Red Black Trees

Instructor: Krishna Venkatasubramanian

CSC 212

Based on slides from Prof. David Luebke at VirginiaTech

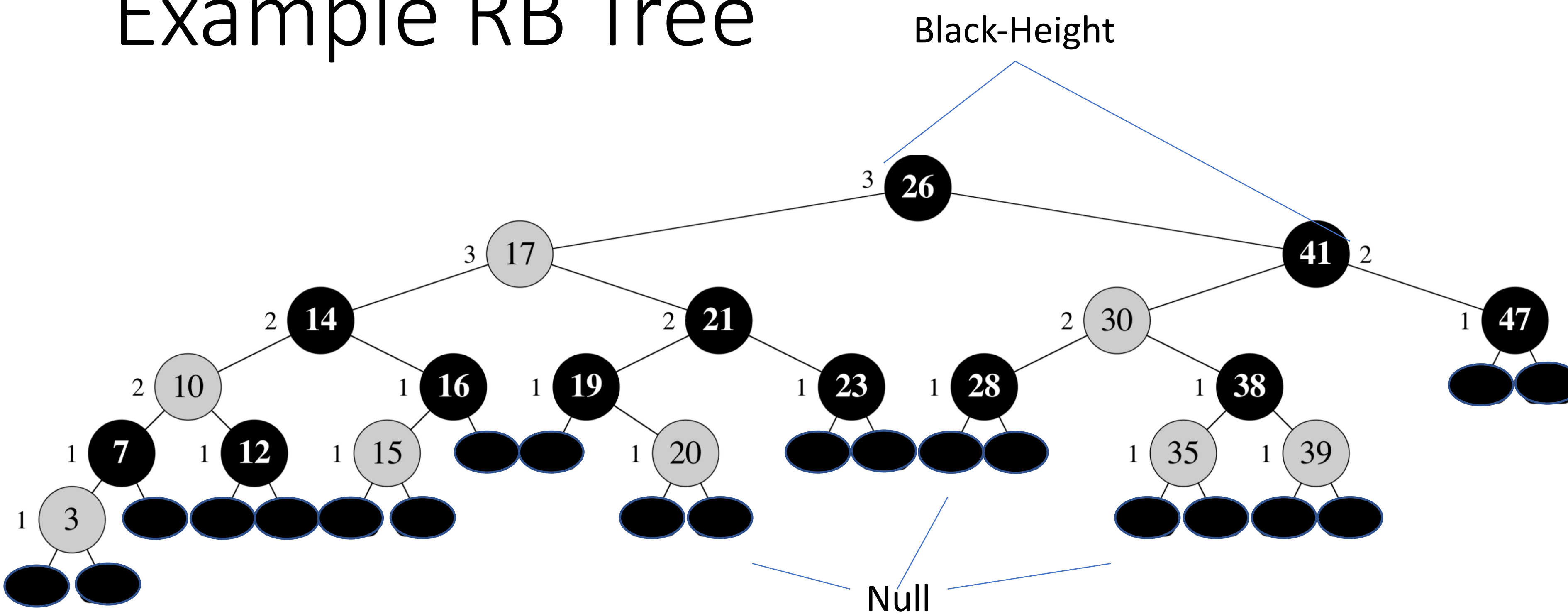
Red-Black Trees

- *Red-black (RB) trees*:
  - Binary search trees augmented with node color
  - Operations designed to guarantee that the height $h = O(\lg n)$
- We will do three things with RB Trees:
  - describe the properties of red-black trees
  - show that these guarantee $h = O(\lg n)$
    - Produce balanced trees!
    - Remember: BST works well when the trees are balanced
  - describe operations on RB trees

Red-Black Properties

- The *red-black properties*:
  1. Every node is either RED or BLACK
  2. Every NULL pointer at the base of the tree is BLACK
     - Note: even if null pointers are not show, always assume they are black for RB trees
  3. If a node is RED, both children are BLACK
     - Note: can’t have 2 consecutive reds on a path
  4. Every path from node to descendent leaf contains the same number of BLACK nodes
  5. The root is always BLACK

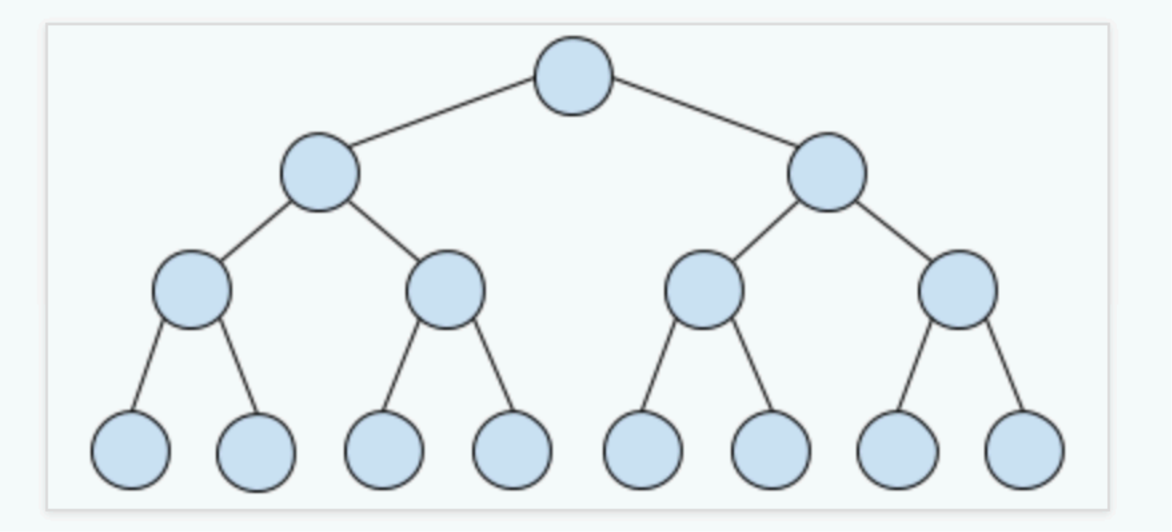
Example RB Tree



We call the number of black nodes on any simple path from (<u>but not including</u>) a node X down to a leaf as the **BLACK-Height** of the node → **bh(x)**

bh(26) = 3 (any path from 26 to Nil, excluding 26 has 3 black nodes in it’s path)

A Full and Complete Balanced Binary Tree



- A **full and complete** Binary Tree is
  - All non-leaf nodes have two children.
  - All leaf nodes are at the same depth.
- For such a tree
  - **n = $2^{(h+1)}$ - 1**
    - where n = number of nodes in a tree and h is the height of tree)
    - Above: h = 3, therefore n = 2^4-1 = 15
  - **h+1 = lg(n+1)**

RB Trees Height

- Rule 4 for RB Trees:
  - Every path from node to descendent leaf contains the same number of black nodes

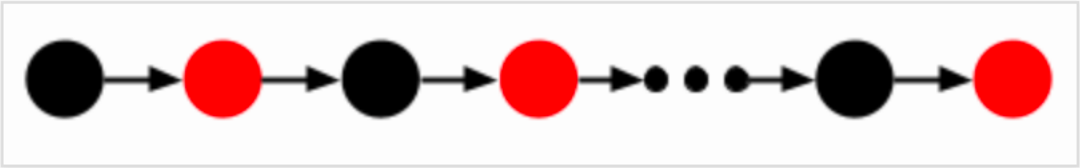
- Further: A black node can have two black children
  - As there are no rules restricting this

- The maximum number of **black nodes** in <u>any</u> <u>root-to-null (both inclusive) path</u> is **lg(*n*+1)** (i.e., h+1 from previous slide)

RB Trees Height (2)

- Now in a path from root to Null (both inclusive) can have both red and black nodes
- Rule 3 says:
  - If a node is red, both children are black

- Therefore, in a path from root to Null (both inclusive), if you have red and black nodes then **maximum number of red nodes appear when red and black nodes alternate**



- Therefore the **max-height of longest path of the RB Tree** is
  - Max Black Nodes + Max Red Nodes
  - **lg(n+1) + lg(n+1) = 2lg(n+1)**

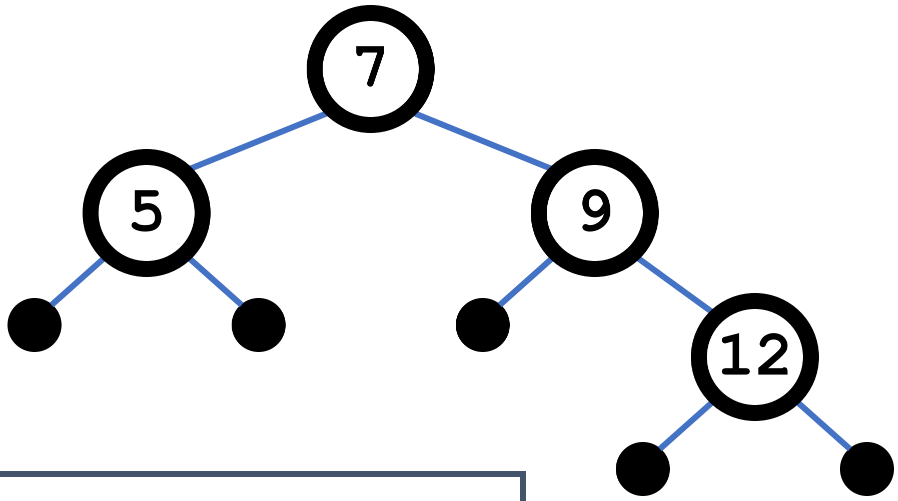
- **Red Black Trees have a height that is always O(lg n)**

RB Trees: Worst-Case Time

- Since a red-black tree has O(lg *n*) height
- These operations take O(lg *n*) time:
  - Minimum(), Maximum()
  - Successor(), Predecessor()
  - Search()
- Insert() and Delete():
  - Will also take O(lg *n*) time
  - But will need special care since they modify tree

Red-Black Trees: An Example

- *Color this tree:*

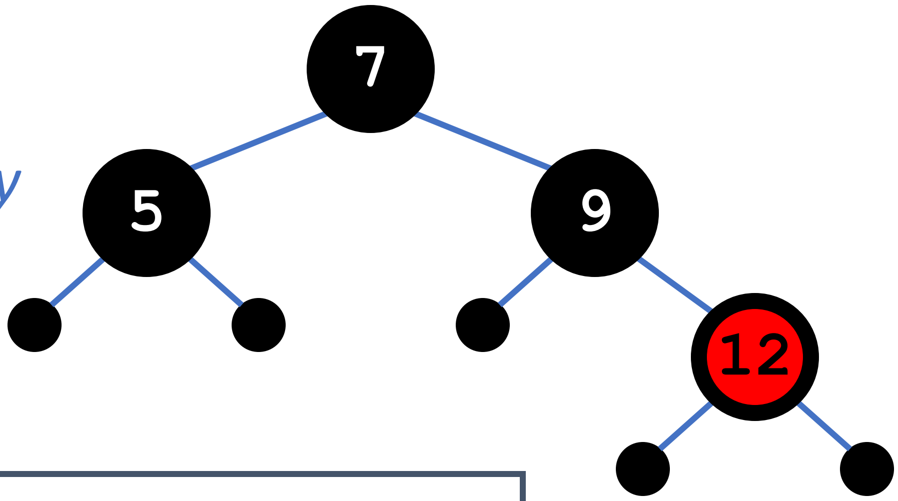


Red-black properties:

1. Every node is either red or black
2. Every leaf (NULL pointer) is black
3. If a node is red, both children are black
4. Every path from node to descendent leaf contains the same number of black nodes
5. The root is always black

Red-Black Trees: An Example

- *Color this tree:*
  - *Follows the rule every path has same black height* (#4)

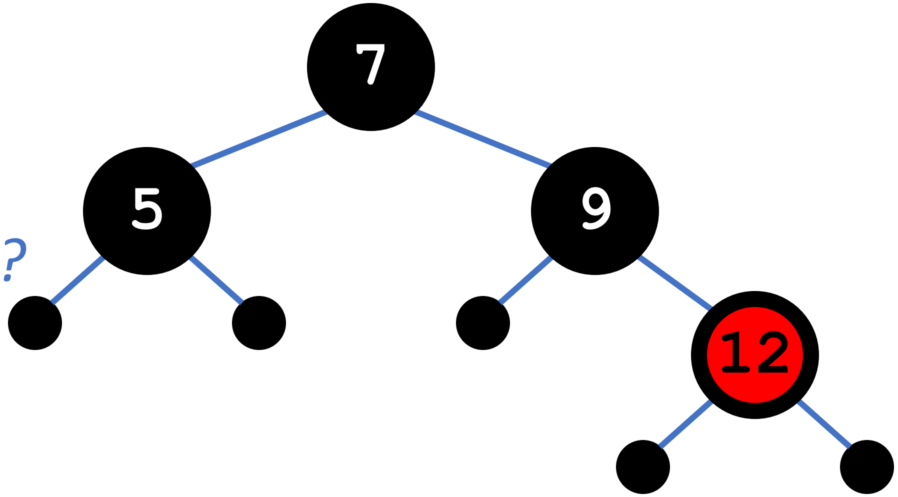


Red-black properties:

1. Every node is either red or black
2. Every leaf (NULL pointer) is black
3. If a node is red, both children are black
4. Every path from node to descendent leaf contains the same number of black nodes
5. The root is always black

Red-Black Trees: The Problem With Insertion

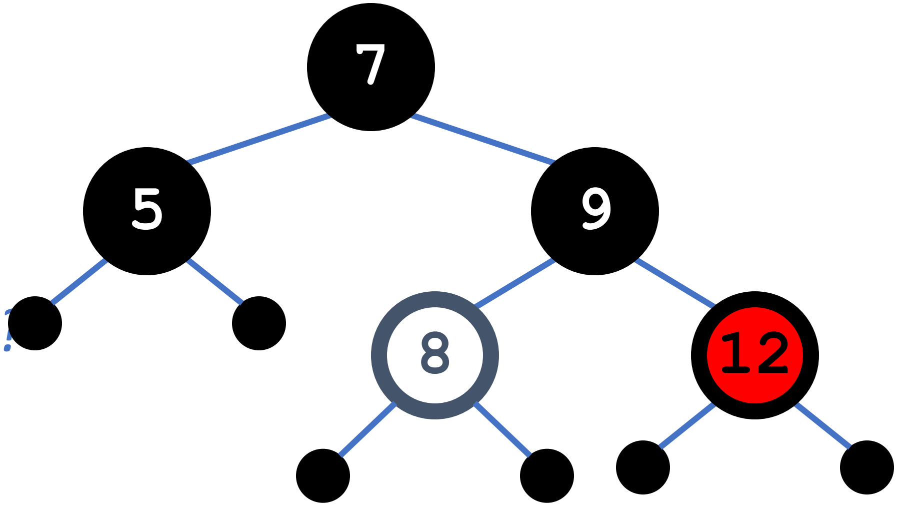
- Insert 8
  - *Where does it go?*
  - *What color should it be?*



1. Every node is either red or black
2. Every leaf (NULL pointer) is black
3. If a node is red, both children are black
4. Every path from node to descendent leaf contains the same number of black nodes
5. The root is always black

Red-Black Trees: The Problem With Insertion

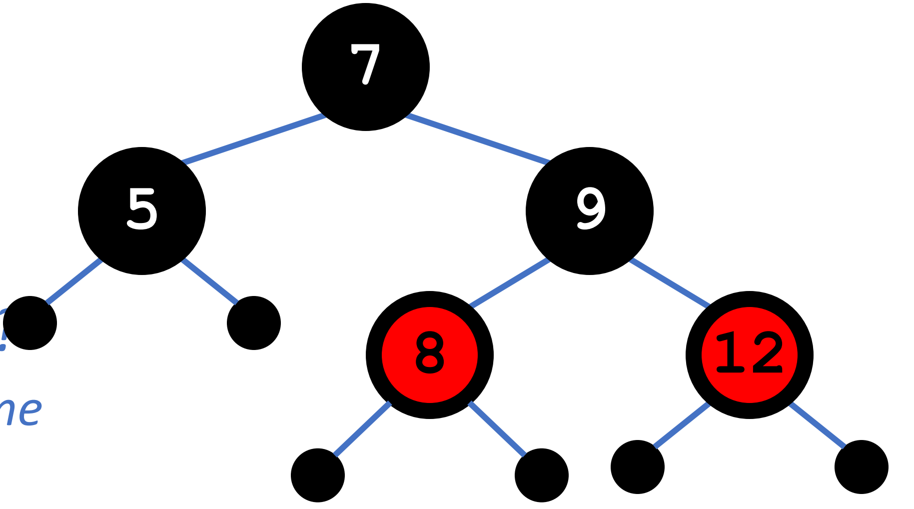
- Insert 8
  - *Where does it go?*
    - *Follow BST insert*
  - *What color should it be?*



1. Every node is either red or black
2. Every leaf (NULL pointer) is black
3. If a node is red, both children are black
4. Every path from node to descendent leaf contains the same number of black nodes
5. The root is always black

Red-Black Trees: The Problem With Insertion

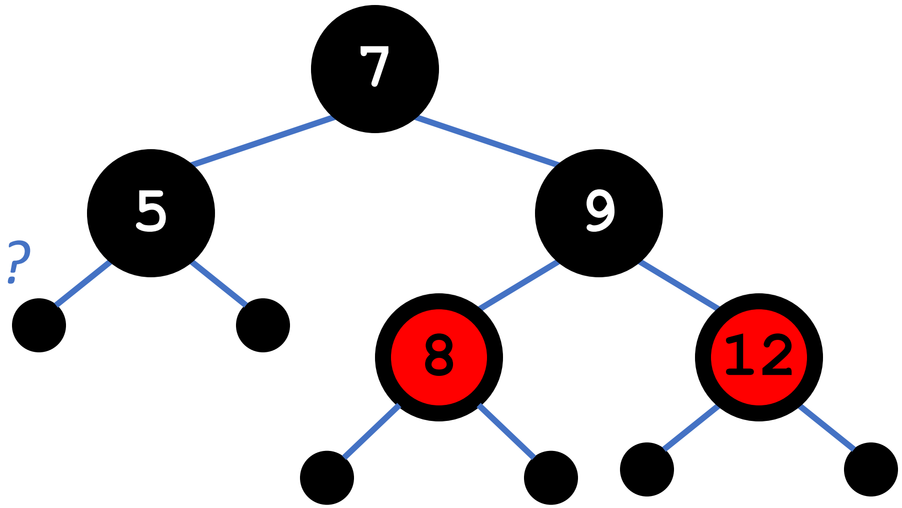
- Insert 8
  - *Where does it go?*
    - *Follow BST insert*
  - *What color should it be?*
    - *RED: every path has same black height* (#4)



1. Every node is either red or black
2. Every leaf (NULL pointer) is black
3. If a node is red, both children are black
4. Every path from node to descendent leaf contains the same number of black nodes
5. The root is always black

Red-Black Trees: The Problem With Insertion

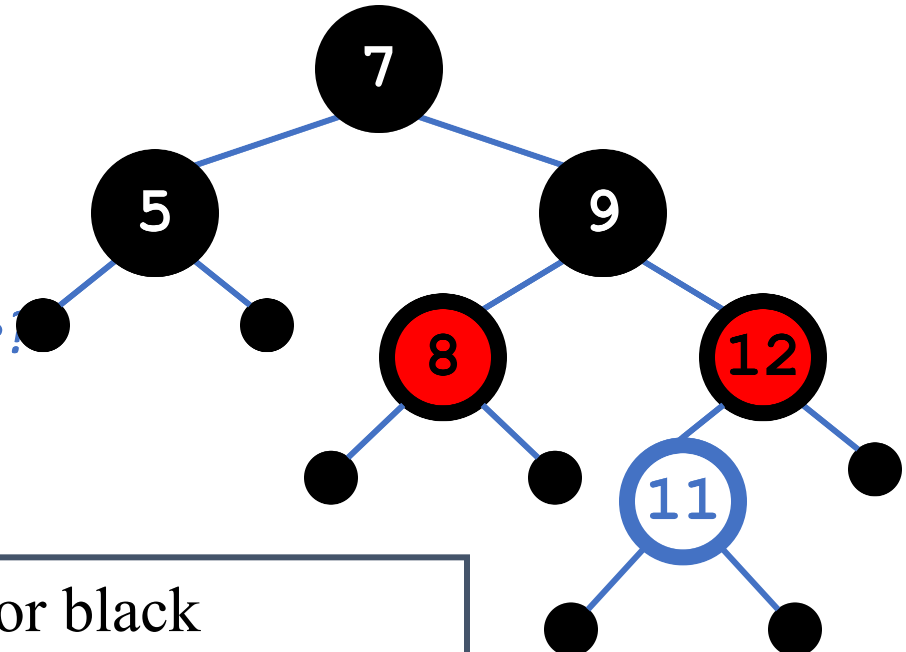
- Insert 11
  - *Where does it go?*
  - *What color should it be?*



1. Every node is either red or black
2. Every leaf (NULL pointer) is black
3. If a node is red, both children are black
4. Every path from node to descendent leaf contains the same number of black nodes
5. The root is always black

Red-Black Trees: The Problem With Insertion

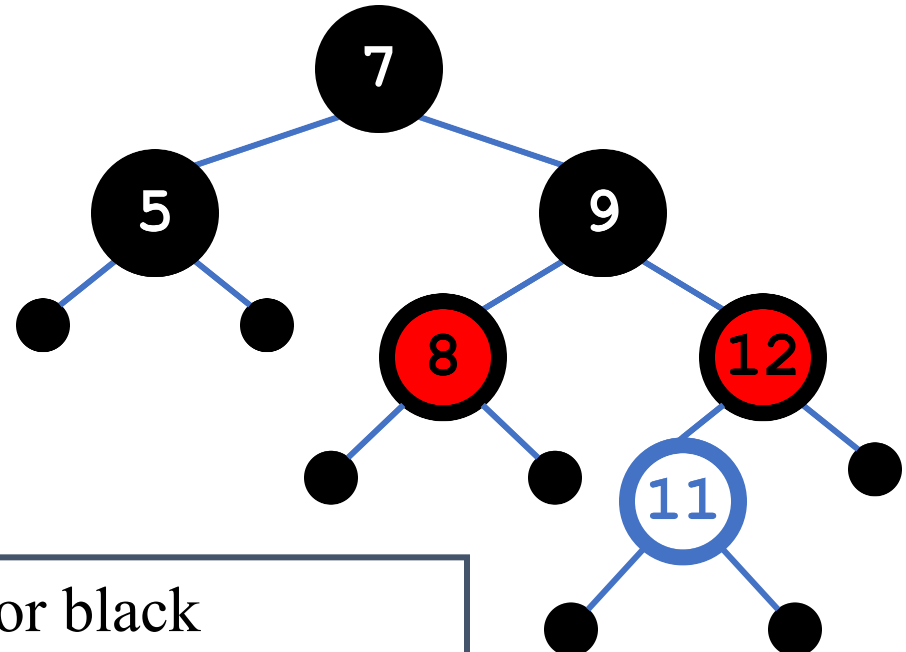
- Insert 11
  - *Where does it go?*
    - *Follow BST*
  - *What color should it be?*



1. Every node is either red or black
2. Every leaf (NULL pointer) is black
3. If a node is red, both children are black
4. Every path from node to descendent leaf contains the same number of black nodes
5. The root is always black

Red-Black Trees:
The Problem With Insertion

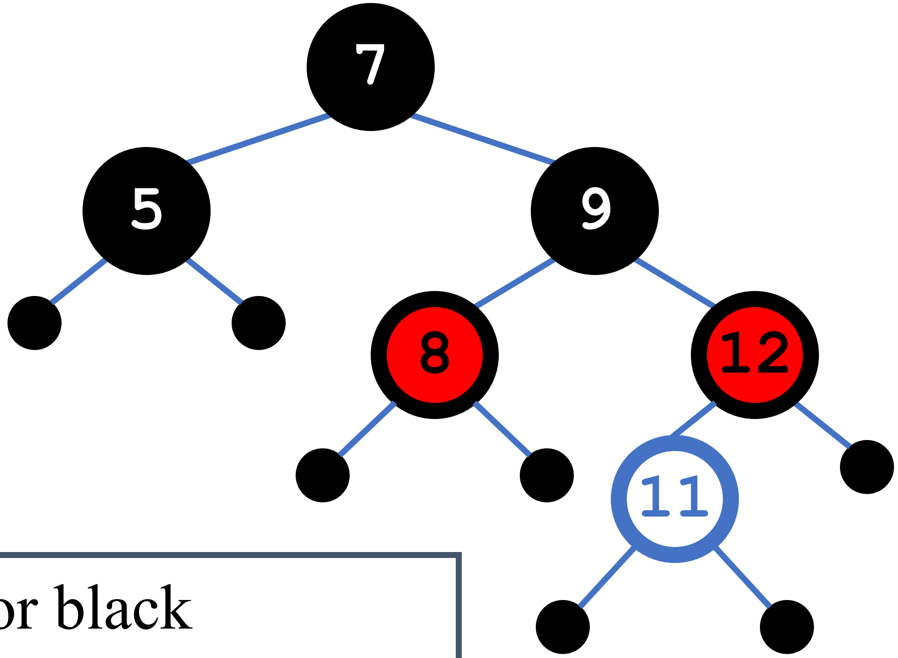
- Insert 11
  - *Where does it go?*
    - *Follow BST insert*
  - *What color?*
    - Can't be red! (#3)



1. Every node is either red or black
2. Every leaf (NULL pointer) is black
3. If a node is red, both children are black
4. Every path from node to descendent leaf contains the same number of black nodes
5. The root is always black

Red-Black Trees: The Problem With Insertion

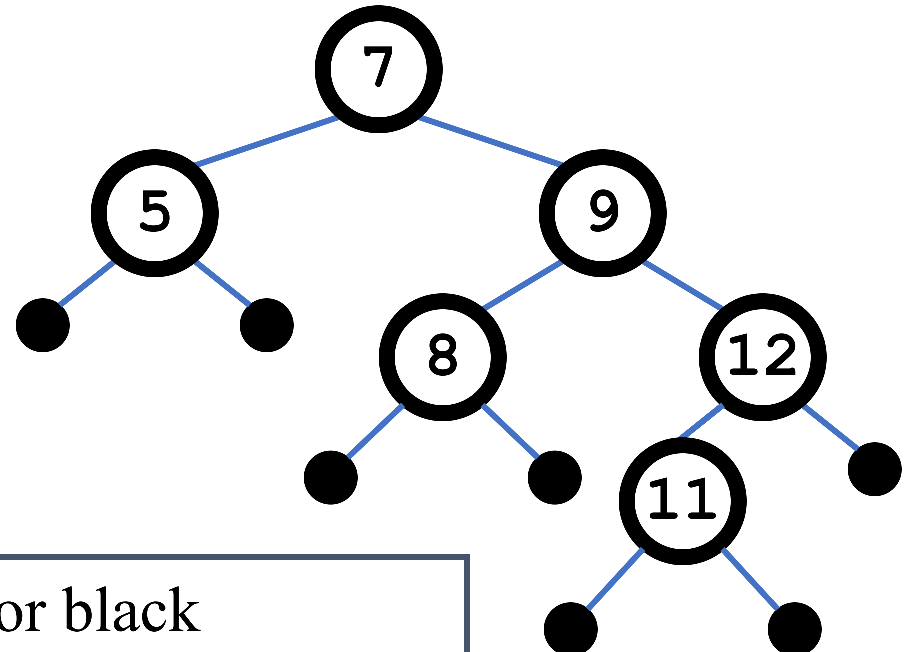
- Insert 11
  - *Where does it go?*
    - *Follow BST insert*
  - *What color?*
    - Can't be red! (#3)
    - Can't be black! (#4)



1. Every node is either red or black
2. Every leaf (NULL pointer) is black
3. If a node is red, both children are black
4. Every path from node to descendent leaf contains the same number of black nodes
5. The root is always black

Red-Black Trees: The Problem With Insertion

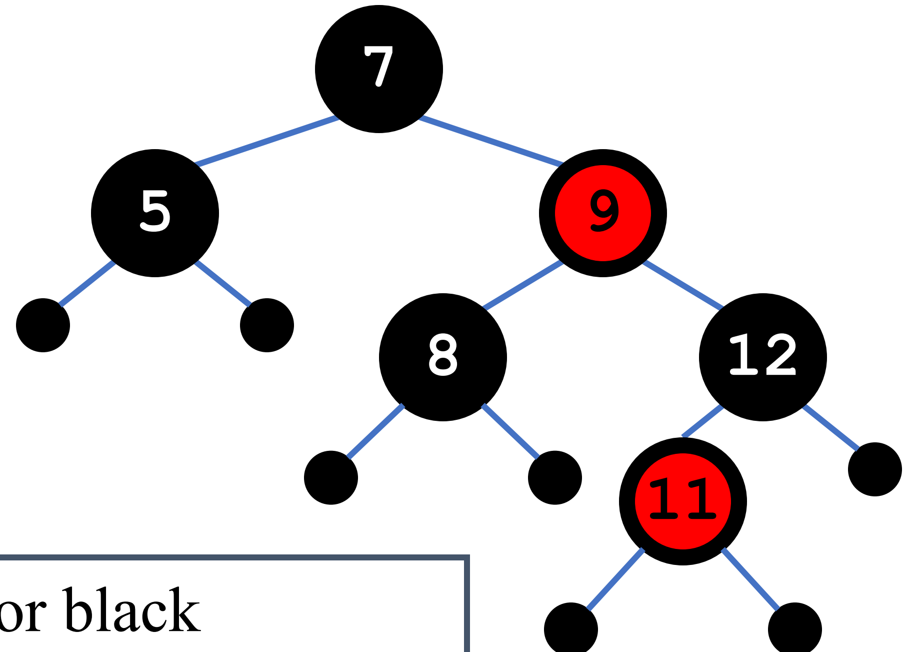
- Insert 11
  - *Where does it go?*
    - *Follow BST insert*
  - *What color?*
    - **Solution: recolor the tree**



1. Every node is either red or black
2. Every leaf (NULL pointer) is black
3. If a node is red, both children are black
4. Every path from node to descendent leaf contains the same number of black nodes
5. The root is always black

Red-Black Trees:
The Problem With Insertion

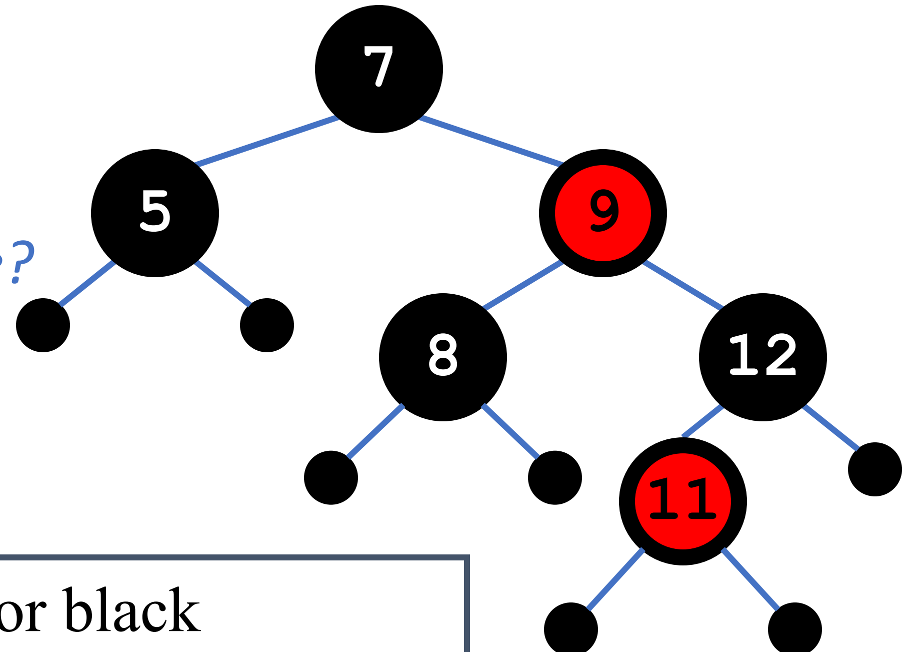
- Insert 11
  - *Where does it go?*
  - *What color?*
    - **Solution: recolor the tree**



1. Every node is either red or black
2. Every leaf (NULL pointer) is black
3. If a node is red, both children are black
4. Every path from node to descendent leaf contains the same number of black nodes
5. The root is always black

Red-Black Trees: The Problem With Insertion

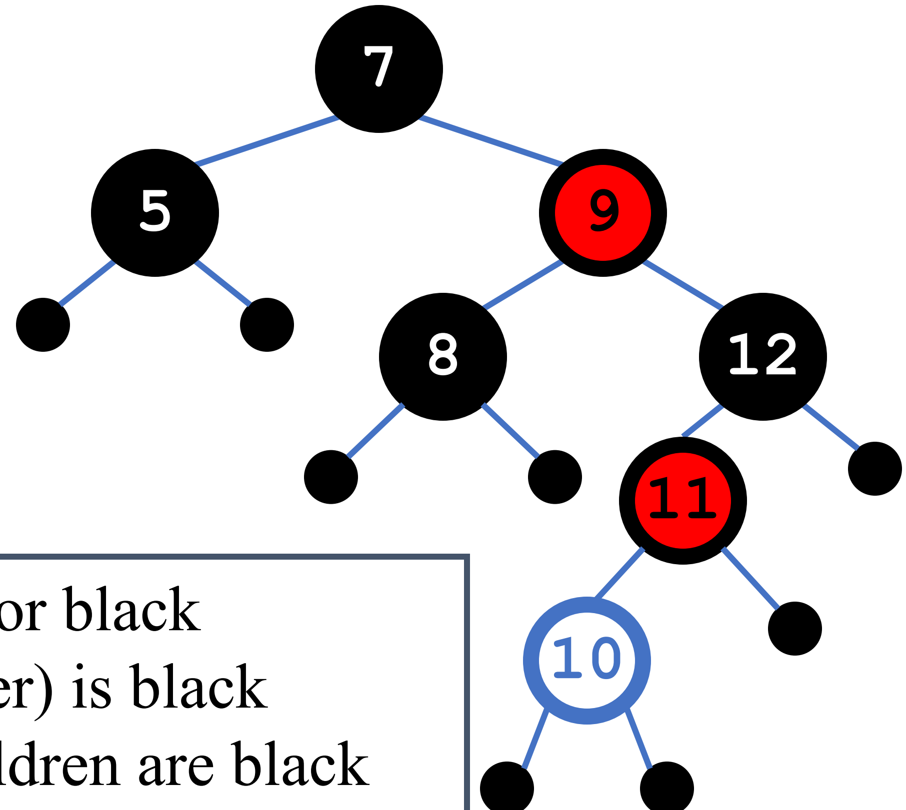
- Insert 10
  - *Where does it go?*
  - *What color should it be?*



1. Every node is either red or black
2. Every leaf (NULL pointer) is black
3. If a node is red, both children are black
4. Every path from node to descendent leaf contains the same number of black nodes
5. The root is always black

Red-Black Trees: The Problem With Insertion

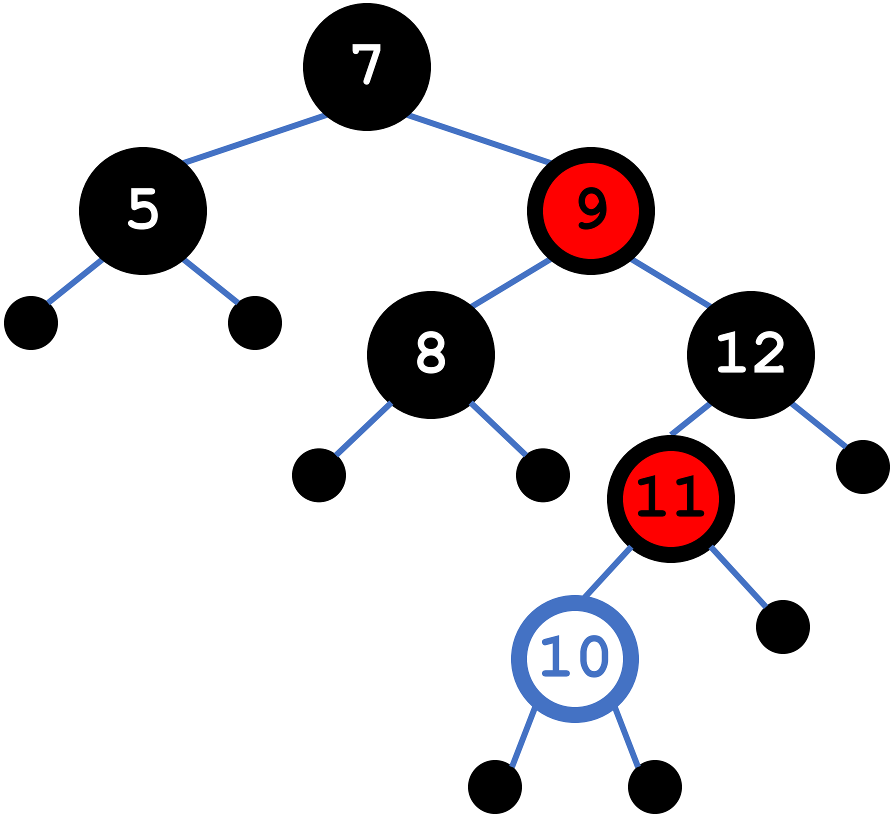
- Insert 10
  - *Where does it go?*
    - *Follow BST insert*
  - *What color?*



1. Every node is either red or black
2. Every leaf (NULL pointer) is black
3. If a node is red, both children are black
4. Every path from node to descendent leaf contains the same number of black nodes
5. The root is always black

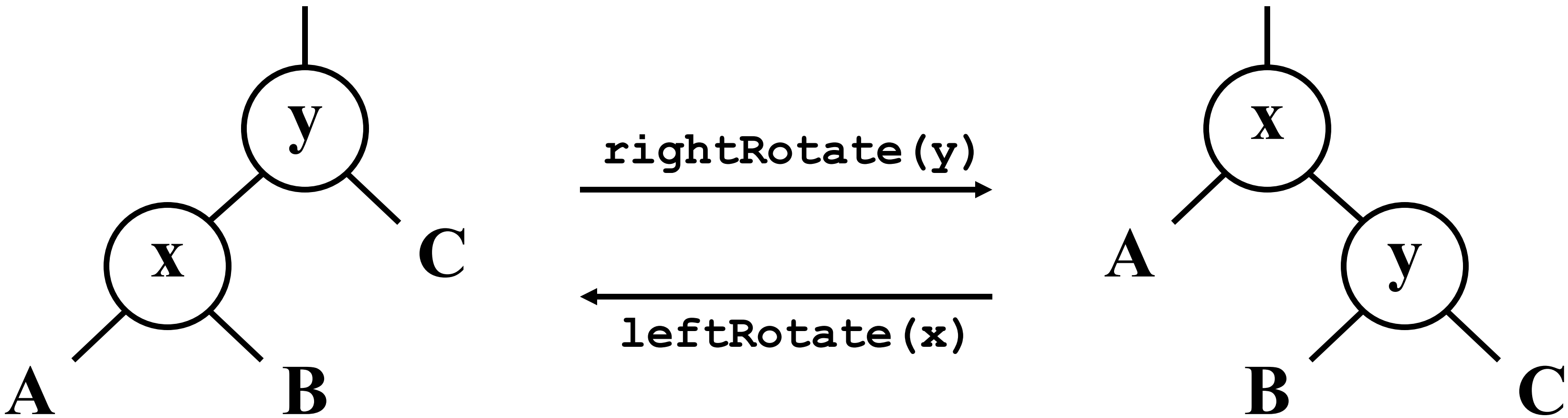
Red-Black Trees: The Problem With Insertion

- Insert 10
  - *Where does it go?*
  - *What color?*
    - A: no color possible
      Tree is too imbalanced
    - Must change tree structure
      to allow recoloring
  - Goal: restructure tree in
    O(lg *n*) time



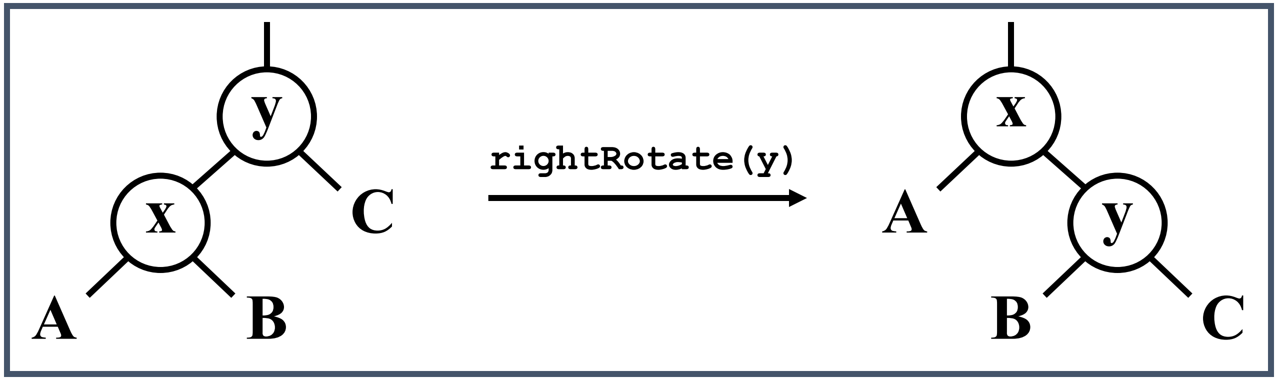
RB Trees: Rotation

- Our basic operation for changing tree structure is called ***rotation***:



- So what’s going on here?

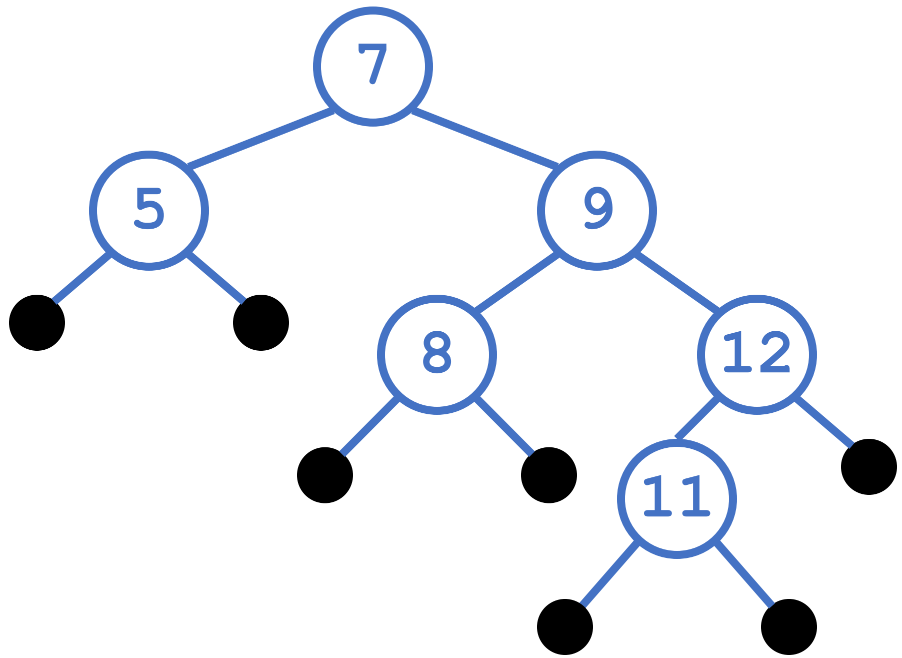
RB Trees: Rotation



- Answer: A lot of Tree Node Link manipulation
  - *x* keeps its left child
  - *y* keeps its right child
  - *x*'s right child becomes *y*'s left child
  - *x*'s and *y*'s parents change
- *What is the running time?* *O(1)*

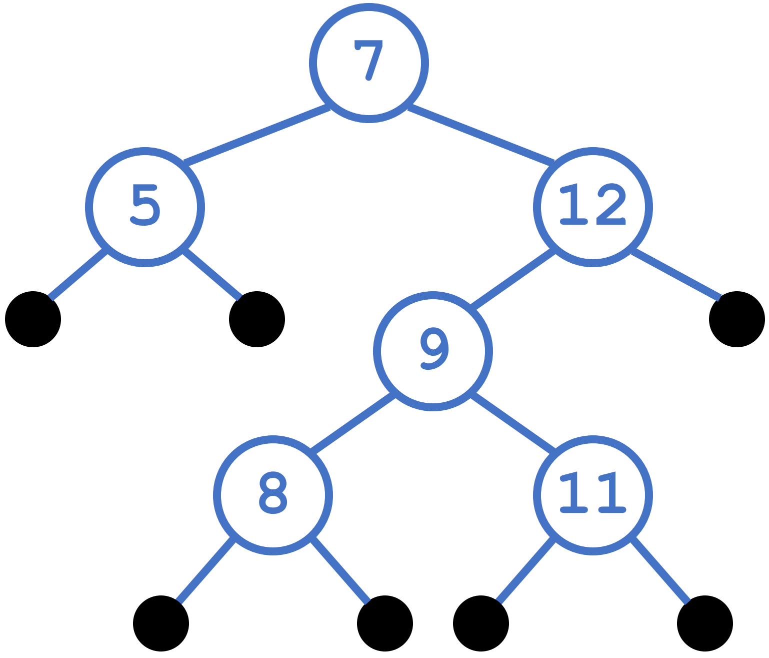
Rotation Example

- Rotate left about 9:



Rotation Example

- Rotate left about 9:



Red-Black Trees: Insertion

- **Case 0:** New node is (z) is always RED .Insert as you would in BST
- **Case 1:** If the new node (z) is RED, and its parent (z.p) is BLACK you don't need to do anything.
- **Case 2:** If a new node (z) is RED and its parent (z.p) is RED, then:
  - **2.a**: if the uncle (y) is BLACK, a **rotation** needs to be performed
    - **2.a.i.** If the insertion path from grand-parent -> parent -> node **BOTH LEFT** then
      - Do RIGHT rotation **around grandparent (z.p.p)**
      - Color flip parent (z.p), grandparent (z.p.p)
    - **2.a.ii.** If the insertion path from grand-parent -> parent -> node **BOTH RIGHT** then
      - Do LEFT rotation **around grandparent** (z.p.p)
      - Color flip parent (z.p), grandparent (z.p.p)
    - **2.a.iii.** If the insertion path from grand-parent -> parent -> node is **LEFT then RIGHT** do:
      - Do LEFT rotation **around parent** (z.p)
      - Do RIGHT rotation around (z)
      - Color flip parent (z.p), grandparent (z.p.p)
    - **2.a.iV.** If the insertion path from grand-parent -> parent -> node is **RIGHT then LEFT** do:
      - Do RIGHT rotation **around parent** (z.p)
      - Do LEFT rotationaround (z)
      - Color flip parent (z.p), grandparent (z.p.p)
  - **2.b:** If the Uncle (y) is RED, a flip parent (z.p), uncle (y) and grandparent (z.p.p) color
- **Case 3:** If the Root is RED, change it to BLACK.

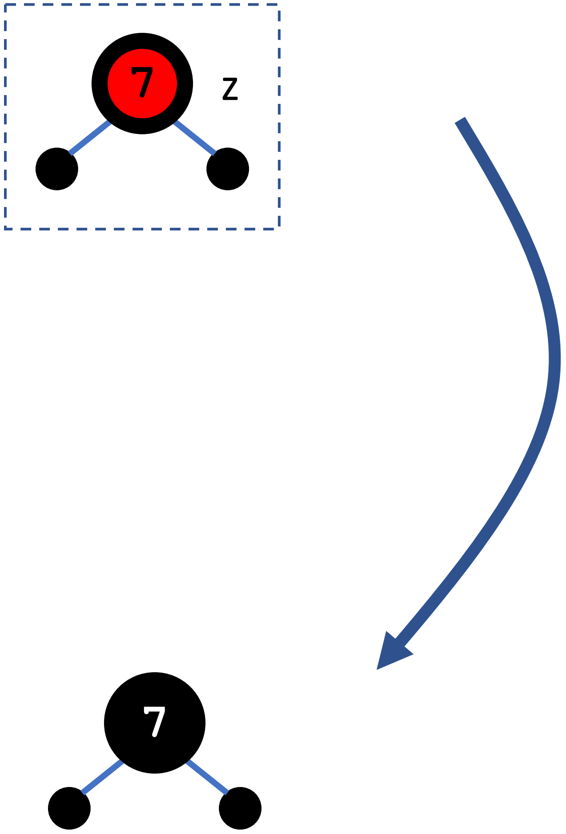
FROM: https://cathyatseneca.gitbook.io/data-strutures-and-algorithms/red-black-trees

Red-Black Trees: Insertion Example

- Insert 7

- **Case 0:** New node is (z) is always RED .Insert as you would in BST

- **Case 3:** If the Root is RED, change it to BLACK.

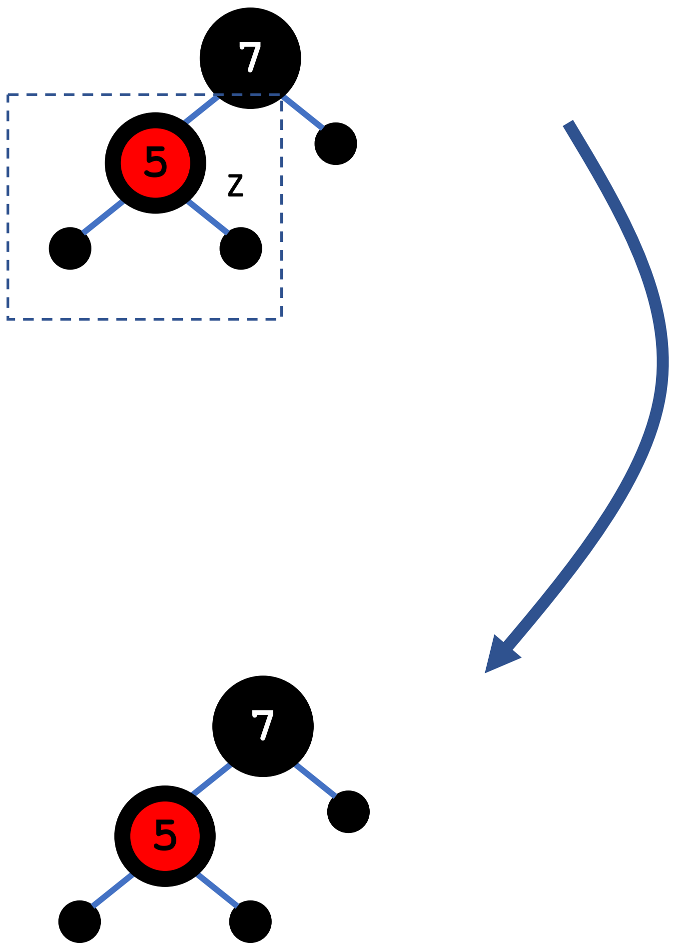


Red-Black Trees: Insertion Example

- Insert 5

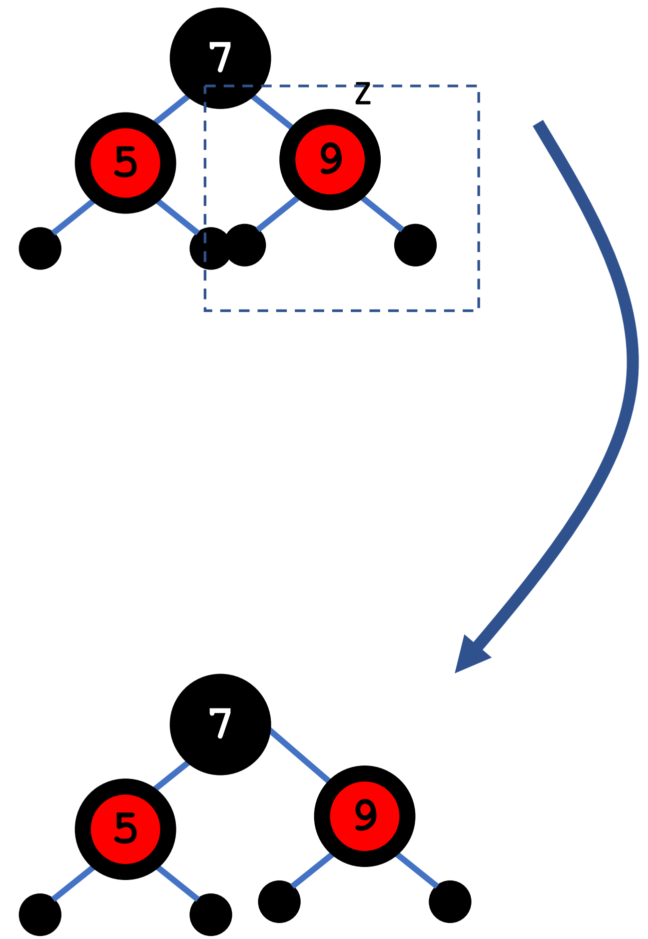
- **Case 0:** New node is (z) is always RED .Insert as you would in BST

- **Case 1:** If the new node (z) is RED, and its parent (z.p) is BLACK you don't need to do anything.



Red-Black Trees: Insertion Example

- Insert 9
- **Case 0:** New node is (z) is always RED .Insert as you would in BST
- **Case 1:** If the new node (z) is RED, and its parent (z.p) is BLACK you don't need to do anything.



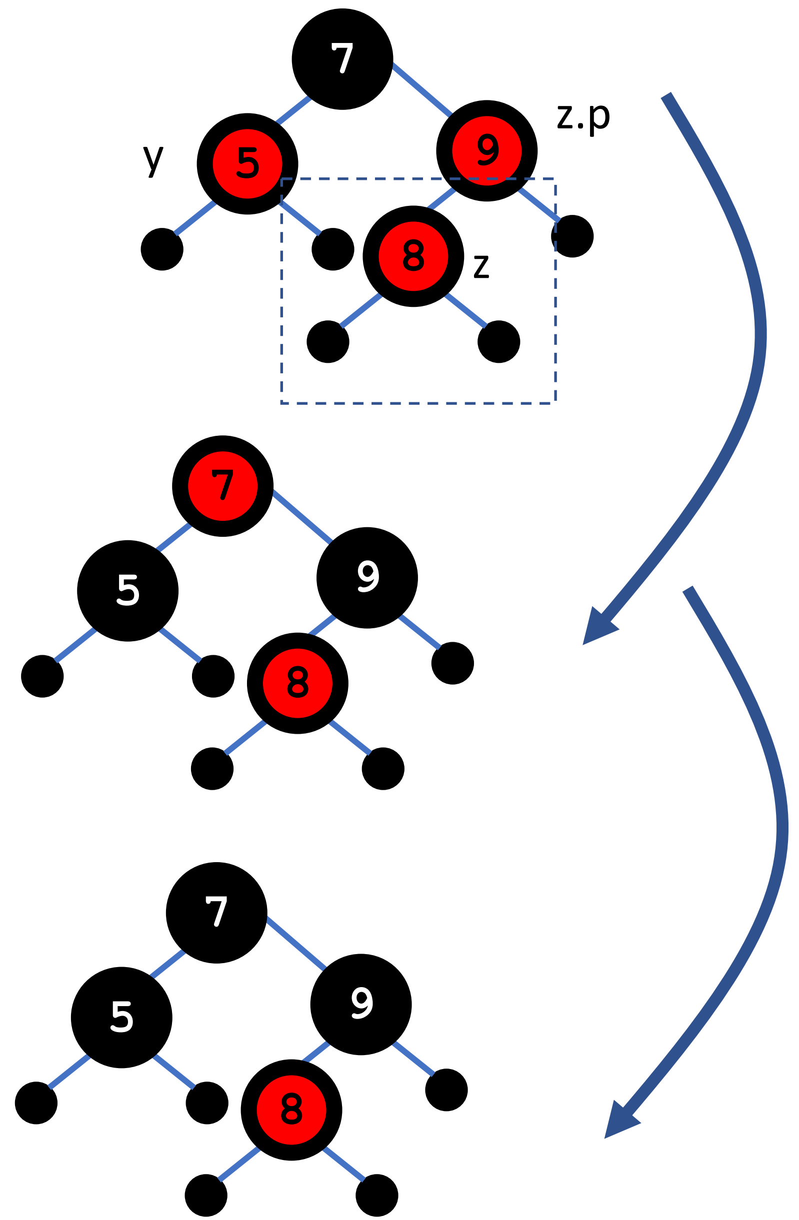
Red-Black Trees: Insertion Example

- Insert 8

- **Case 0:** New node is (z) is always RED .Insert as you would in BST

- **Case 2:** If a new node (z) is RED and its parent (z.p) is RED, then:
  - **2.b:** If the Uncle (y) is RED, a flip parent (z.p), uncle (y) and grandparent (z.p.p) color

- **Case 3:** If the Root is RED, change it to BLACK.

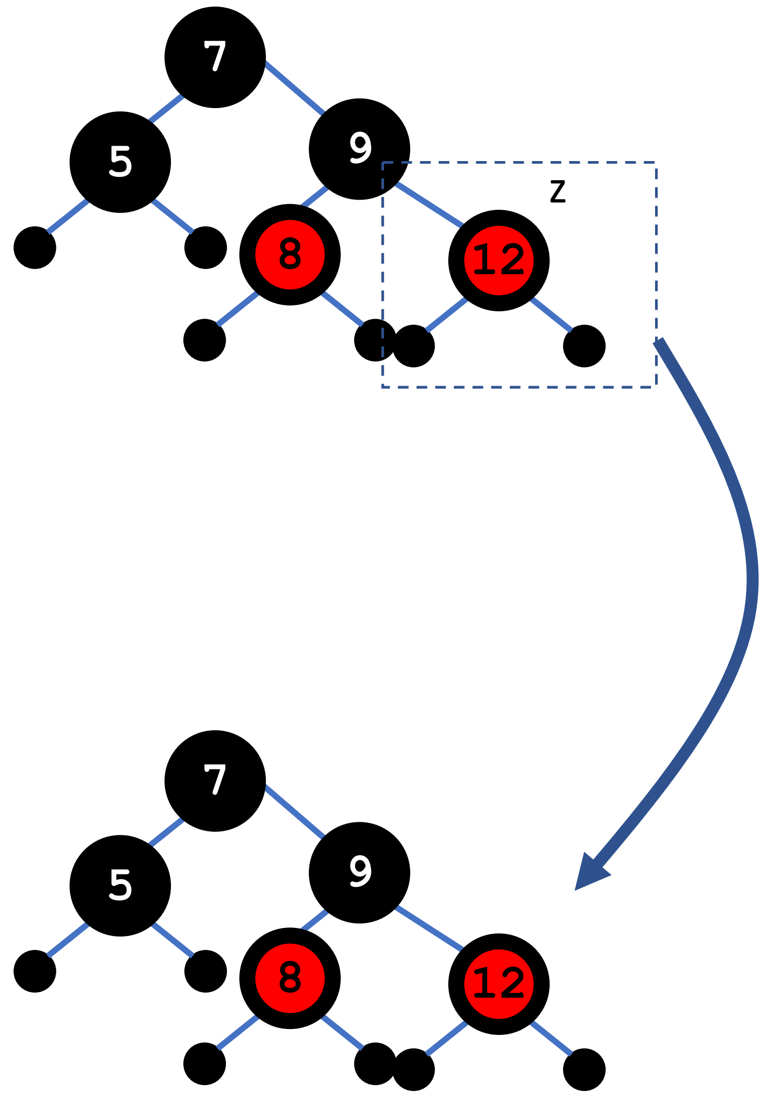


Red-Black Trees: Insertion Example

- Insert 12

- **Case 0:** New node is (z) is always RED .Insert as you would in BST

- **Case 1:** If the new node (z) is RED, and its parent (z.p) is BLACK you don't need to do anything.

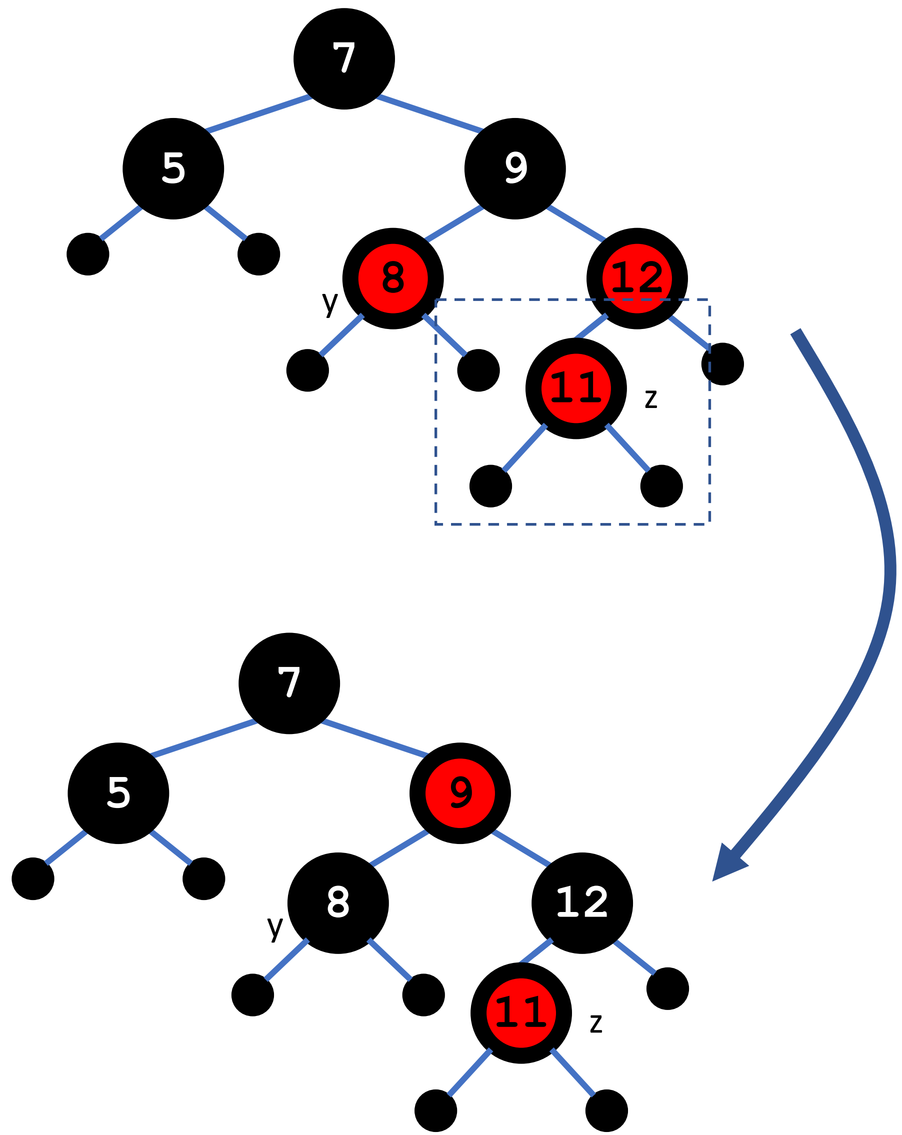


Red-Black Trees: Insertion Example

- Insert 11

- **Case 0:** New node is (z) is always RED .Insert as you would in BST

- **Case 2:** If a new node (z) is RED and its parent (z.p) is RED, then:
  - **2.b:** If the Uncle (y) is RED, a flip parent (z.p), uncle (y) and grandparent (z.p.p) color

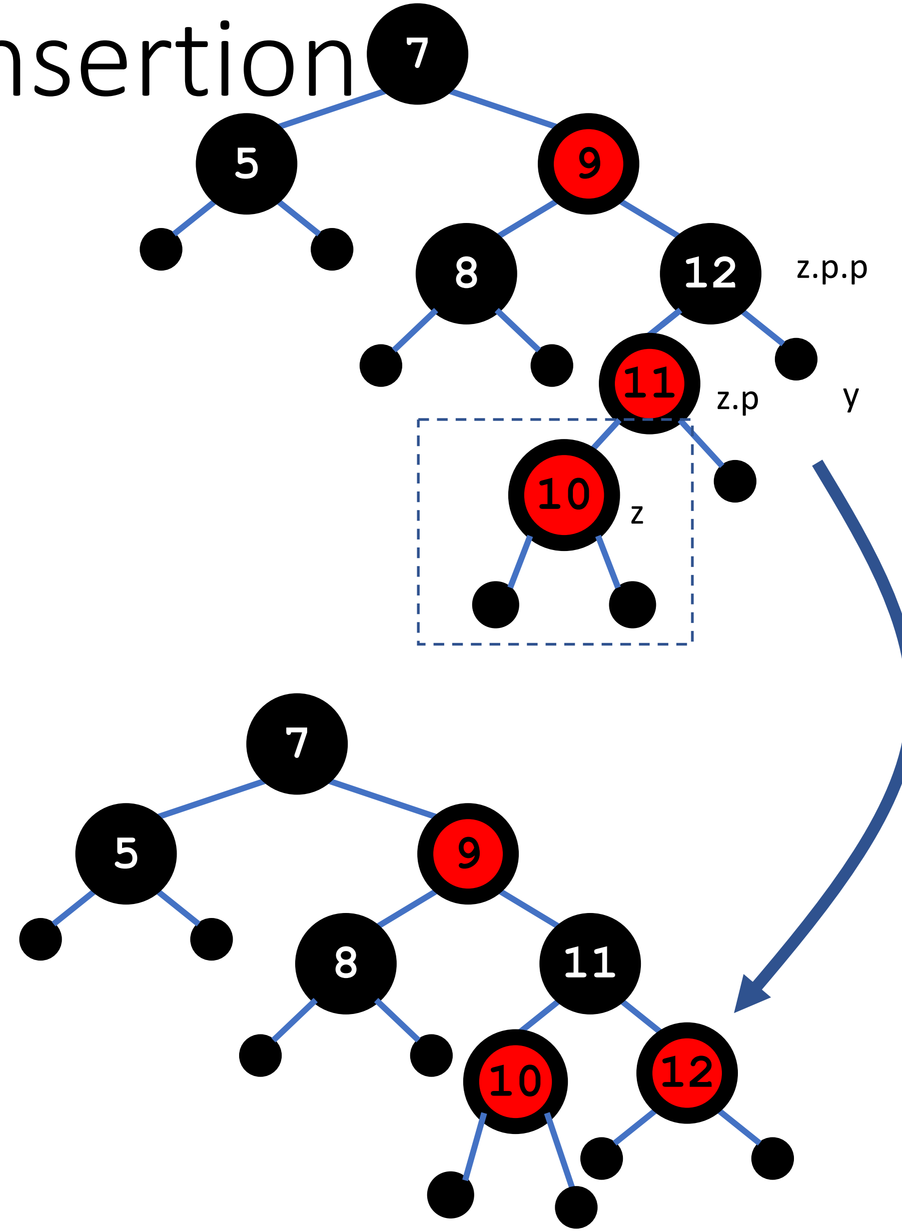


Red-Black Trees: Insertion Example

- Insert 10

- **Case 0:** New node is (z) is always RED .Insert as you would in BST

- **Case 2:** If a new node (z) is RED and its parent (z.p) is RED, then:
  - **2.a**: if the uncle (y) is BLACK, a **rotation** needs to be performed
    - **2.a.i.** If the insertion path from grand-parent -> parent -> node **BOTH LEFT** then
      - Do RIGHT rotation **around grandparent** (z.p.p)
      - Color flip parent (z.p), grandparent (z.p.p)



Red-Black Trees: Deletion

- And you thought insertion was tricky…
- We will not cover RB delete in class
  - If you want you can read section 13.4 of CR book on your own
  - I would recommend read for the overall picture, not the details


That's all Folks!
Any Question ?