Data Structures

AVL Trees

CS 225

September 25, 2023

Brad Solomon & G Carl Evans

UNIVERSITY OF
ILLINOIS
URBANA-CHAMPAIGN

Department of Computer Science

Learning Objectives

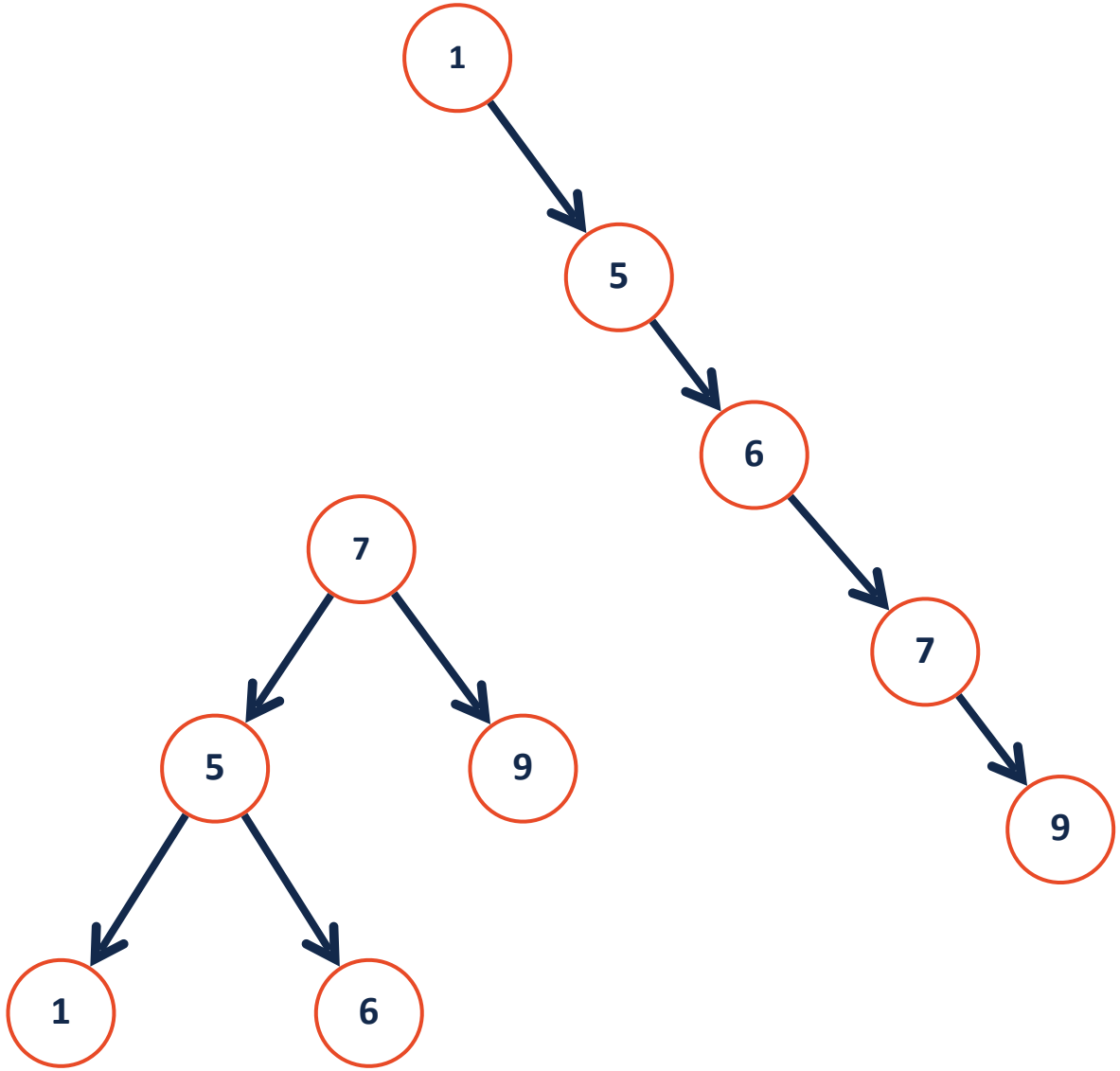
Review why we need balanced trees

Review what an AVL rotation does

Explore the four possible rotations for an AVL tree

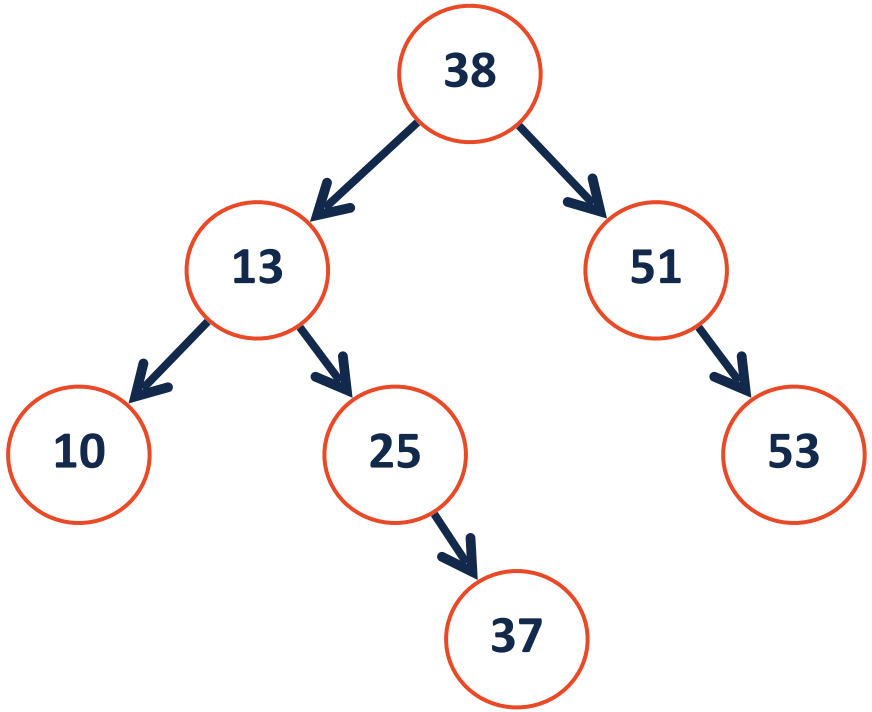
BST Analysis – Running Time

	BST Worst Case
find	O(h)
insert	O(h)
delete	O(h)
traverse	O(n)

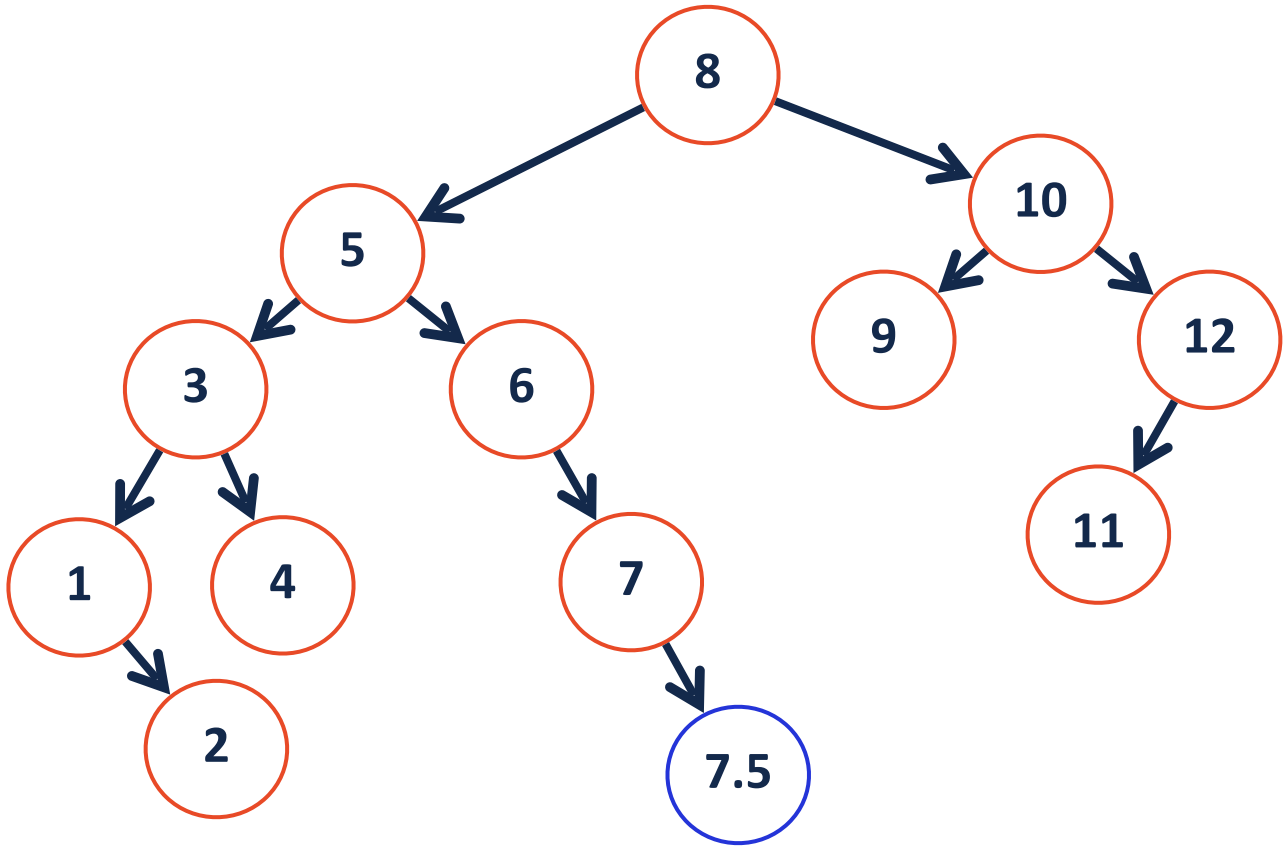


AVL-Tree: A self-balancing binary search tree

Every node in an AVL tree has a balance of:

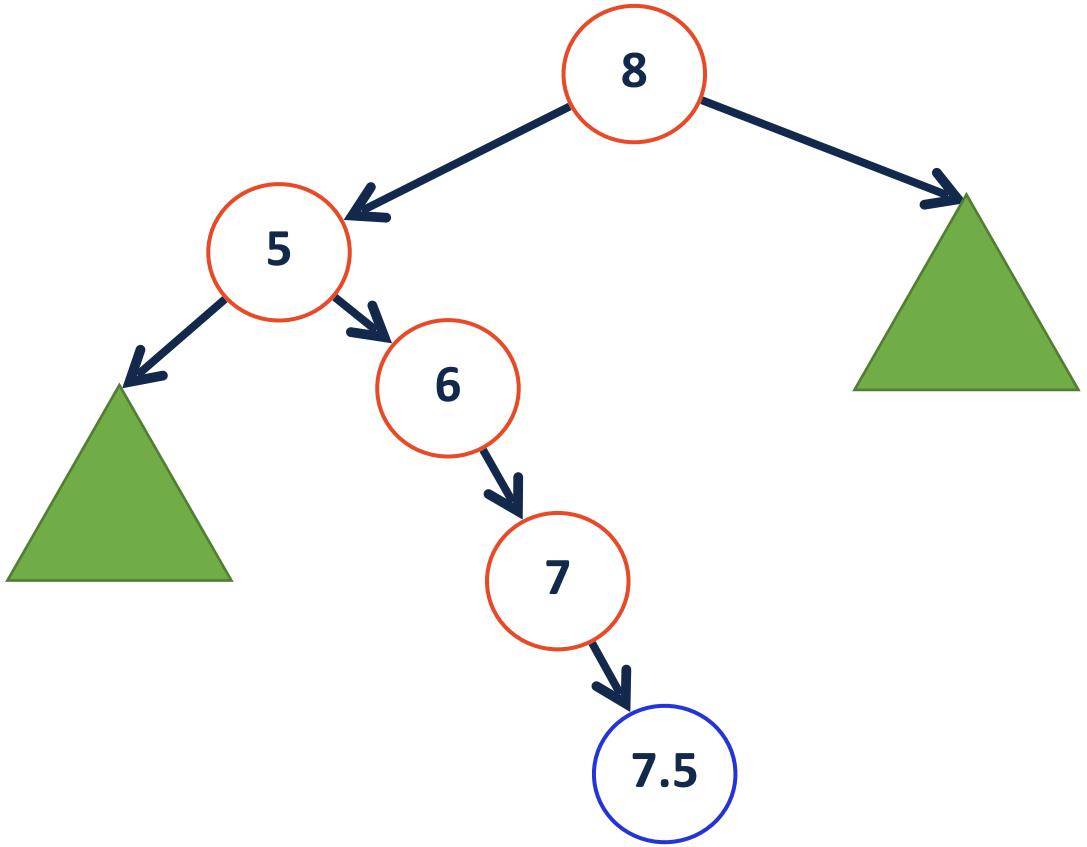


Left Rotation



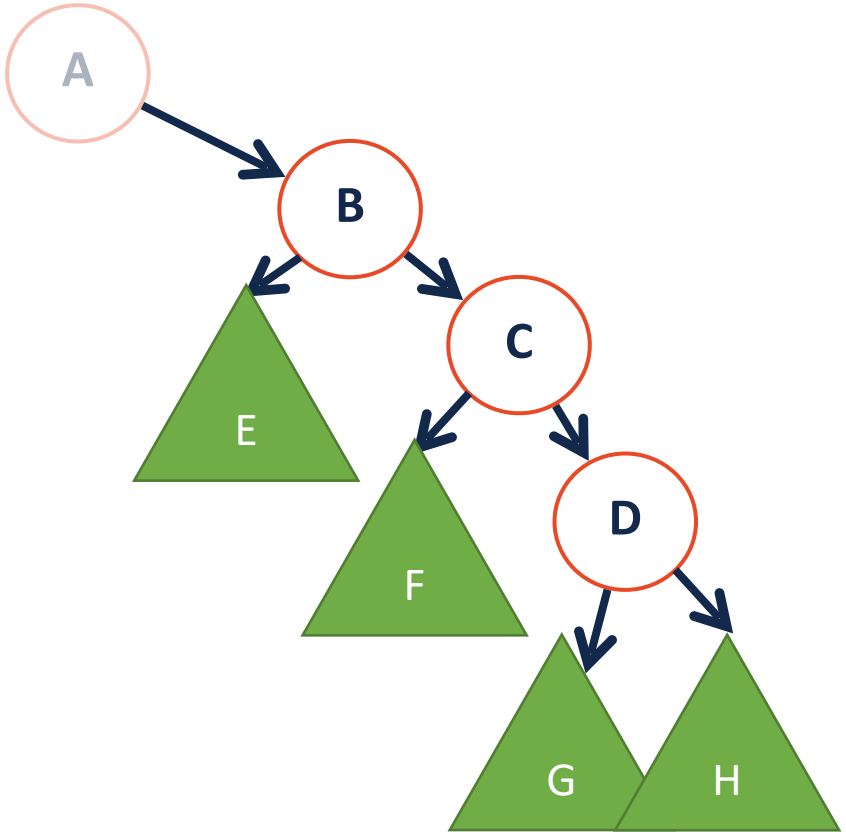
Left Rotation

All rotations are local (subtrees are not impacted)

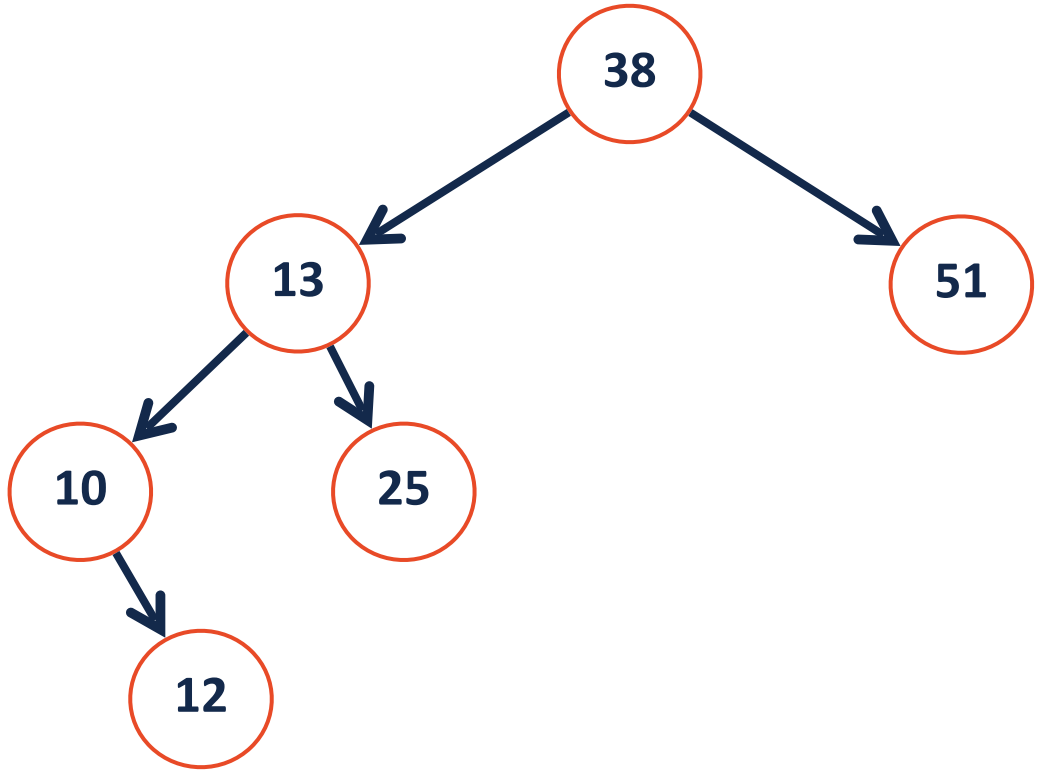


Left Rotation

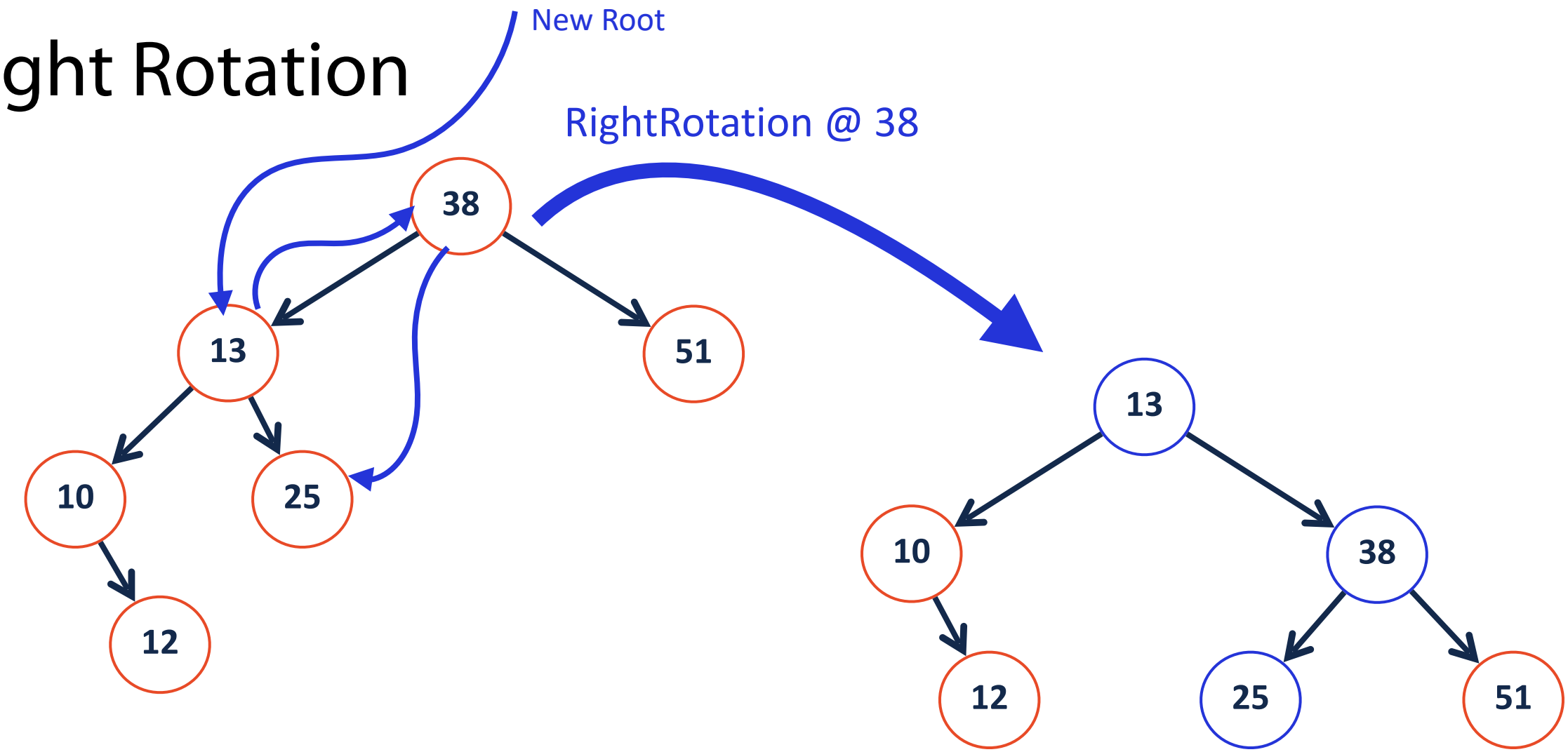
All rotations preserve BST property



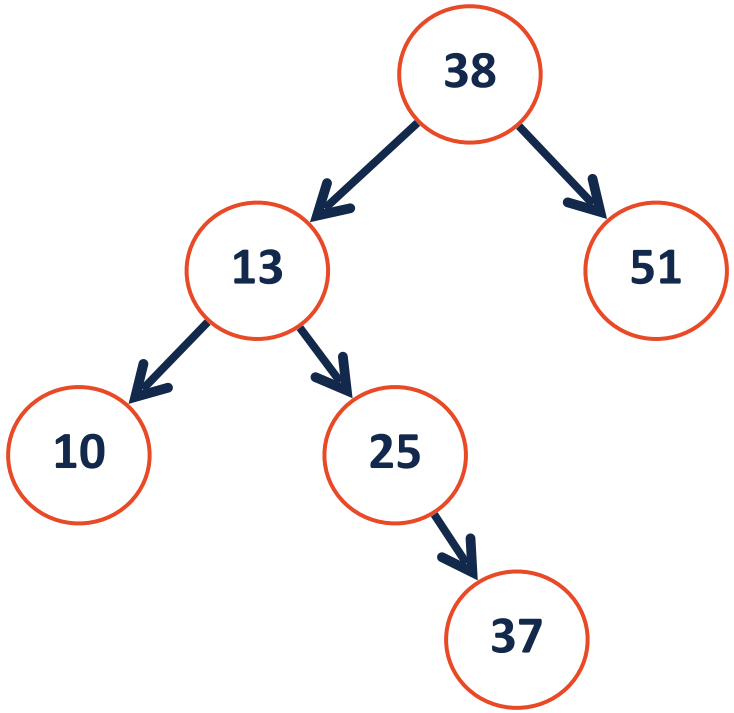
Right Rotation



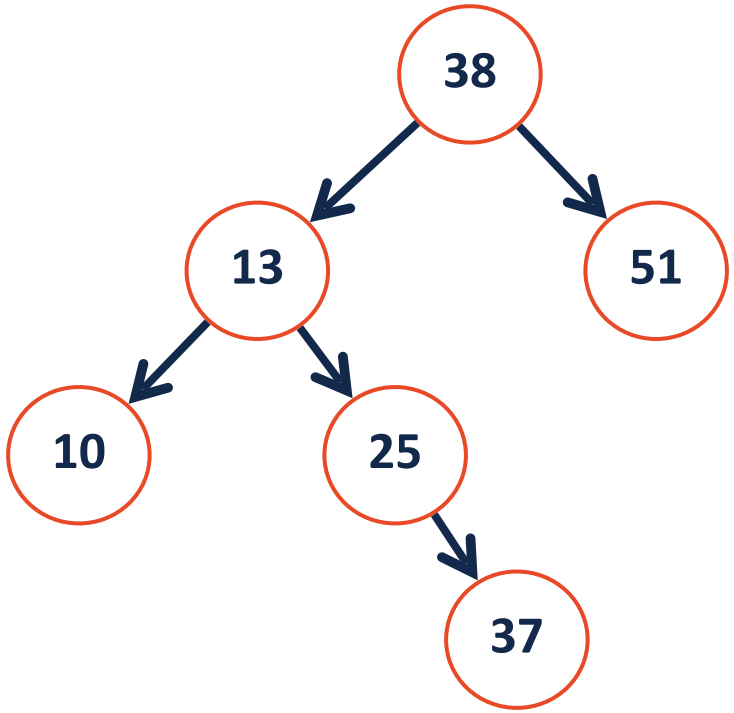
Right Rotation

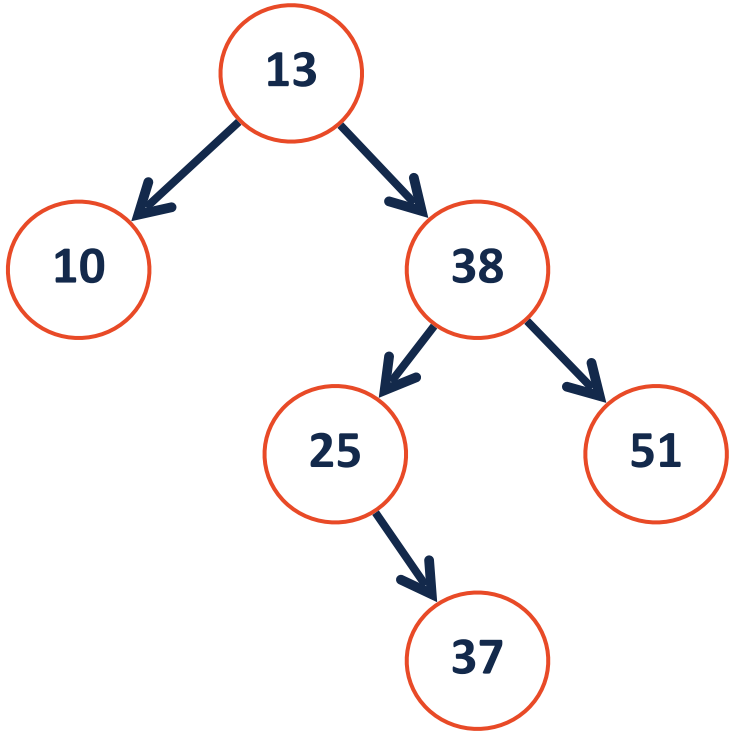


AVL Rotation Practice



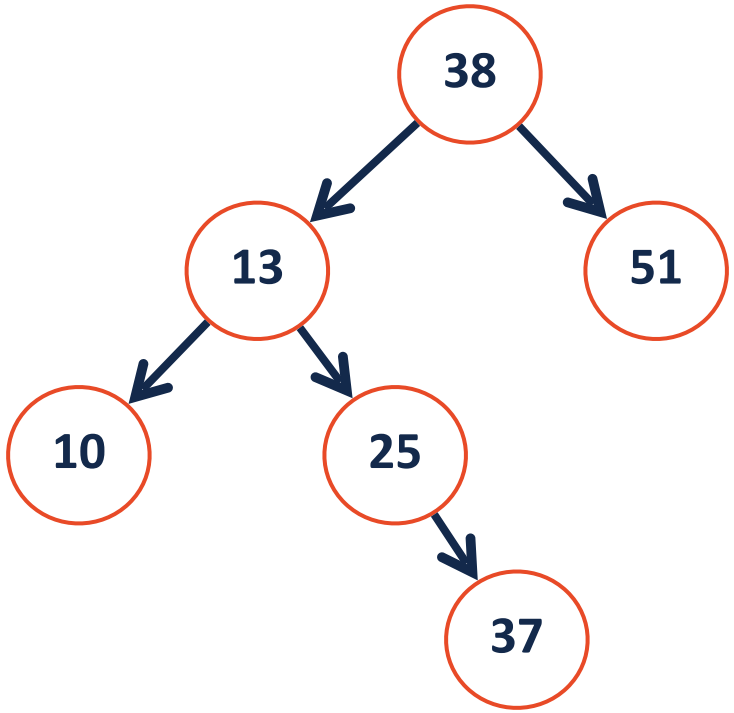
AVL Rotation Practice



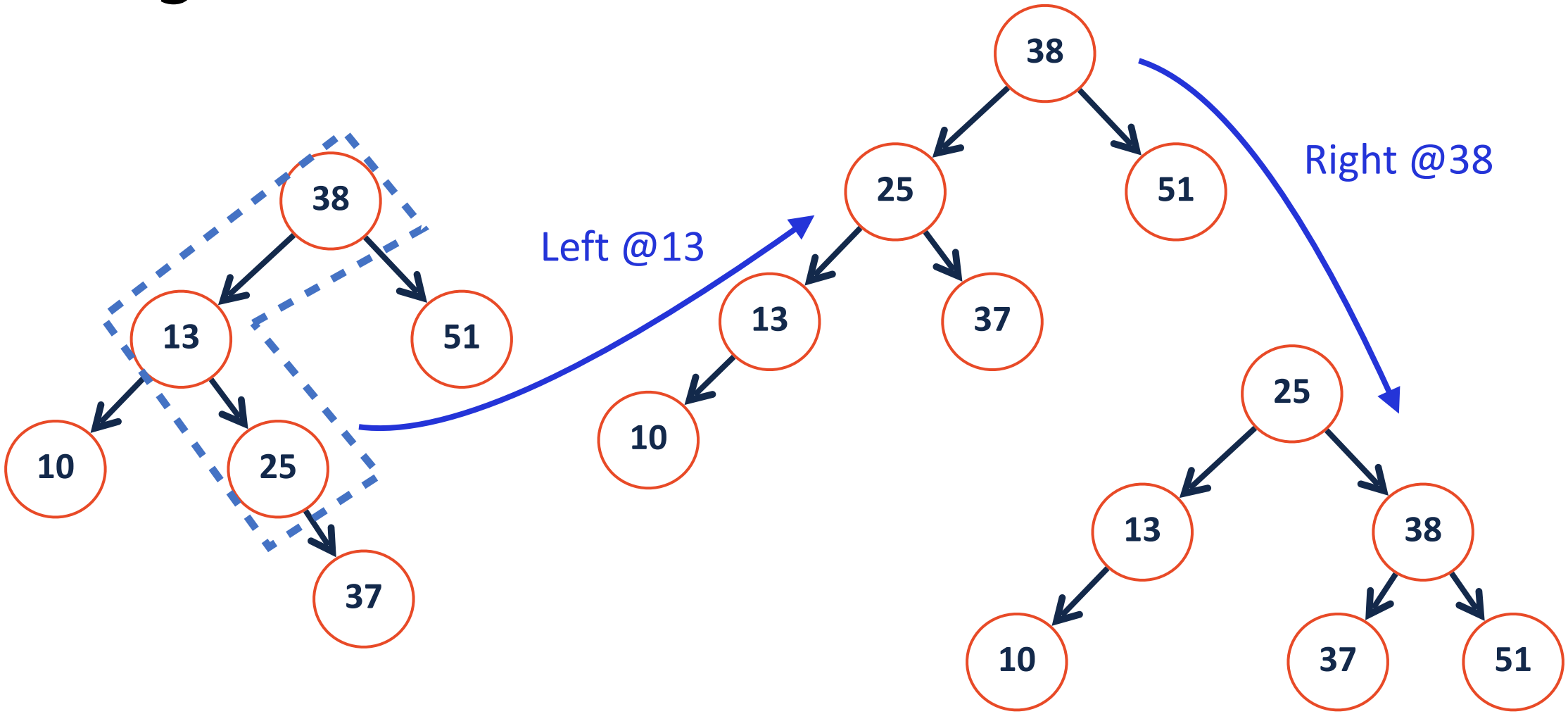


Somethings not quite right…

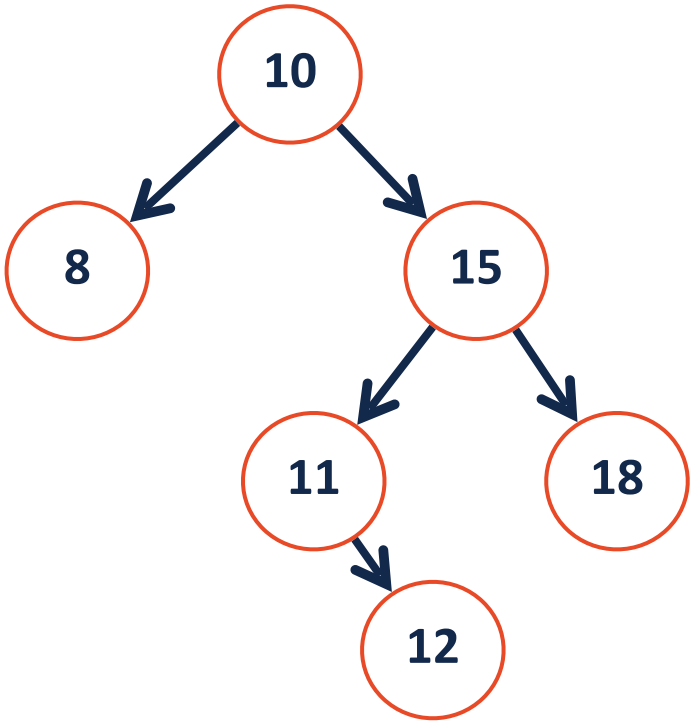
LeftRight Rotation



LeftRight Rotation

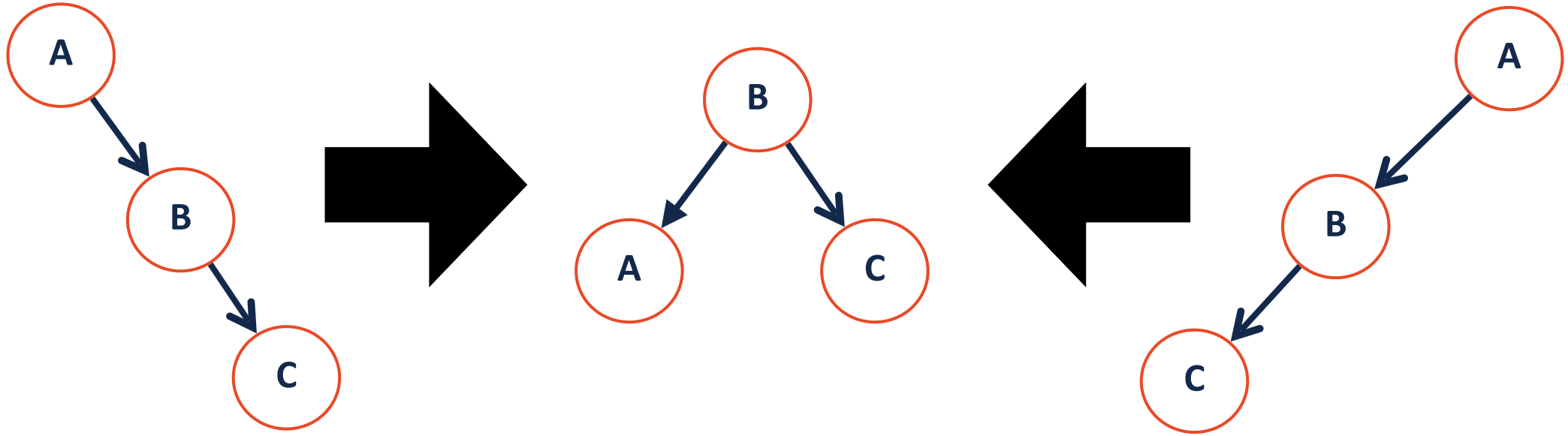


RightLeft Rotation



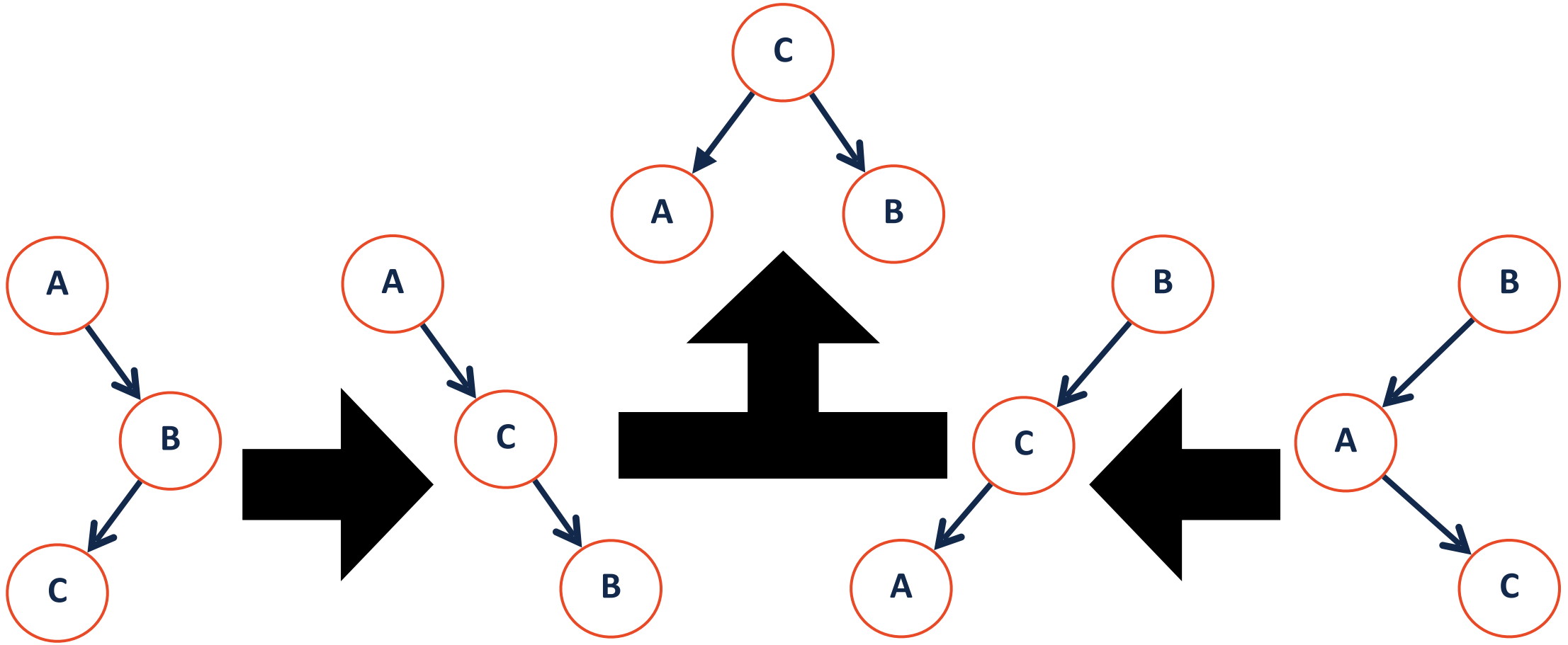
AVL Rotations

Left and right rotation convert **sticks** into **mountains**



AVL Rotations

LeftRight (RightLeft) convert **elbows** into **sticks** into **mountains**



AVL Rotations

Four kinds of rotations: (L, R, LR, RL)

1. All rotations are local (subtrees are not impacted)

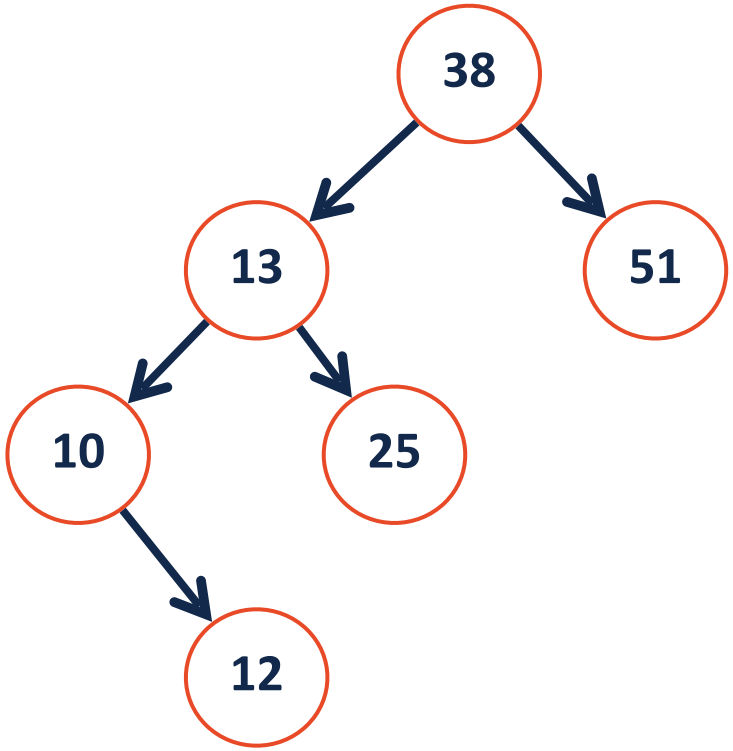
2. The running time of rotations are constant

3. The rotations maintain BST property

Goal:

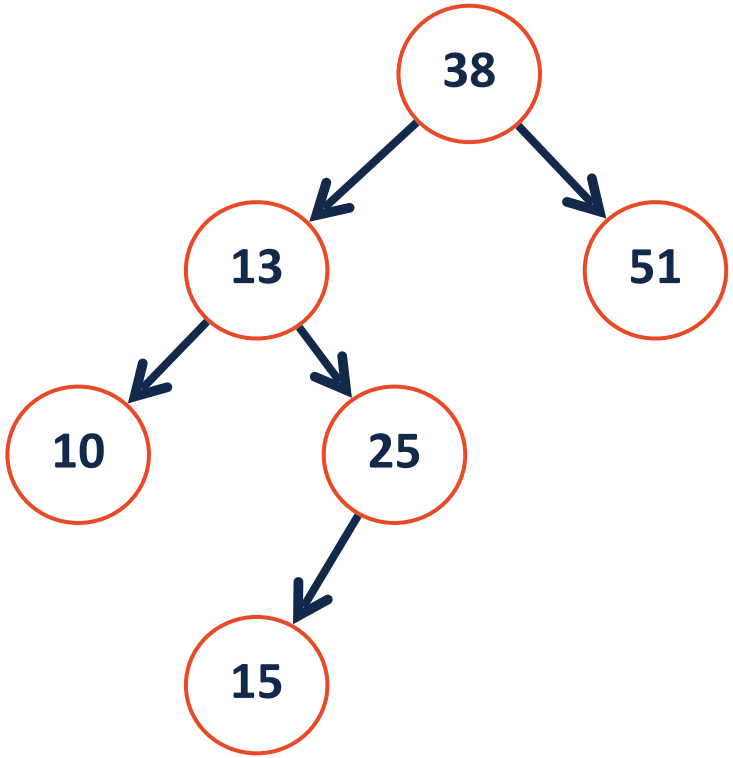
AVL Rotations

We can identify which rotation to do using **balance**

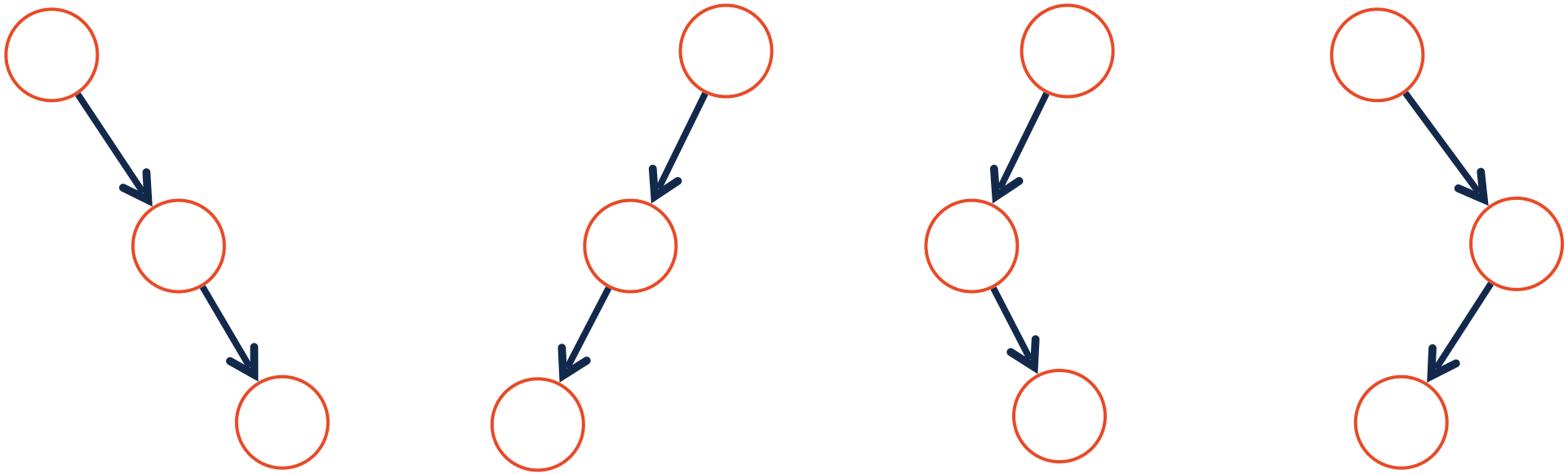


AVL Rotations

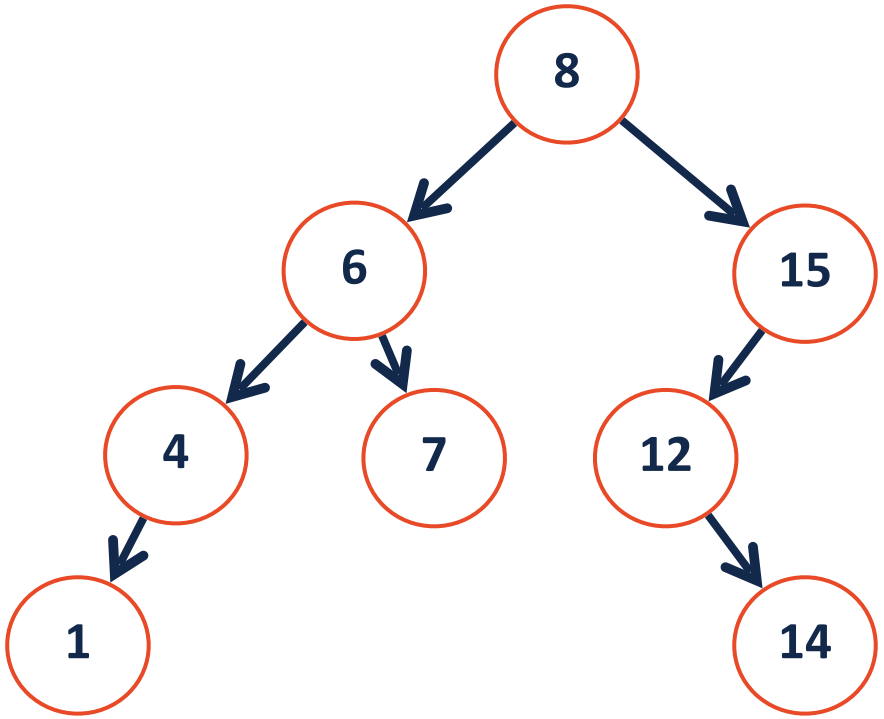
We can identify which rotation to do using **balance**



AVL Rotations



AVL Rotation Practice



AVL vs BST ADT

The AVL tree is a modified binary search tree that rotates **when necessary**

```
1  struct TreeNode {
2    T key;
3    unsigned height;
4    TreeNode *left;
5    TreeNode *right;
6  };
```

How does the constraint on balance affect the core functions?

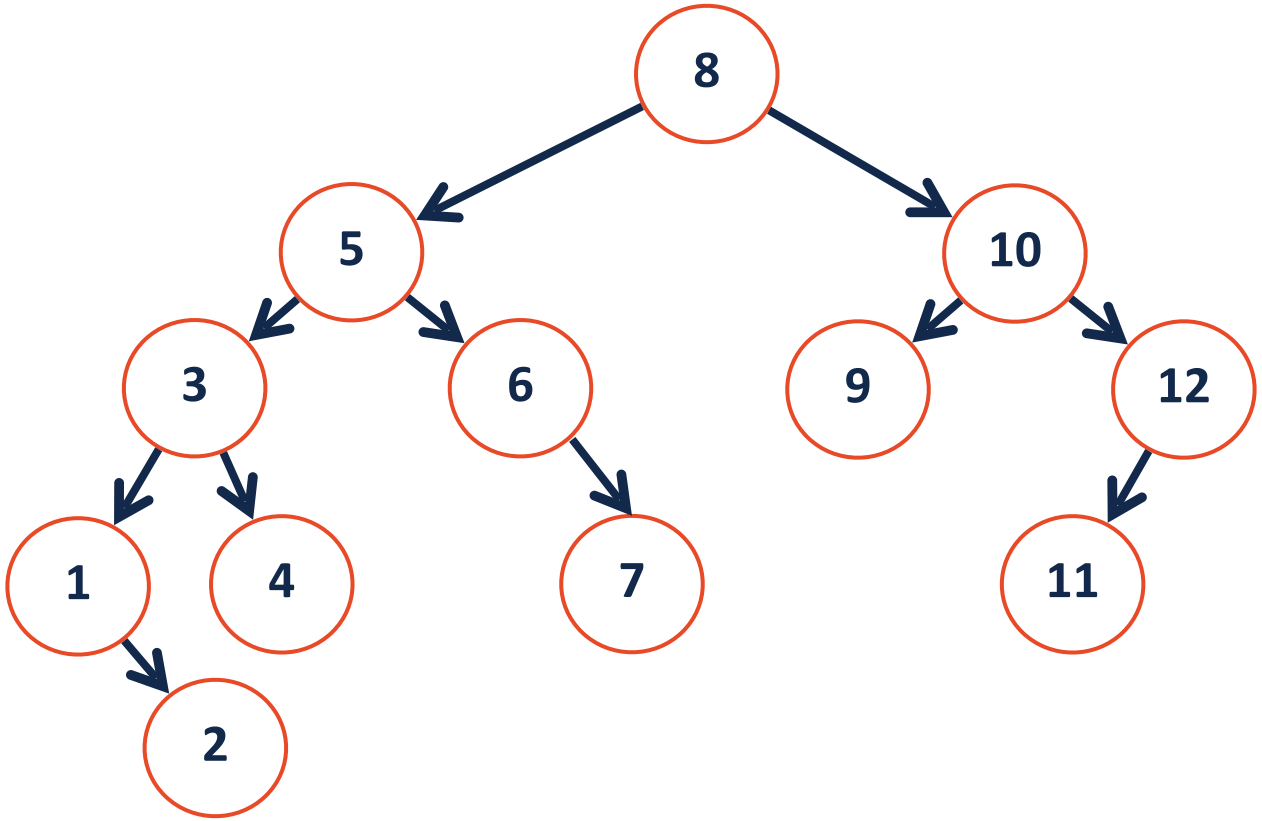
Find

Insert

Remove

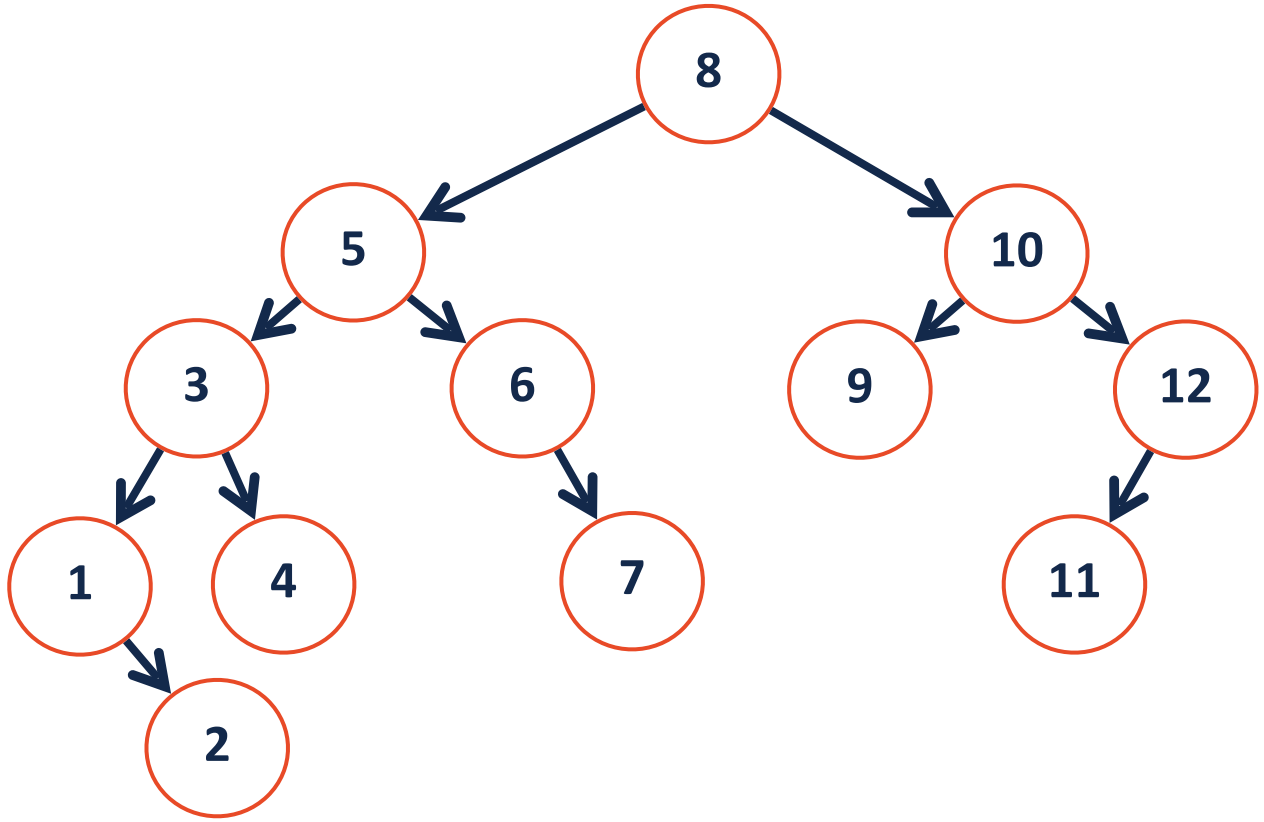
AVL Find

`_find(7)`



AVL Insertion

_insert(6.5)



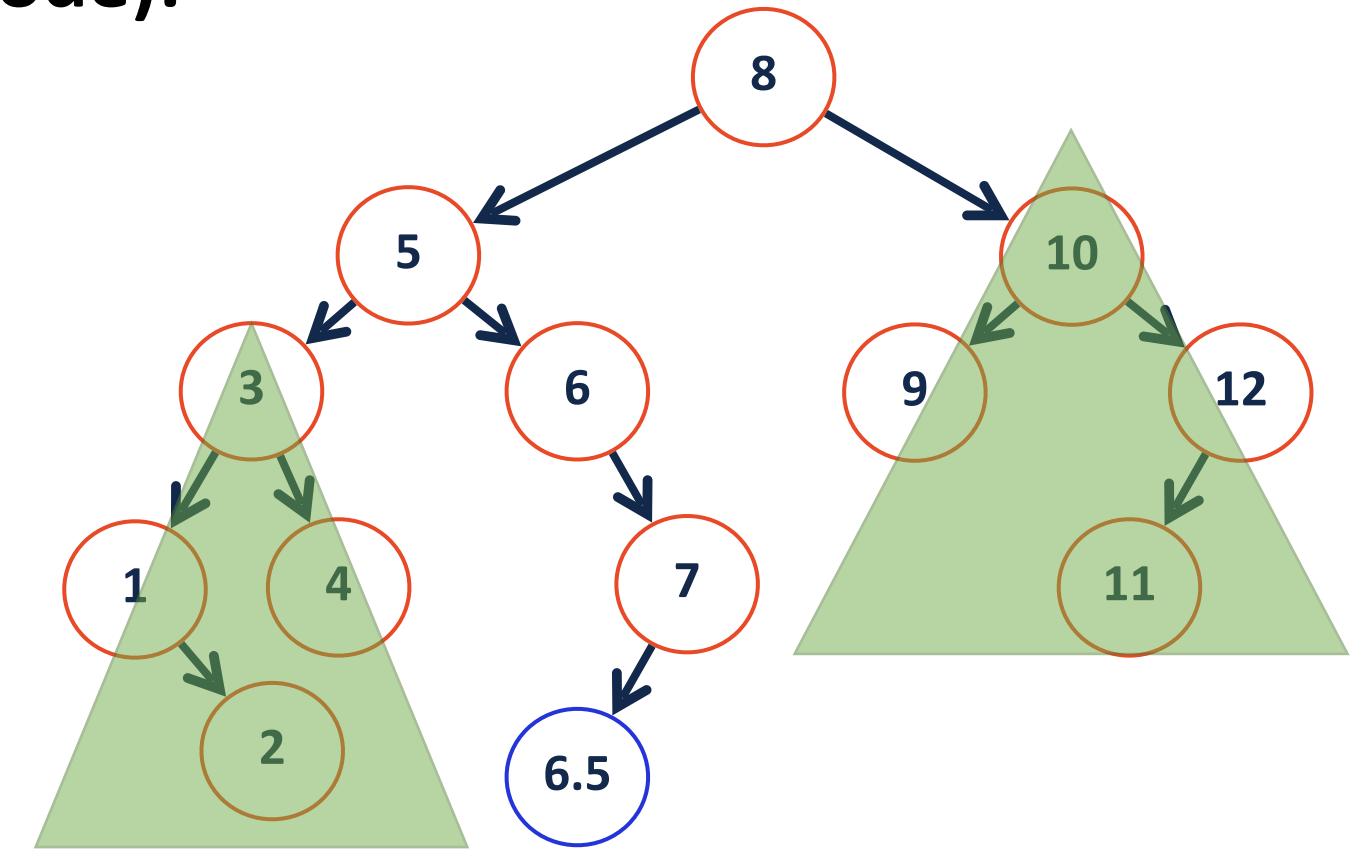

```
1  struct TreeNode {
2    T key;
3    unsigned height;
4    TreeNode *left;
5    TreeNode *right;
6  };
```

AVL Insertion

_insert(6.5)

Insert (recursive pseudo code):

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



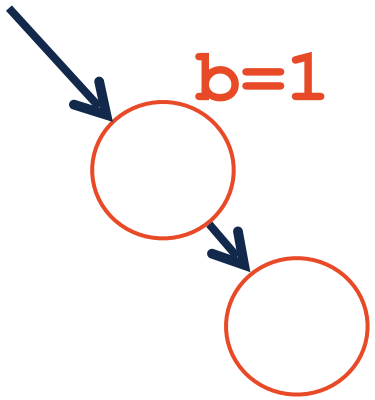

```
1  struct TreeNode {
2    T key;
3    unsigned height;
4    TreeNode *left;
5    TreeNode *right;
6  };
```

```
151 template <typename K, typename V>
152 void AVL<K, D>::_insert(const K & key, const V & data, TreeNode
    *& cur) {
153   if (cur == NULL)           { cur = new TreeNode(key, data);   }
157   else if (key < cur->key) { _insert( key, data, cur->left ); }
160   else if (key > cur->key) { _insert( key, data, cur->right );}
166   _ensureBalance(cur);
167 }
```

```
119 template <typename K, typename V>
120 void AVL<K, D>::_ensureBalance(TreeNode *& cur) {
121   // Calculate the balance factor:
122   int balance = height(cur->right) - height(cur->left);
123
124   // Check if the node is current not in balance:
125   if ( balance == -2 ) {
126     int l_balance =
            height(cur->left->right) - height(cur->left->left);
127     if ( l_balance == -1 ) { _____________________________________; }
128     else                   { _____________________________________; }
129   } else if ( balance == 2 ) {
130     int r_balance =
            height(cur->right->right) - height(cur->right->left);
131     if( r_balance == 1 ) { _____________________________________; }
132     else                 { _____________________________________; }
133   }
134
135   _updateHeight(cur);
136 };
```

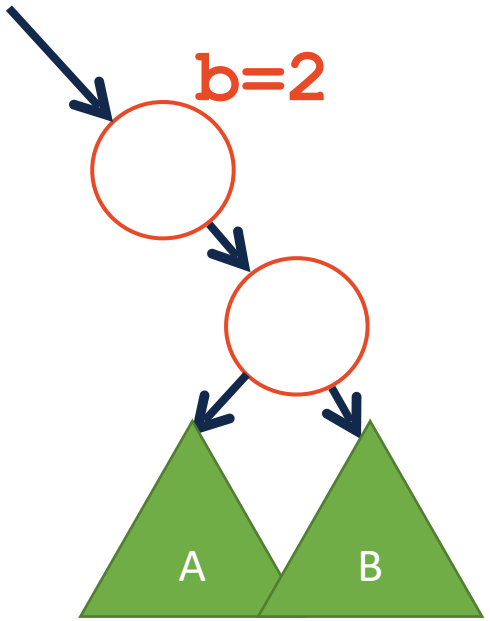
AVL Insertion

Given an AVL is balanced, insert can insert **at most** one imbalance



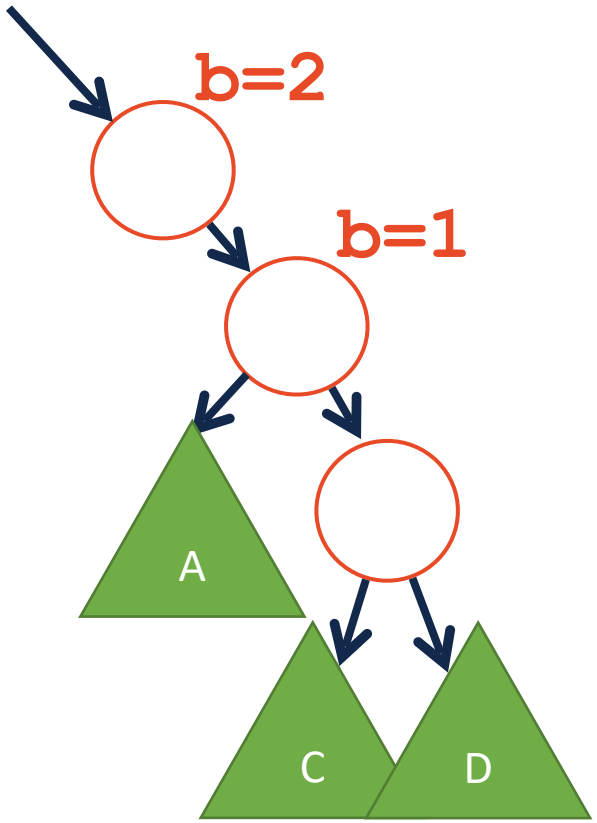
AVL Insertion

Given an AVL is balanced, insert can insert **at most** one imbalance



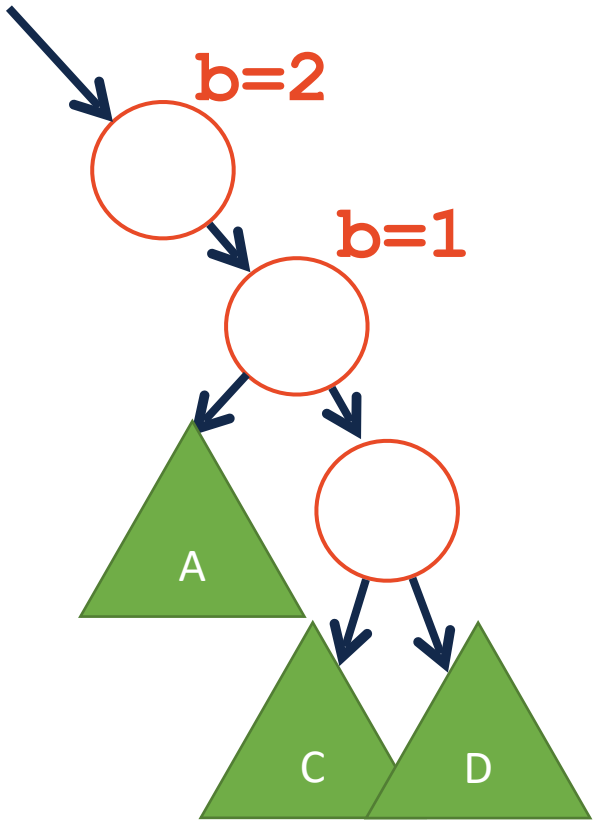
AVL Insertion

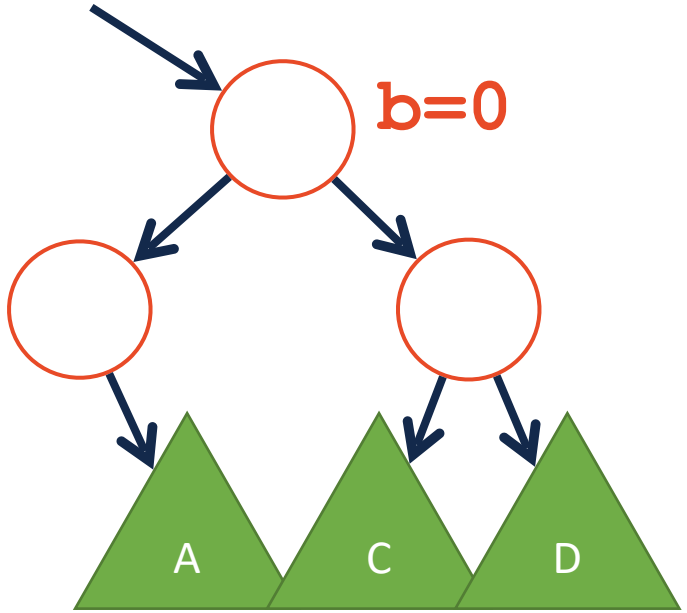
If we insert in B, I must have a balance pattern of **2, 1**



AVL Insertion

A **left** rotation fixes our imbalance in our local tree.

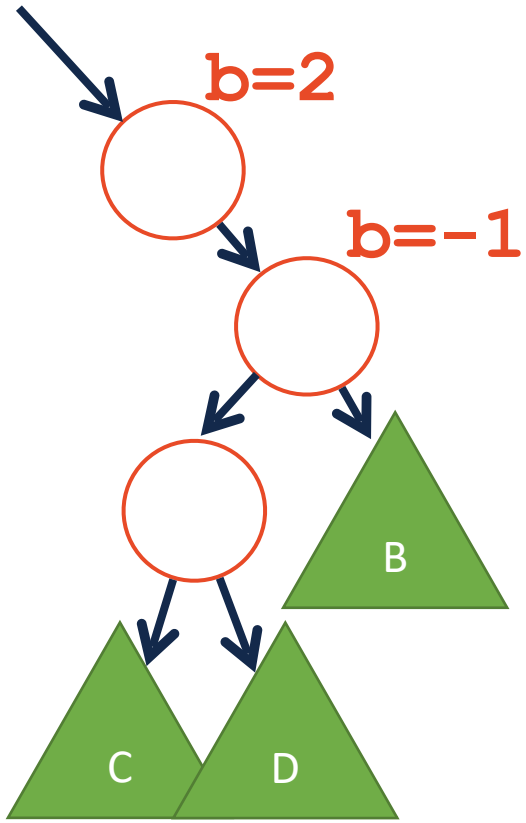




After rotation, subtree has **pre-insert height**. (Overall tree is balanced)

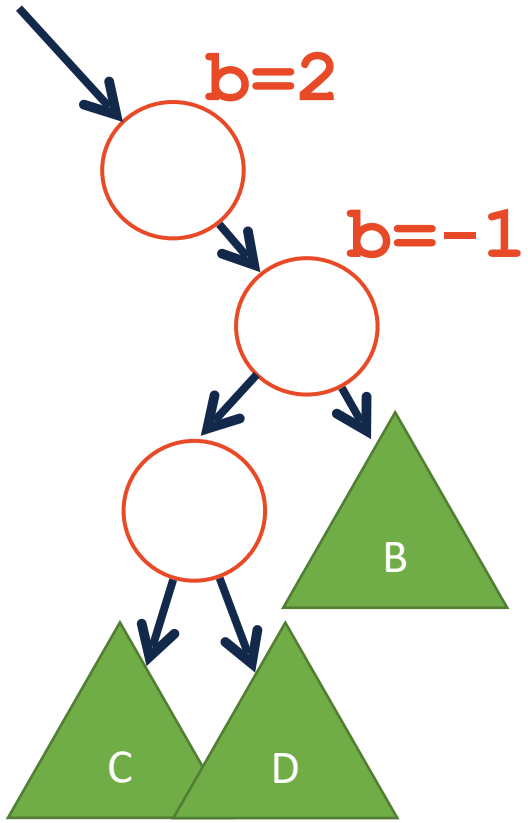
AVL Insertion

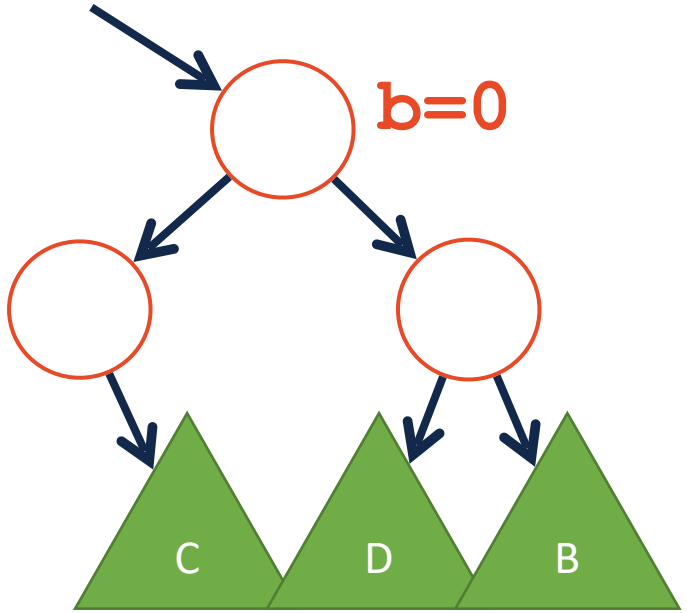
If we insert in A, I must have a balance pattern of **2, -1**



AVL Insertion

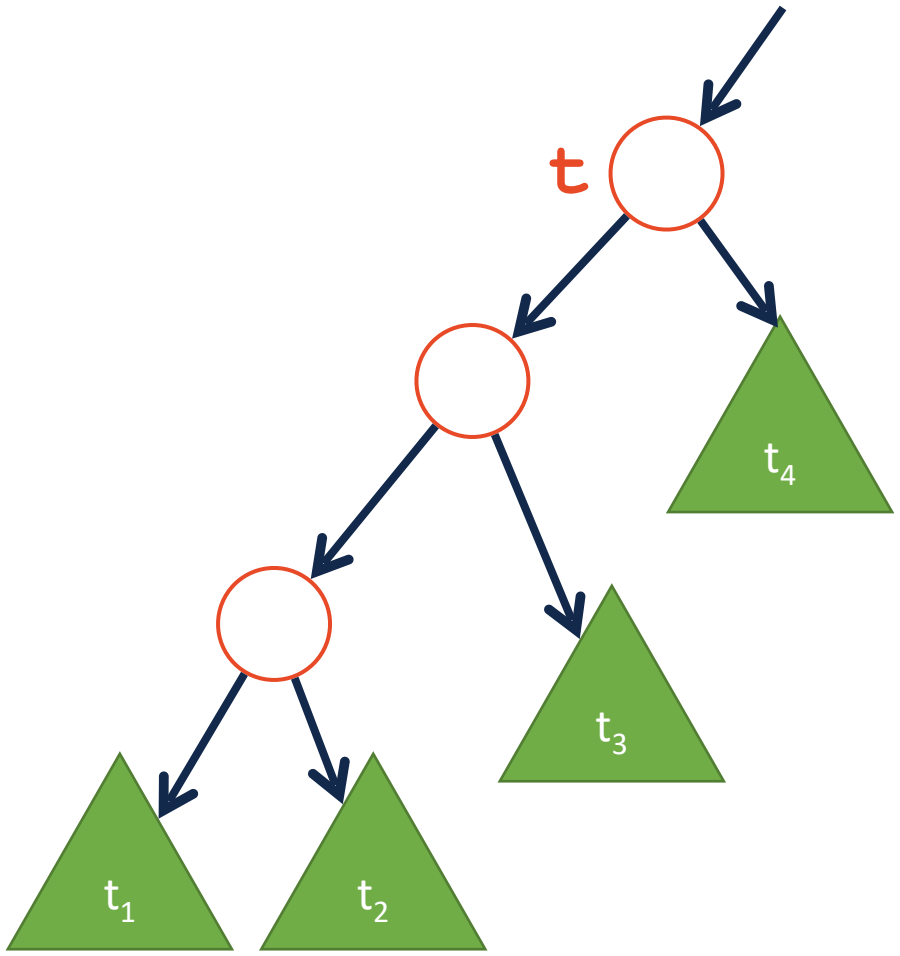
A **rightLeft** rotation fixes our imbalance in our local tree.





After rotation, subtree has **pre-insert height**. (Overall tree is balanced)

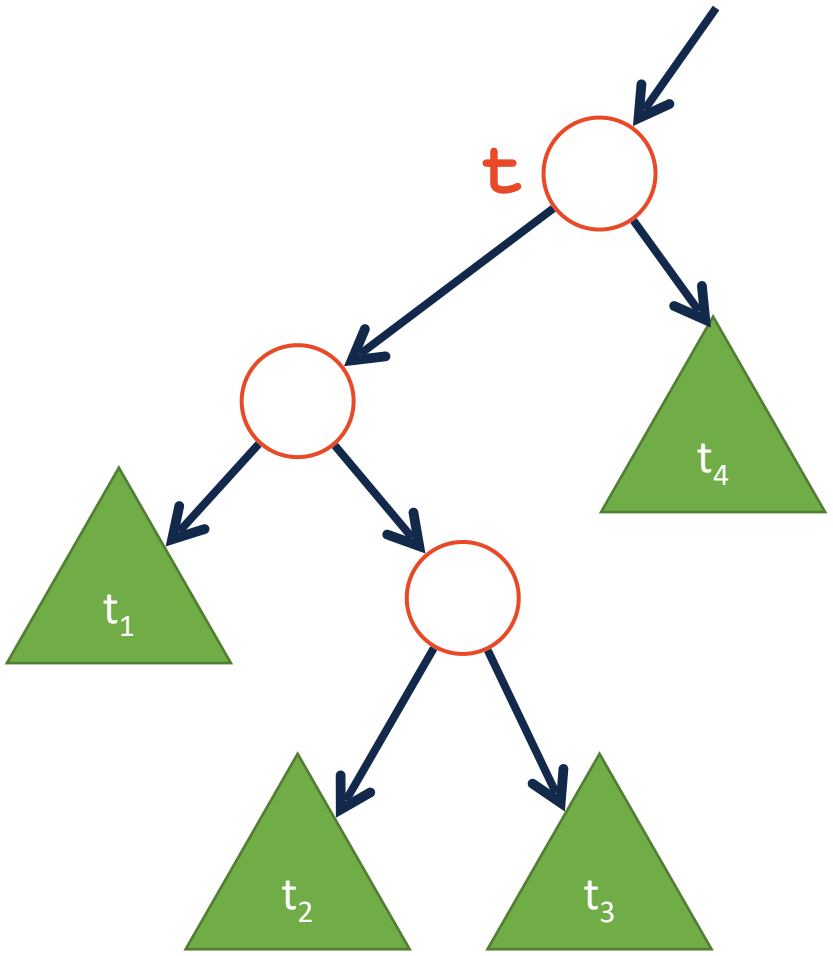
AVL Insertion



Theorem:
If an insertion occurred in subtrees $\mathbf{t_1}$ or $\mathbf{t_2}$ and an imbalance was first detected at **t**, then a _____ rotation about **t** restores the balance of the tree.

We gauge this by noting the balance factor of **t is** _____ and the balance factor of **t->left** is _____.

AVL Insertion



Theorem:
If an insertion occurred in subtrees $\mathbf{t_2}$ or $\mathbf{t_3}$ and an imbalance was first detected at **t**, then a _____ rotation about **t** restores the balance of the tree.

We gauge this by noting the balance factor of **t is** _____ and the balance factor of **t->left** is _____.

AVL Insertion

We've seen every possible insert that can cause an imbalance

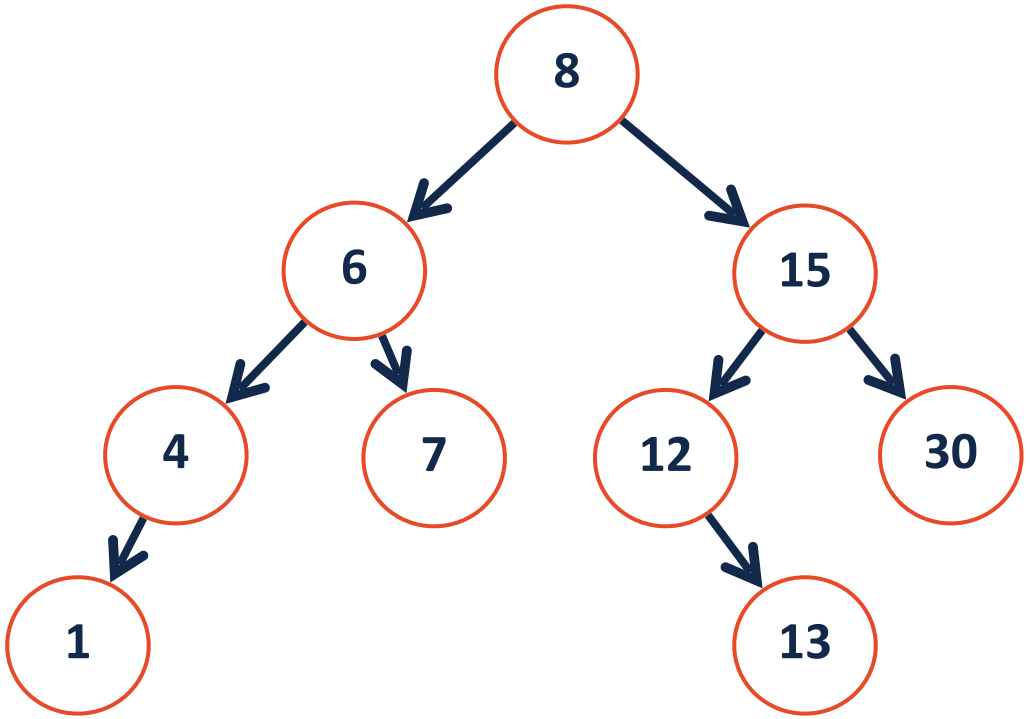
Insert increases height by at most:

A rotation reduces the height of the subtree by:

A single* rotation restores balance and corrects height!

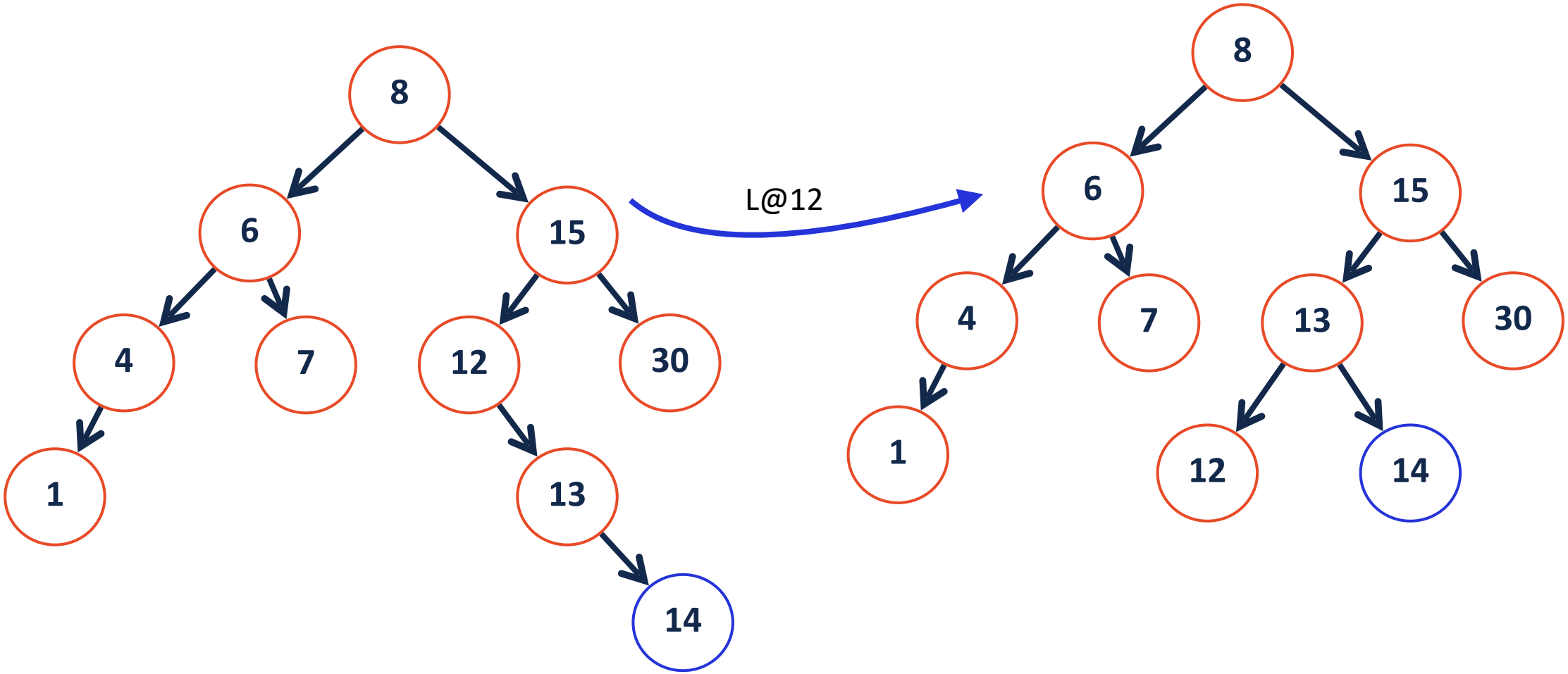
AVL Insertion Practice

```
_insert(14)
```



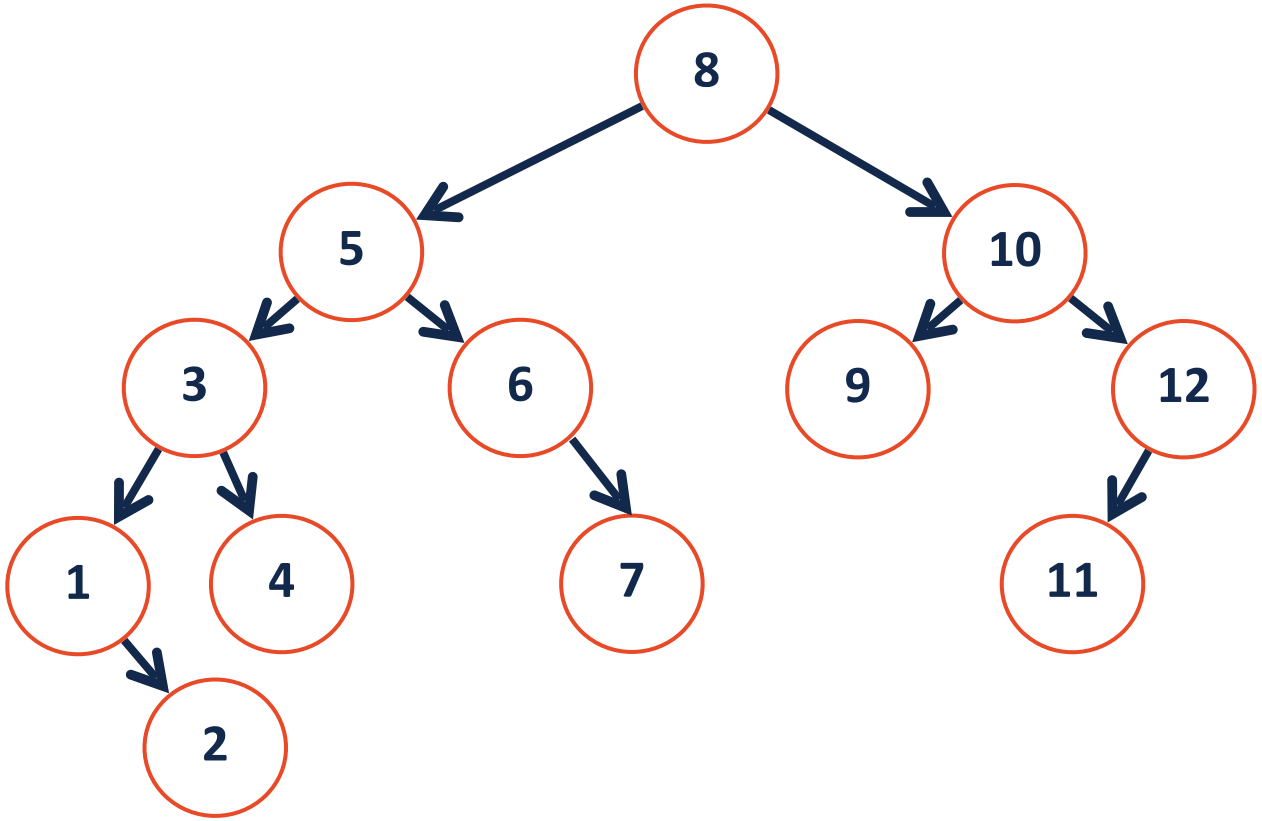
AVL Insertion Practice

`_insert(14)`



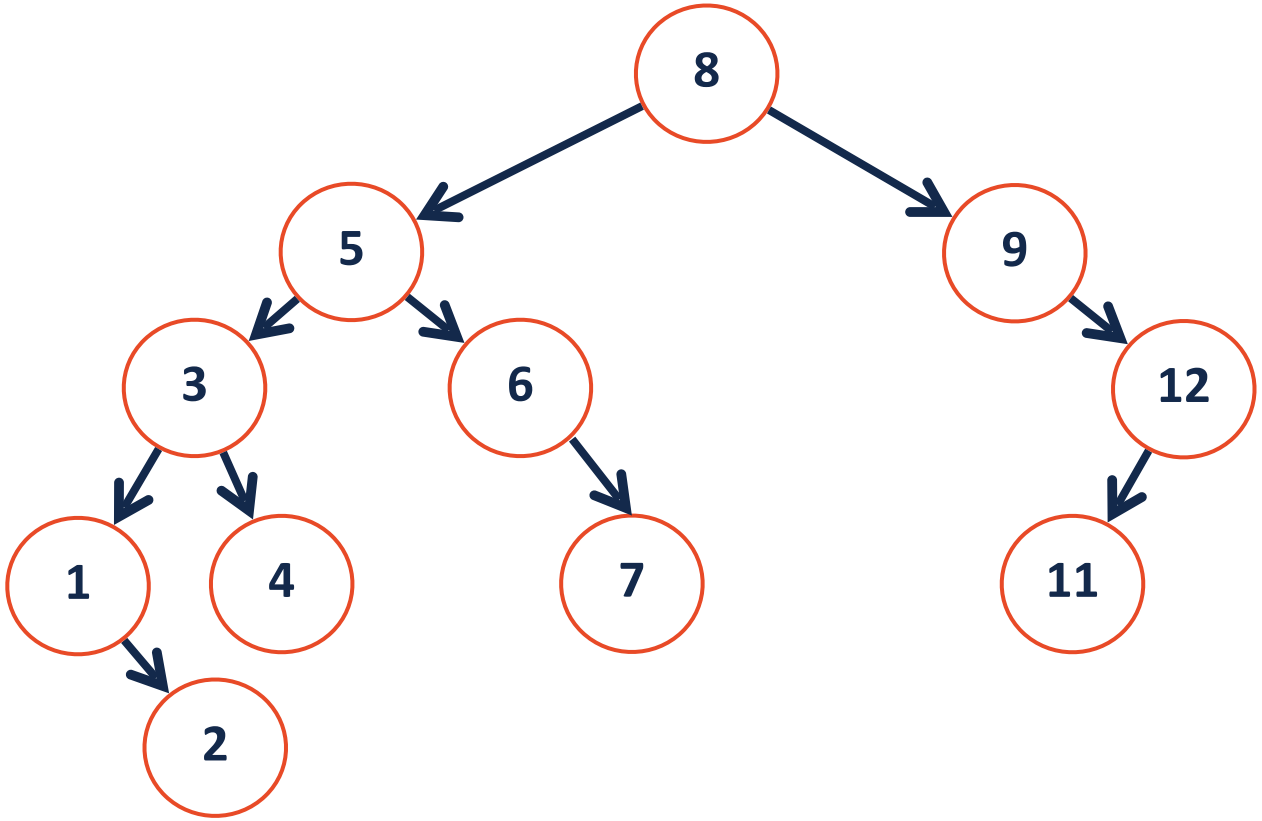
AVL Remove

`_remove(10)`



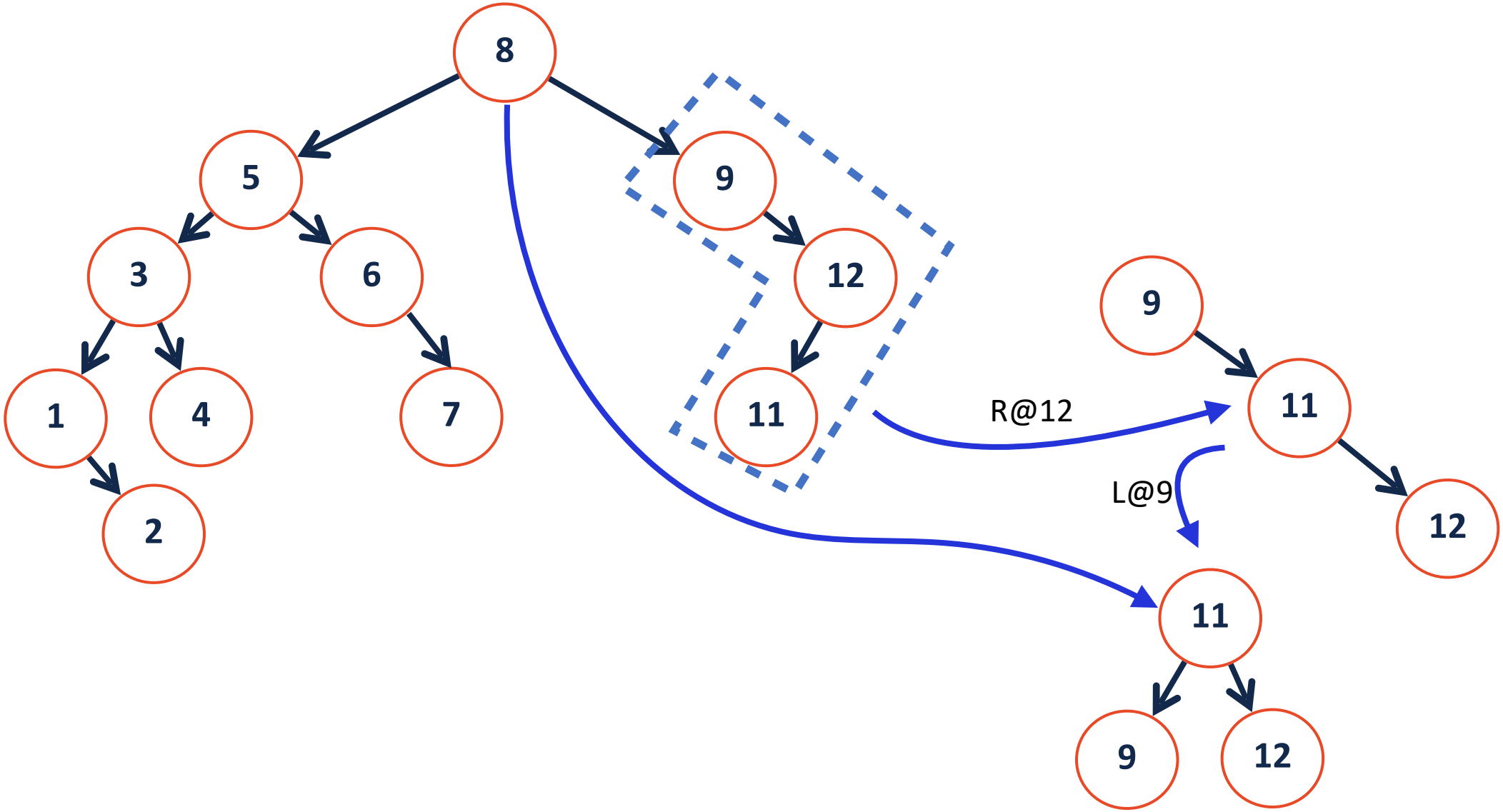
AVL Remove

`_remove(10)`



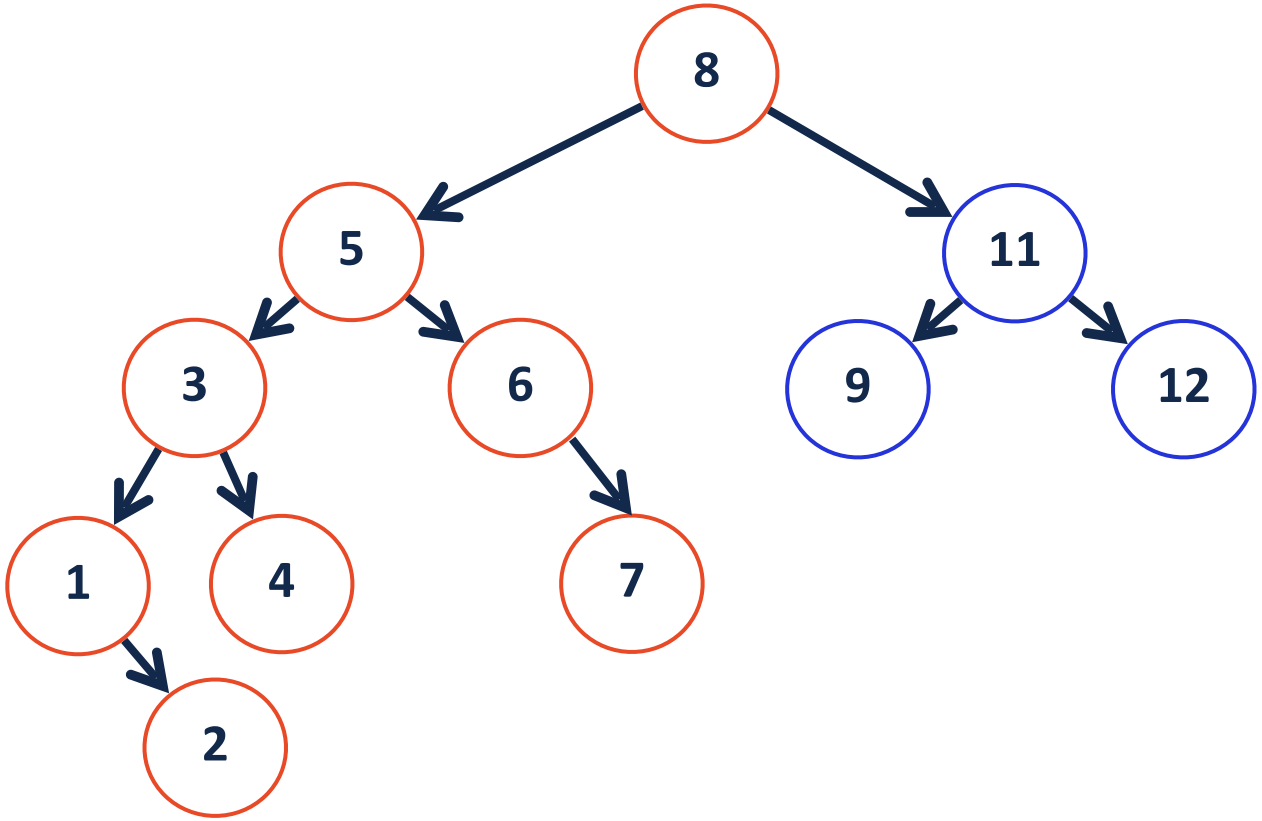
AVL Remove

`_remove(10)`



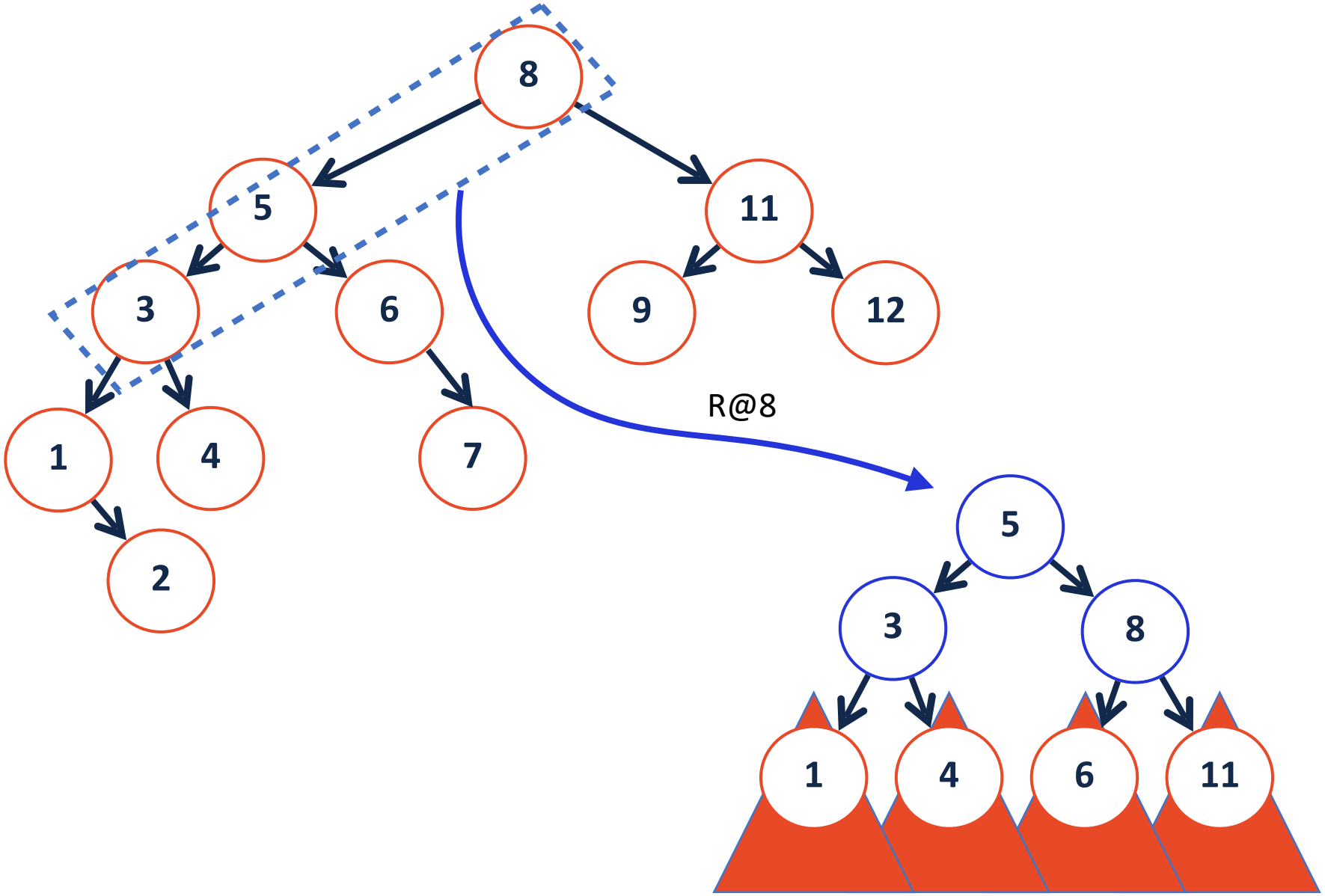
AVL Remove

`_remove(10)`



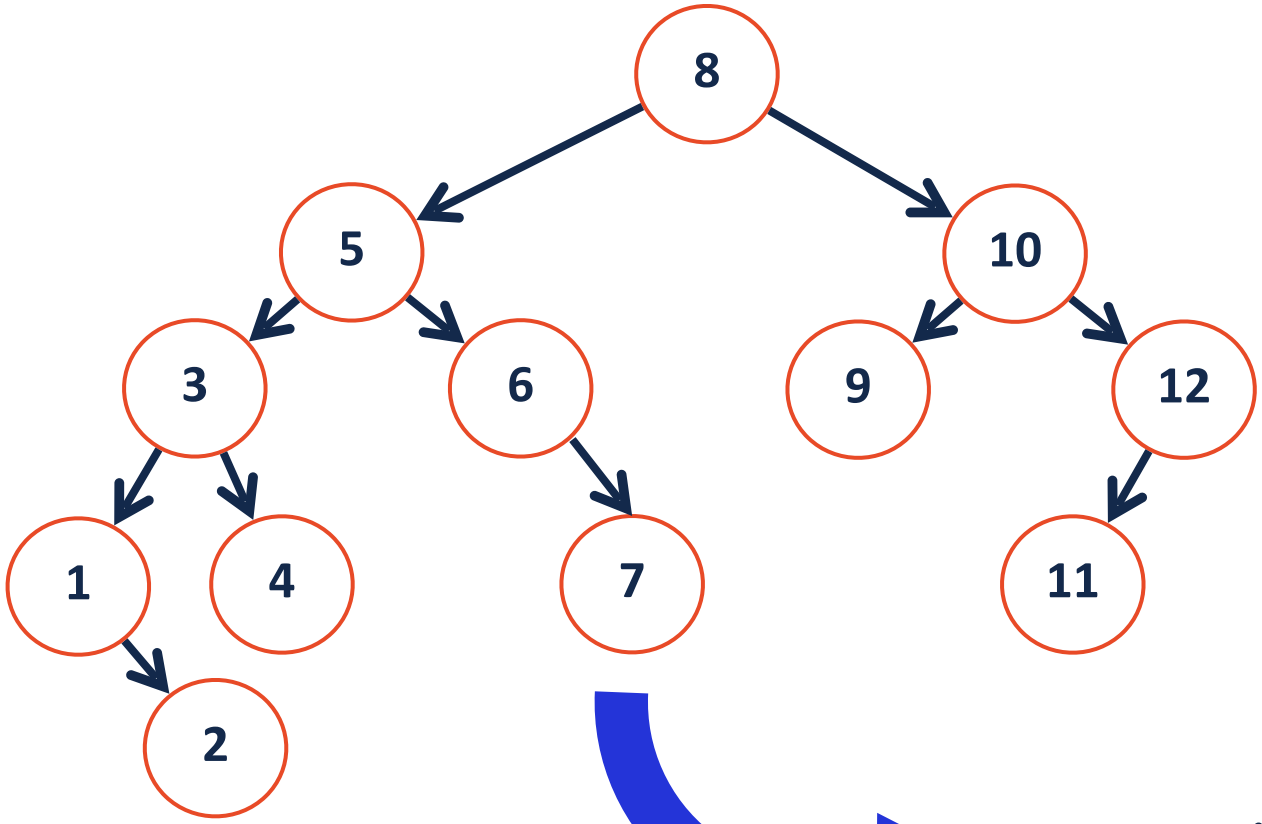
AVL Remove

`_remove(10)`



AVL Remove

`_remove(10)`



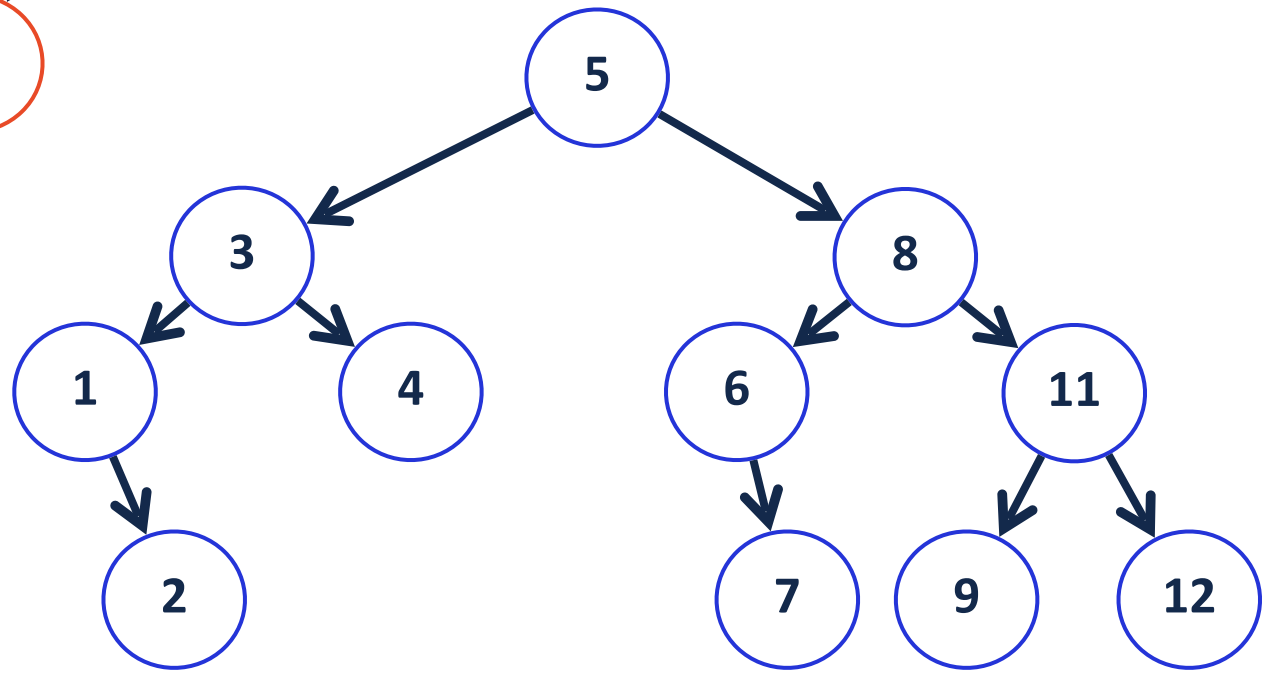
Remove (pseudo code):

1: Remove at proper place

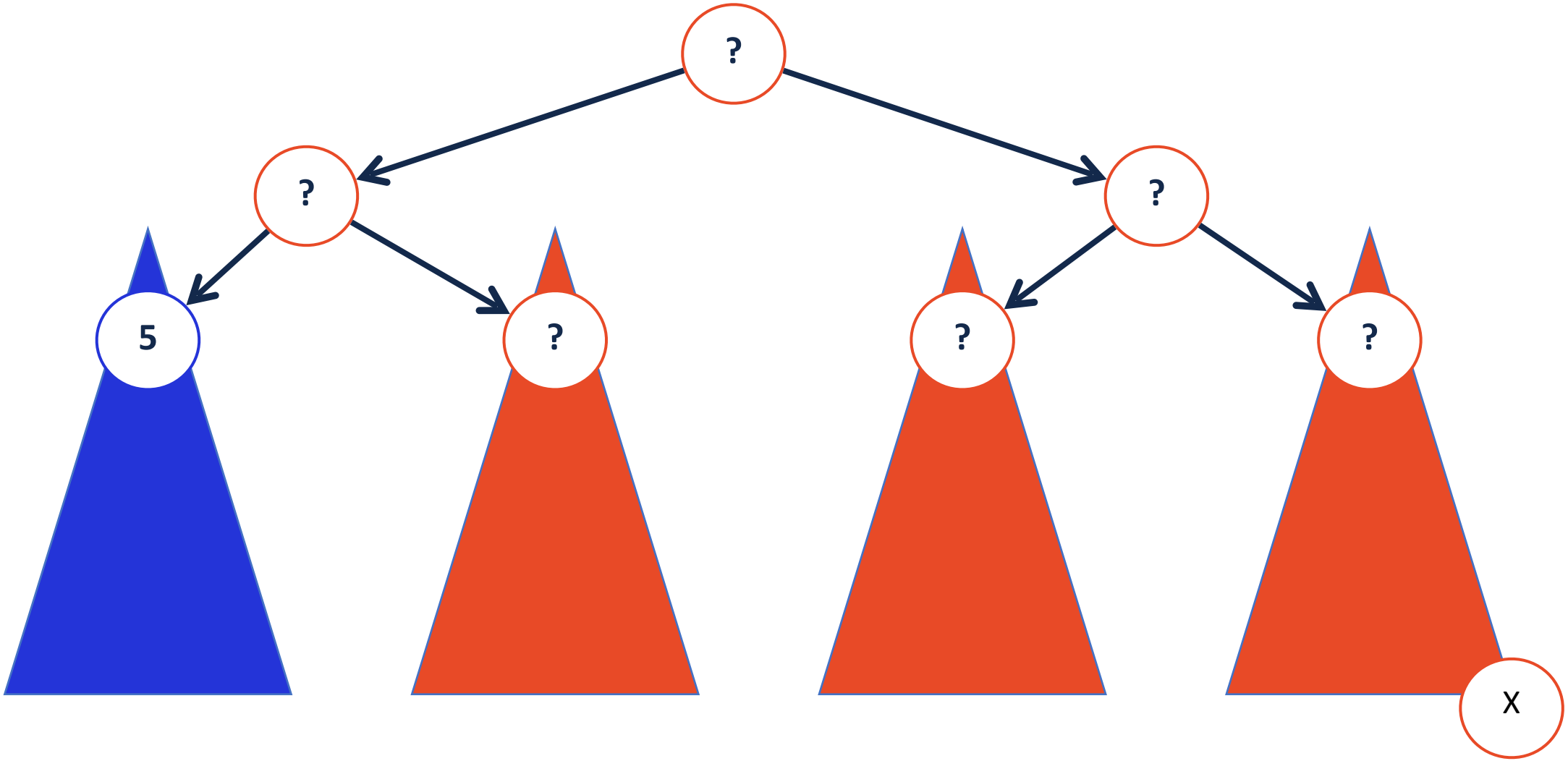
2: Check for imbalance

3: Rotate, if necessary

4: Update height



AVL Remove



AVL Remove

An AVL remove step can reduce a subtree height by at most:

But a rotation ***reduces*** the height of a subtree by one!

We might have to perform a rotation at every level of the tree!

AVL Tree Analysis

For an AVL tree of height h:

Find runs in: _____.

Insert runs in: _____.

Remove runs in: _____.

Claim: The height of the AVL tree with n nodes is: _____**.**