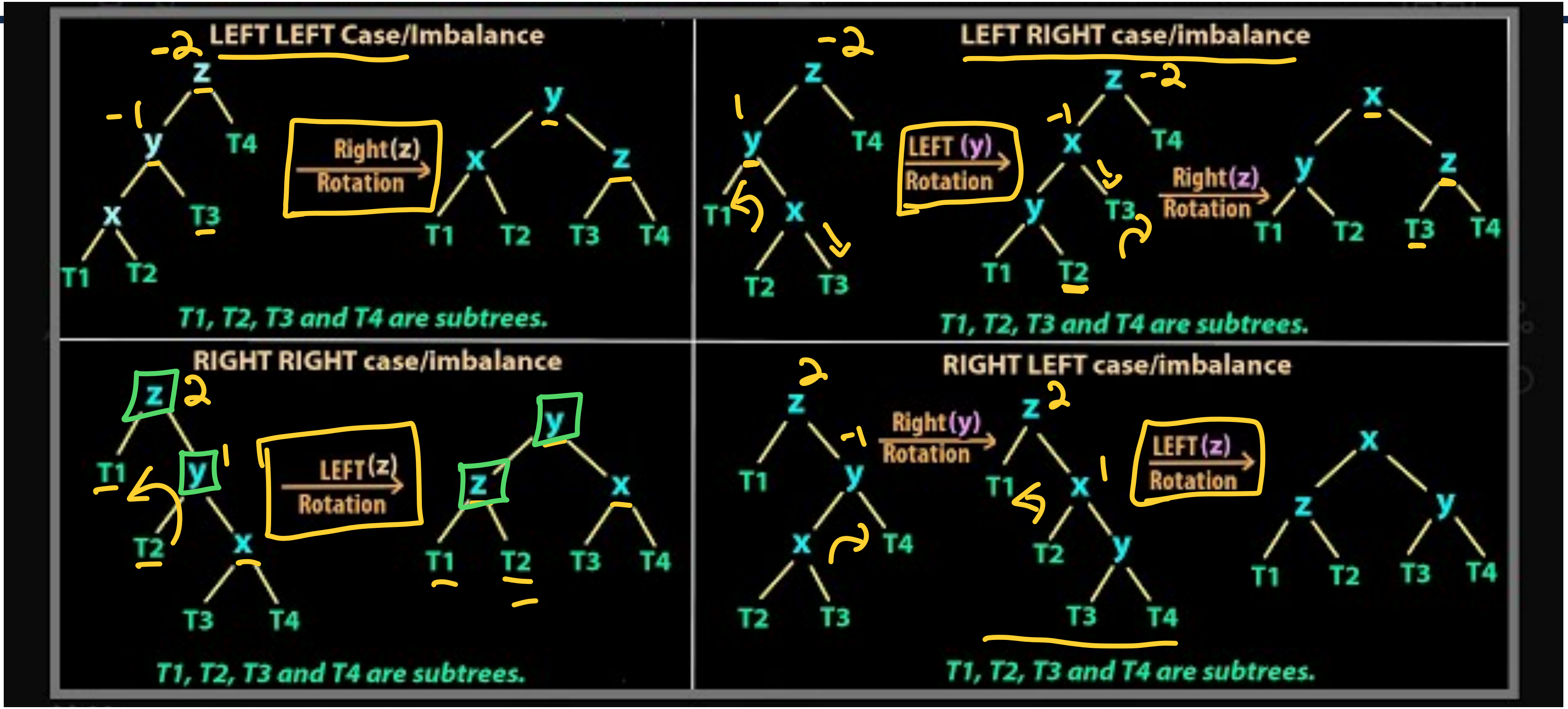
AVL Tree Implementation

Learning Objectives

1. Implement AVL Tree Insert
2. Implement AVL Tree Rebalance
3. Implement AVL Tree Rotations

4 rotations

Rotations Summary



T1, T2, T3 and T4 are subtrees.

T1, T2, T3 and T4 are subtrees.

T1, T2, T3 and T4 are subtrees.

T1, T2, T3 and T4 are subtrees.

Simple

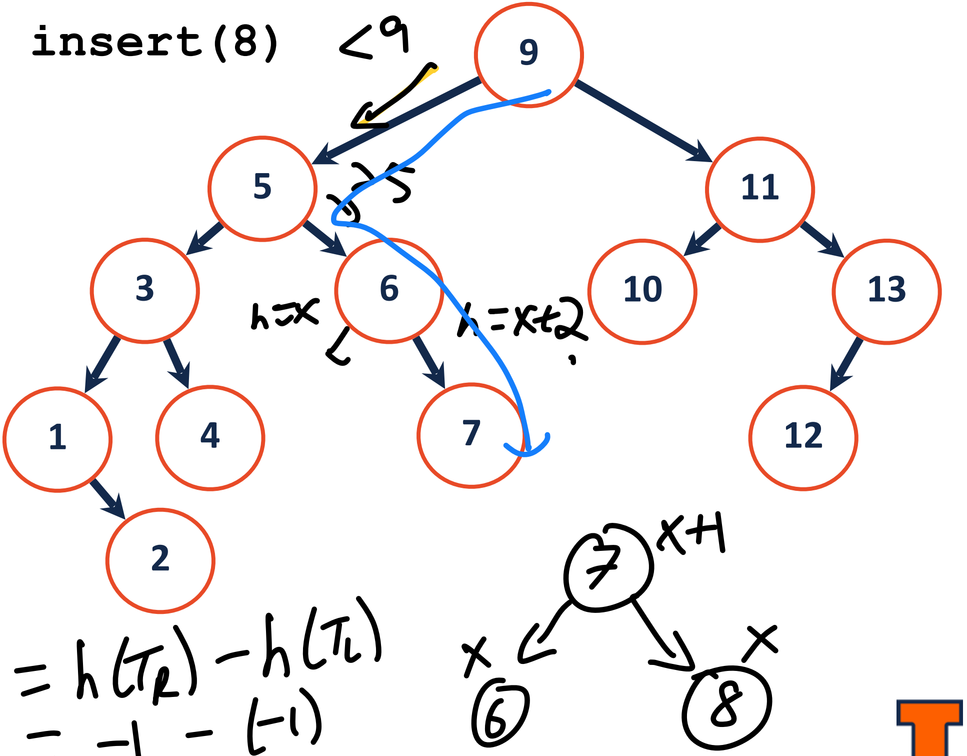
Complex

Insert in an AVL Tree

1. Find spot to insert node and insert node
2. Check if nodes are balanced recursively
3. Rotate to fix imbalance
4. Update heights of nodes

```
struct TreeNode {
  T key;
  unsigned height;
  TreeNode *left;
  TreeNode *right;
};
```

insert(8)



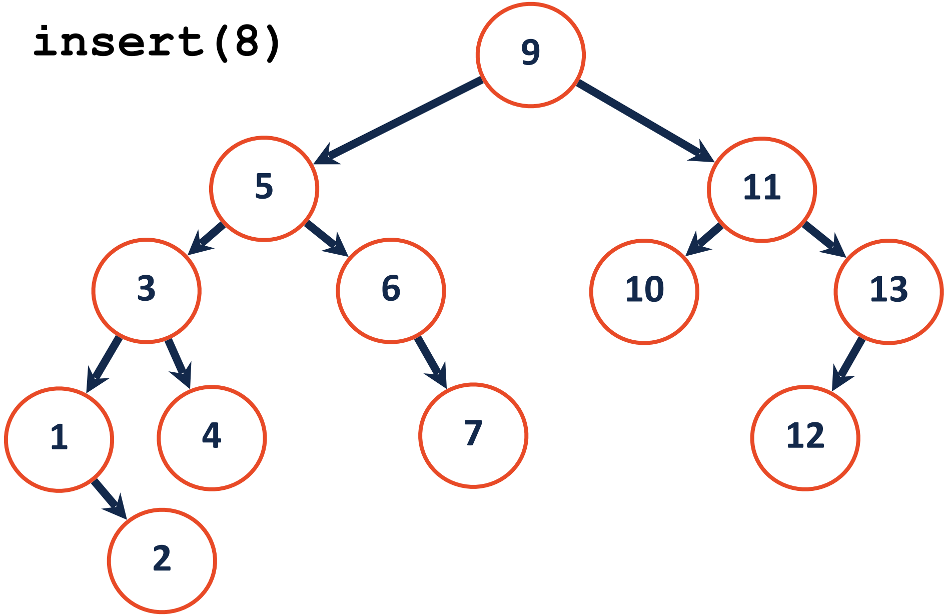
$$b = h(T_R) - h(T_L)$$
$$= -1 - (-1)$$
$$= 0$$

Insert in an AVL Tree

Insert (pseudo code):

```
1: Insert at proper place
2: Check for imbalance
3: Rotate, if necessary
4: Update height
```

`insert(8)`



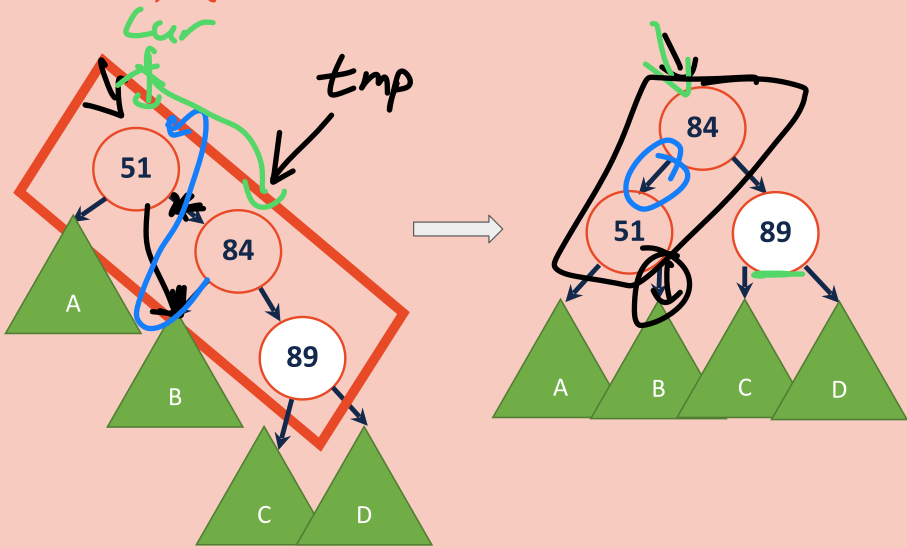
```
struct TreeNode {
  T key;
  unsigned height;
  TreeNode *left;
  TreeNode *right;
};
```

```
template <typename K, typename V>
void AVL<K, V>::_insert(const K & key, const V & data,
TreeNode *& cur) {
   if (cur == NULL){
     cur = new TreeNode(key, data); // Found location, insert node
   } else if (key < cur->key) {
     _insert( key, data, cur->left );// Recursive cases
   } else if (key > cur->key) {
     _insert( key, data, cur->right );//
   }
   _ensureBalance(cur);
}
```

```
template <typename K, typename V>
void AVL<K, V>::_ensureBalance(TreeNode *& cur) {
  // Calculate the balance factor:
  int balance = height(cur->right) - height(cur->left);

  // Check if the node is currently not in balance:
  if ( balance == -2 ) { ← heavy to left
    int l_balance =   // check left subtree balance
        height(cur->left->right) - height(cur->left->left);
    if ( l_balance == -1 ) { ____Right Rotation____; }
    else                   { ____Left Right Rotation____; }
  } else if ( balance == 2 ) { ← heavy to right
    int r_balance = // check right subtree balance
        height(cur->right->right) - height(cur->right->left);
    if( r_balance == 1 ) { ____Left Rotation____; }
    else                 { ____Right- Left Rotation____; }
  }

  _updateHeight(cur);
};
```

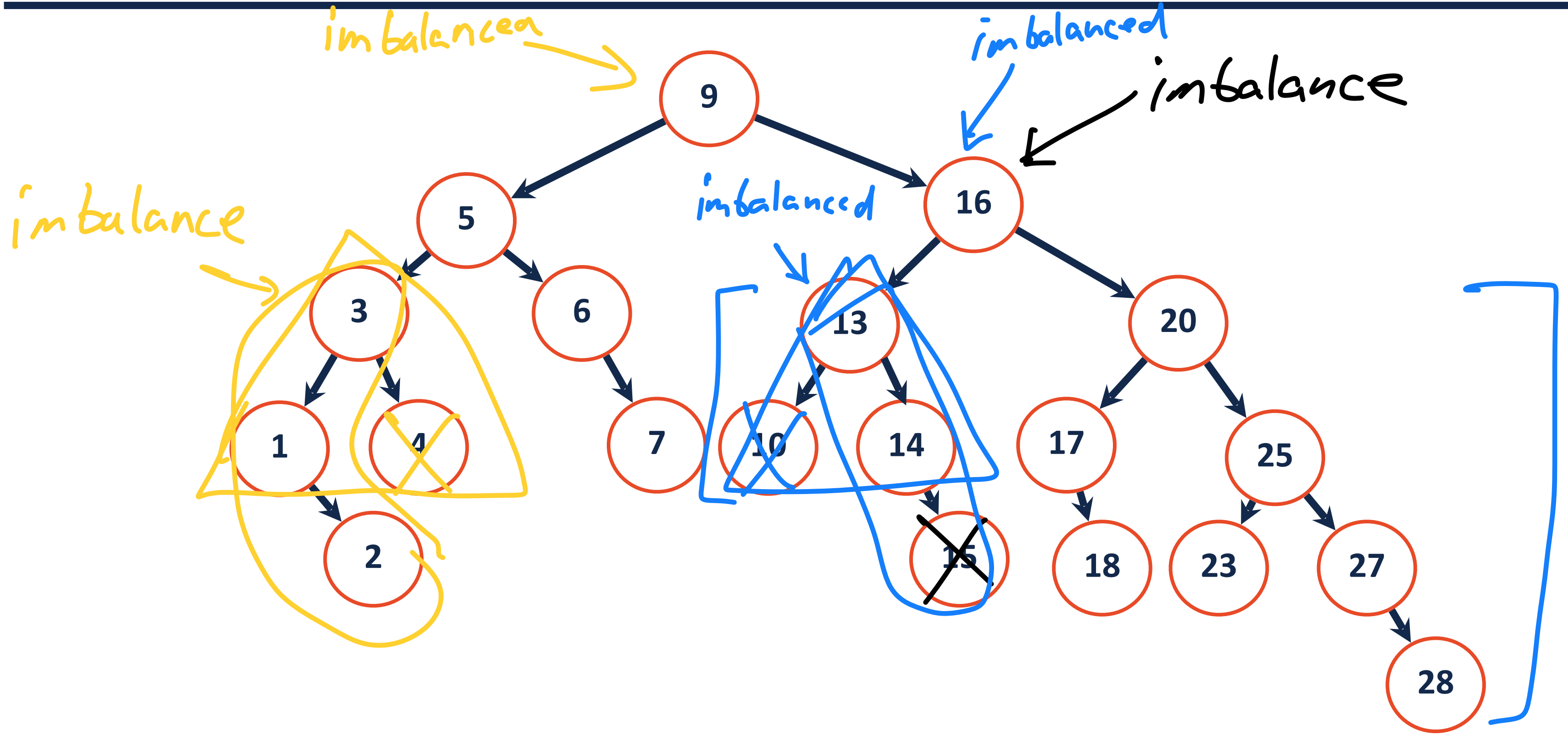
Rotations

```
template <typename K, typename V>
void AVL<K, V>::rotateLeft(TreeNode *& cur) {
  // Modify Pointers
      TreeNode* tmp =cur->right
      cur->right =cur->right->left;
      tmp->left = cur

      cur = temp

  // Update Heights
        cur -> height
        cur -> left -> height

};
```



```
template <typename K, typename V>
void AVL<K, V>::_remove(const K & key, TreeNode *& cur) {
      // Base Case: If cur is null (element isn't there)

      // If key is less than current key,
            // remove in left subtree
            // rebalance
      // If key is greater than current key,
            // remove in right subtree
            // rebalance

      // If key is equal to current key,
      // 0 Child Case
            // rebalance

            // 1 Child Case
            // rebalance

            // 2 Child Case
            // rebalance
}
```

Which node could I remove to cause multiple rotations?



imbalanced
imbalanced
imbalance
imbalance
imbalanced
9
5
16
3
6
13
20
1
4
7
10
14
17
25
2
15
18
23
27
28