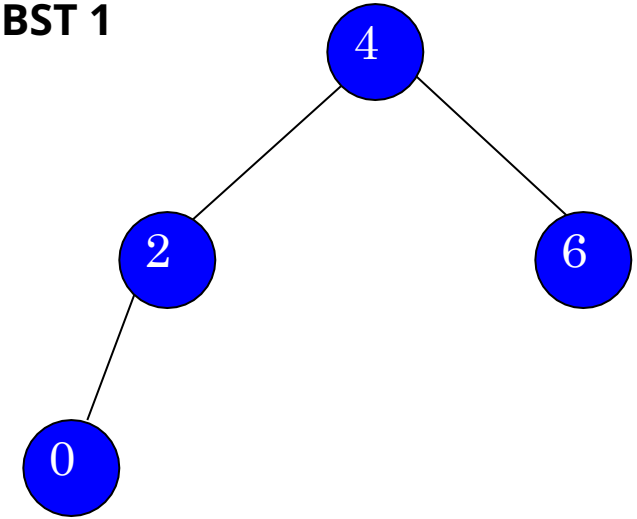
Data Structures
Balancing BSTs

COMP128 Data Structures

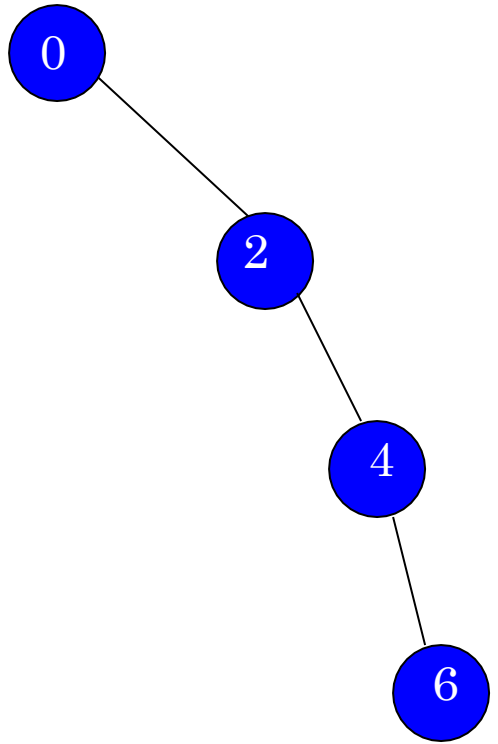
BST Performance

What is the BST shape that is least efficient to search?

BST 1



BST 2

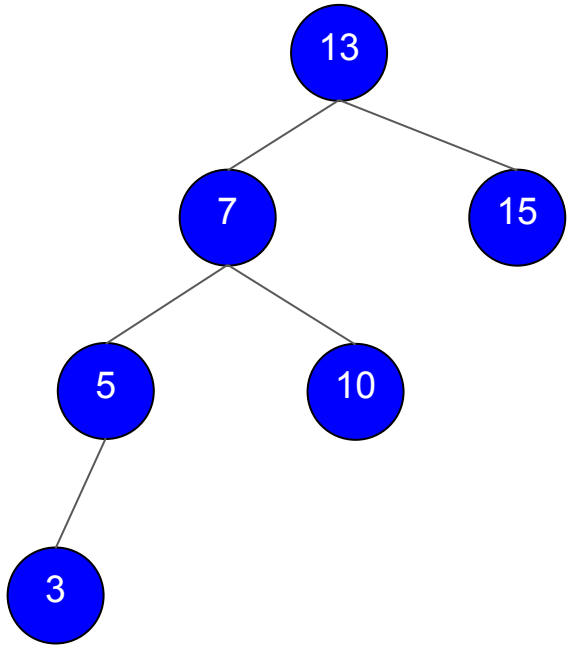


Balancing BSTs

- Our implementation does not ensure the BST stays balanced
- Other approaches do, such as AVL trees and red/black trees
- We will explore rotations – operations on binary search trees to assist in the process of keeping a tree balanced
- Rotations do not solve all problems created by unbalanced trees, but show the basic algorithmic processes that are used to manipulate trees

Right Rotation

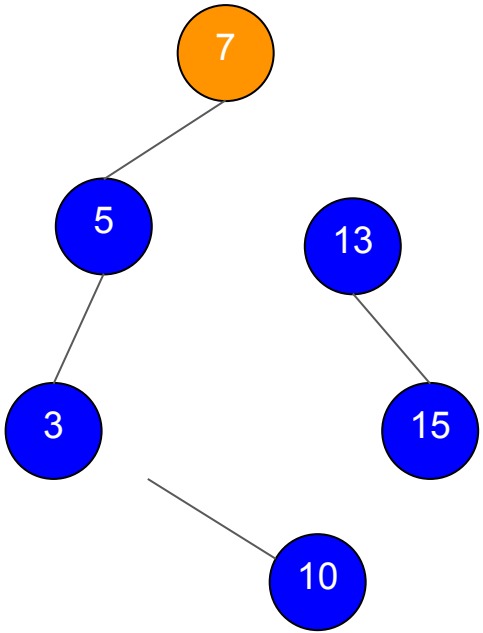
- A right rotation can be performed at any level of a tree, around the root of any subtree
- Corrects an imbalance caused by a long path in the left subtree of the left child of the root



Right Rotation

To correct the imbalance

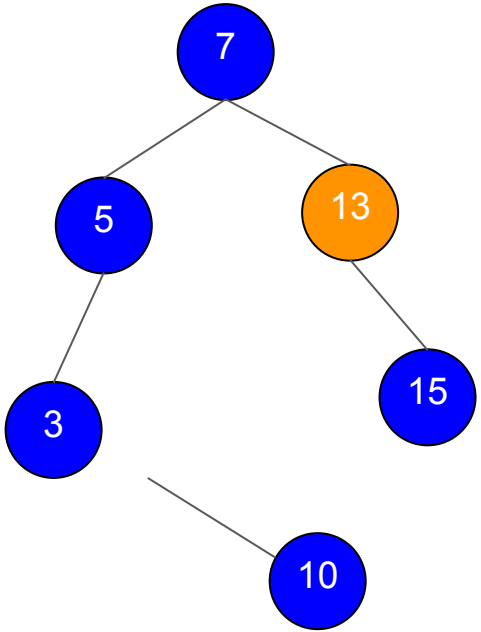
- make the **left child element of the root** the **new root** element



Right Rotation

To correct the imbalance

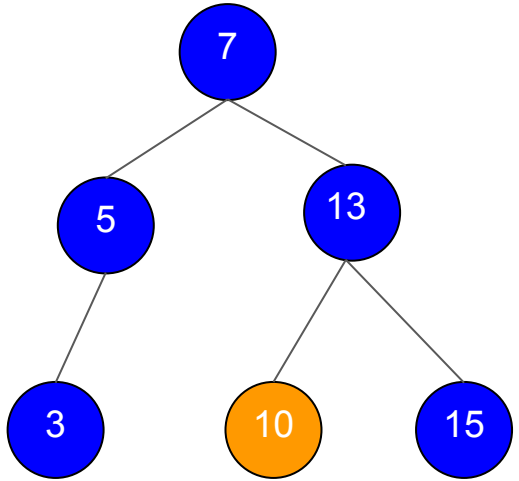
- make the left child element of the root the new root element
- make **the former root elemen**t the **right child** element of the new root



Right Rotation

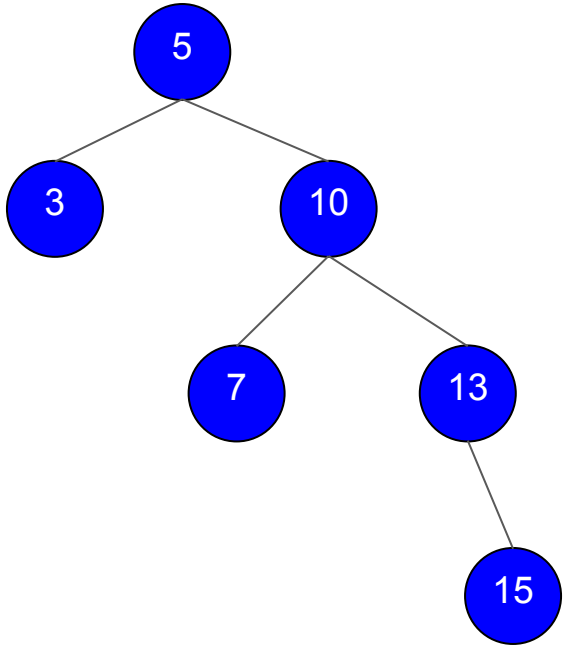
To correct the imbalance

- make the left child element of the root the new root element
- make the former root element the right child element of the new root
- make the **right child** of what was the left child of the former root the **new left child** of the former root



Left Rotation

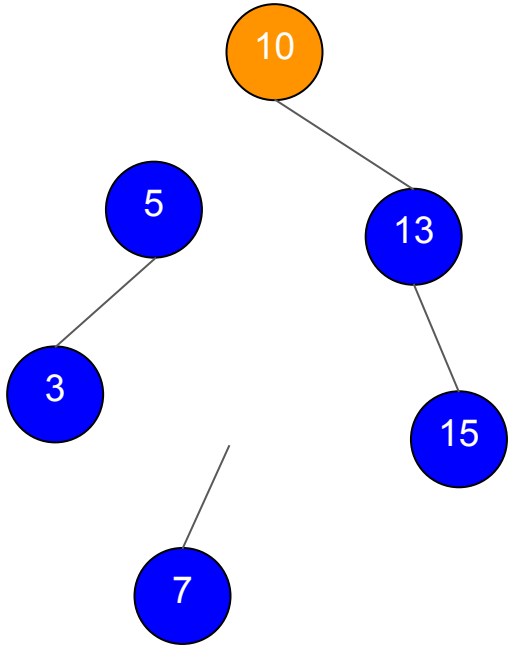
- A left rotation can be performed at any level of a tree, around the root of any subtree
- Corrects an imbalance caused by a long path in the right subtree of the left child of the root



Left Rotation

To correct the imbalance

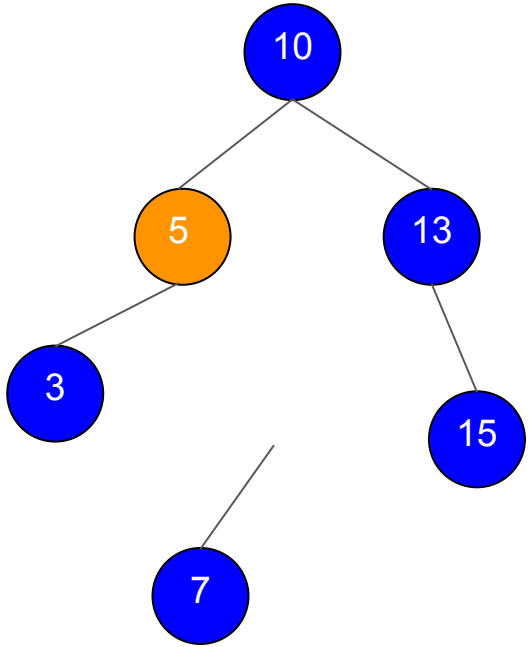
- make the **right child** element of the root **the new root** element



Left Rotation

To correct the imbalance

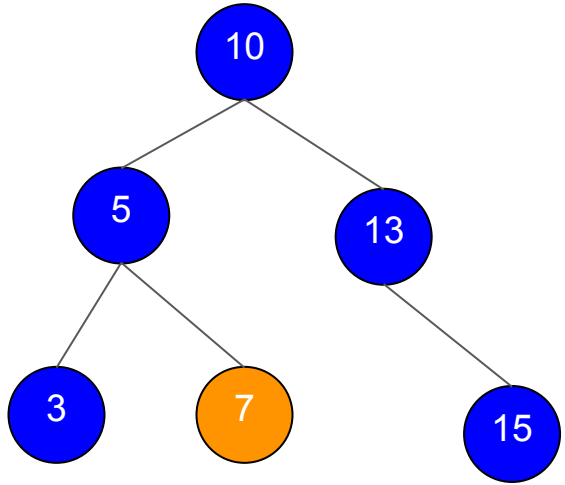
- make the right child element of the root the new root element
- make the **former root** element the **left child element** of the new root



Left Rotation

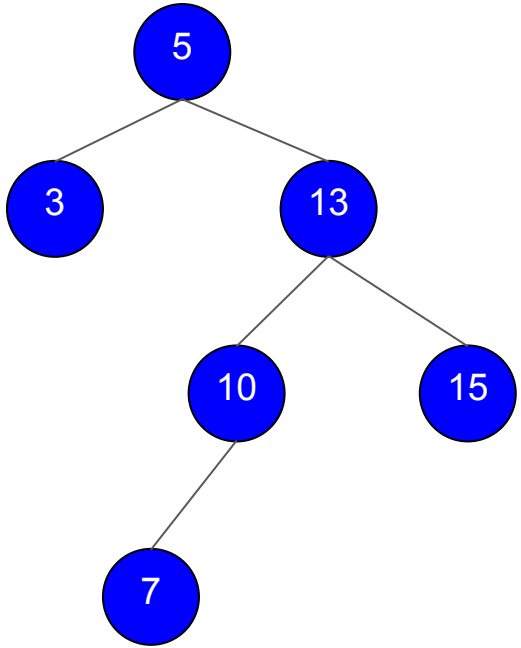
To correct the imbalance

- make the right child element of the root the new root element
- make the former root element the left child element of the new root
- make the **left child** of what was the right child of the former root the **new right child** of the former root



Rightleft Rotation

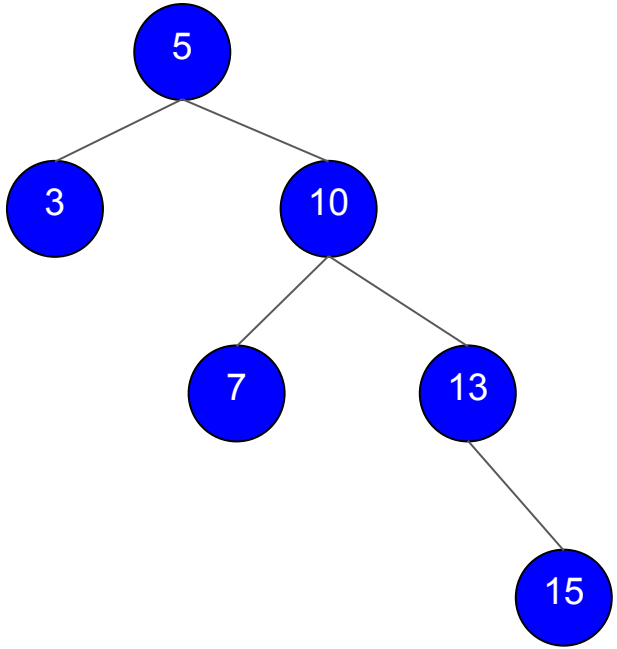
If the imbalance is caused by a long path **in the left subtree of the right child of the root** we can address it by performing a rightleft rotation:



Rightleft Rotation

If the imbalance is caused by a long path **in the left subtree of the right child of the root** we can address it by performing a rightleft rotation:

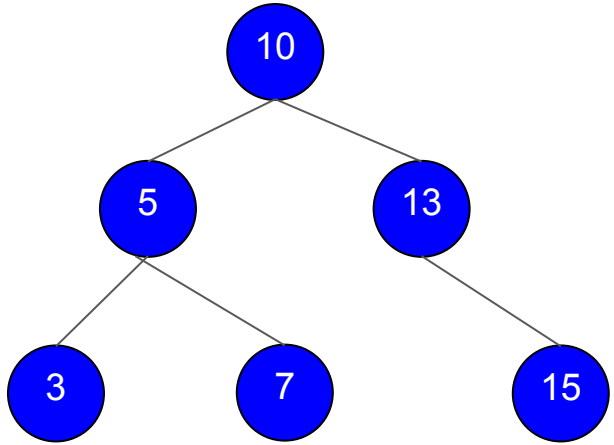
- performing **a right rotation around the heavy subtree**



Rightleft Rotation

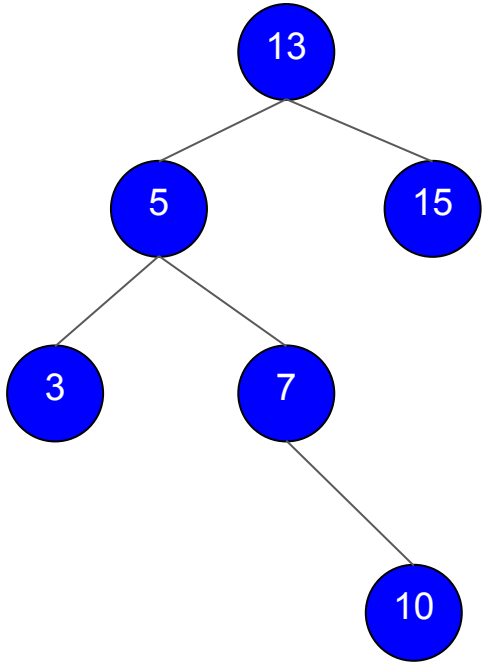
If the imbalance is caused by a long path **in the left subtree of the right child of the root** we can address it by performing a rightleft rotation:

- performing **a right rotation around the heavy subtree**
- and then **performing a left rotation around the root**



LeftRight Rotation

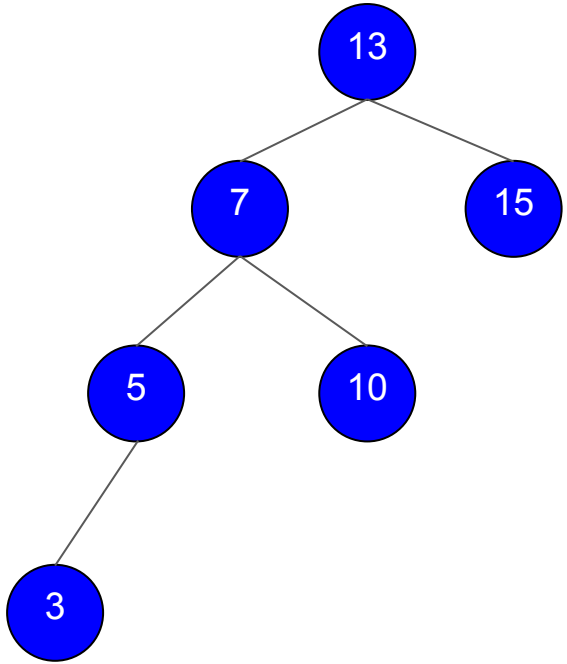
If the imbalance is caused by a long path **in the right subtree of the left child of the root** we can address it by performing a leftright rotation:



LeftRight Rotation

If the imbalance is caused by a long path **in the right subtree of the left child of the root** we can address it by performing a leftright rotation:

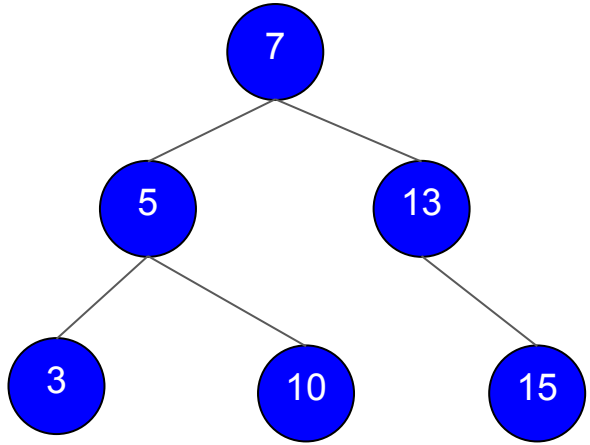
- performing **a left rotation around the heavy subtree**



LeftRight Rotation

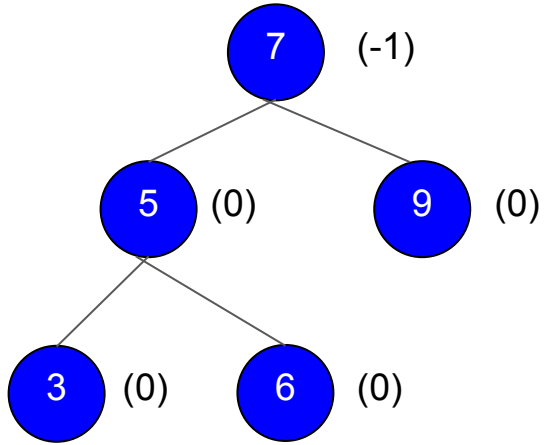
If the imbalance is caused by a long path **in the right subtree of the left child of the root** we can address it by performing a leftright rotation:

- performing **a left rotation around the heavy subtree**
- and then performing **a right rotation around the root**



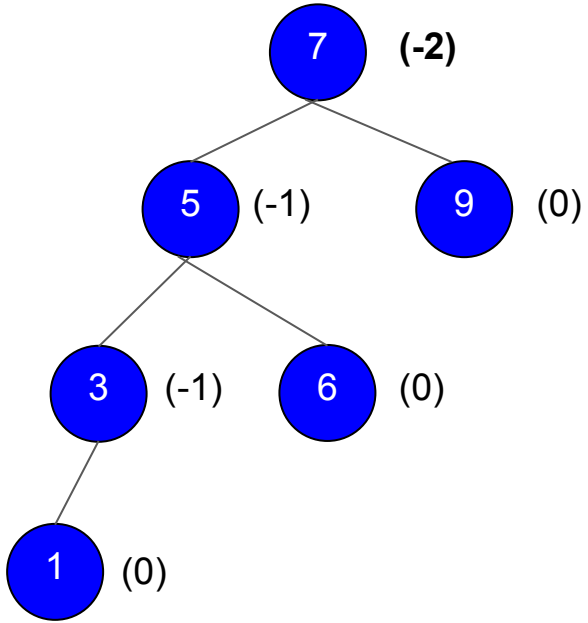
AVL Trees

- An AVL tree (named after the creators) ensures a BST stays balanced
- For each node in the tree, there is a numeric balance factor **– the difference between the heights of its subtrees**



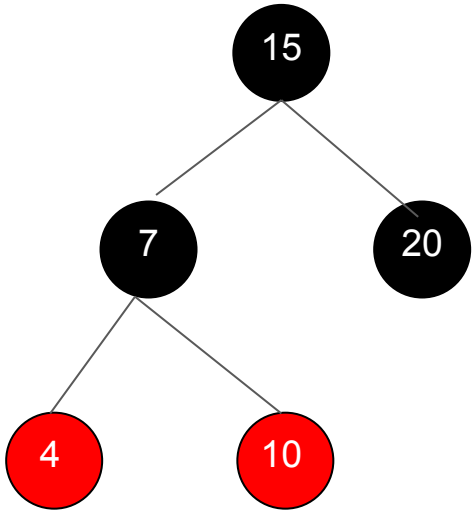
AVL Trees

- After each add or removal, **the balance factors are checked,** and rotations performed as needed



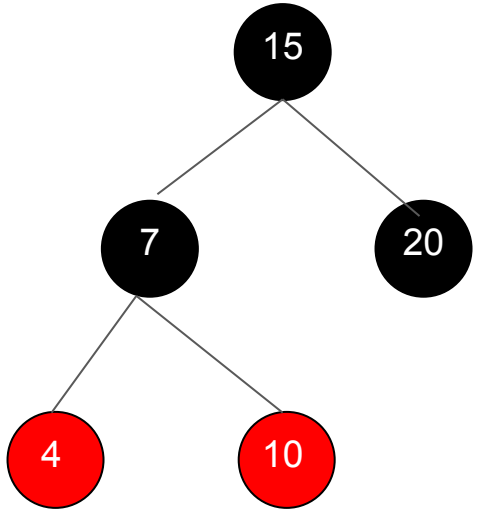
Red/Black Trees

- Another balanced BST approach is a red/black tree
- Each node has a color, usually implemented as a boolean value
- The following rules govern the color of a node:
  - the root is black
  - all children of red nodes are black
  - every path from the root to a leaf contains the same number of black nodes



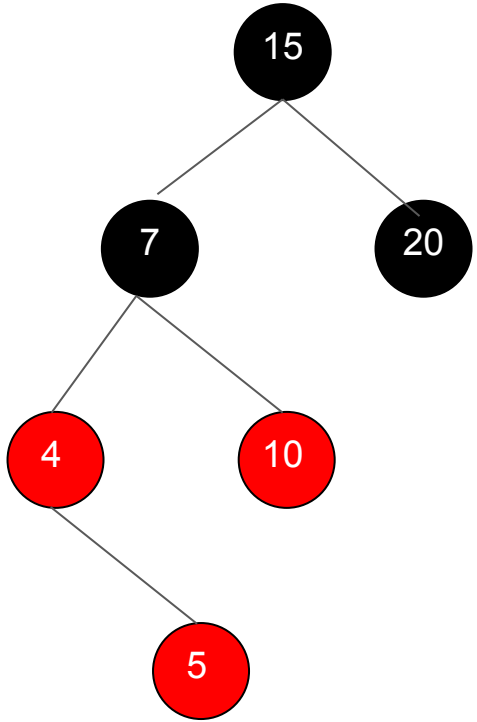
Red/Black Trees

- After each add or removal, **the color properties are checked,** recoloring and rotations performed as needed



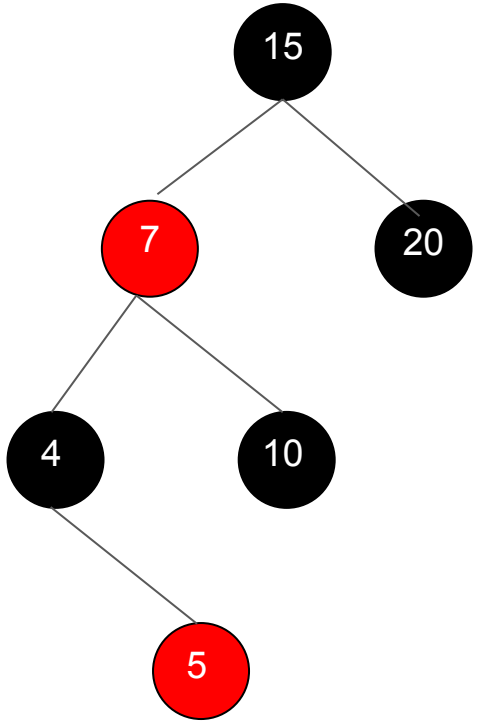
Red/Black Trees

- After each add or removal, **the color properties are checked,** recoloring and rotations performed as needed



Red/Black Trees

- After each add or removal, **the color properties are checked,** recoloring and rotations performed as needed



In-class Activity

Self-Balancing Trees Activity