6 - Red-Black Trees

Joseph Afework
CS 241

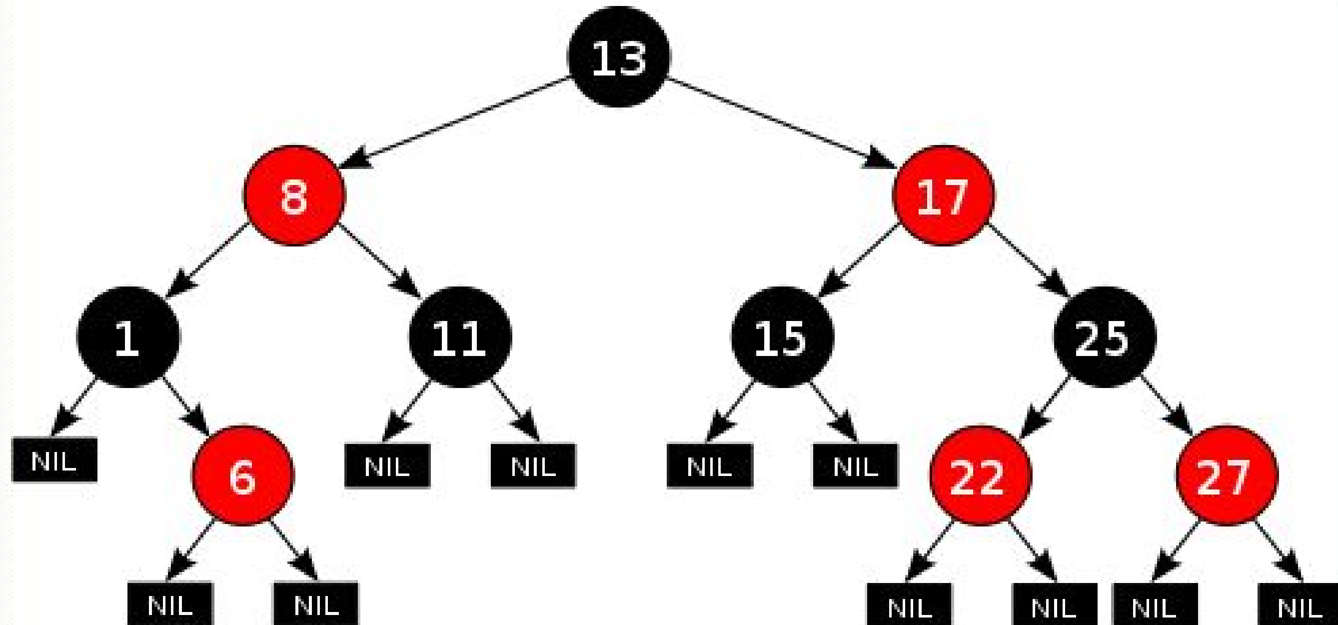
Agenda

- Intro
- Properties
- Search Operation
- Insertion Operation
- Deletion Operations
- AVL vs Red-Black Trees

Reading Assignment

- Read Chapter 27 - Balanced Search Trees
  - Chapter 27 (Read about: AVL, **Red-Black Trees**, B-Trees)

Red-Black Trees



Red-Black Trees

- Type of BST (remember rules of a Binary Search Tree)
- Like an AVL tree:
  - Goal is to keep the tree balanced to yield better run times
  - Special rules/properties are used to **rebalance** the tree after insertions/deletions

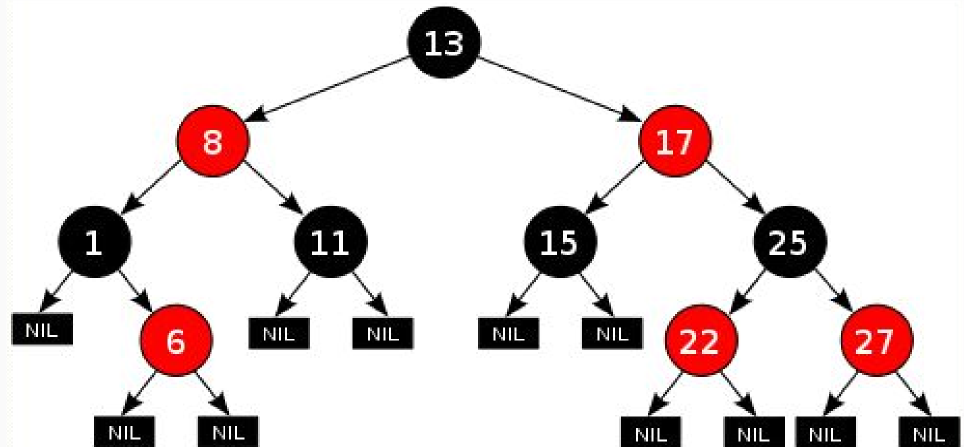
Colors

- **Every node** must be one of two colors:
  - **Red**
  - **Black**

- A node being a specific color must follow certain rules.
- The color information needs to be stored alongside the key in the tree

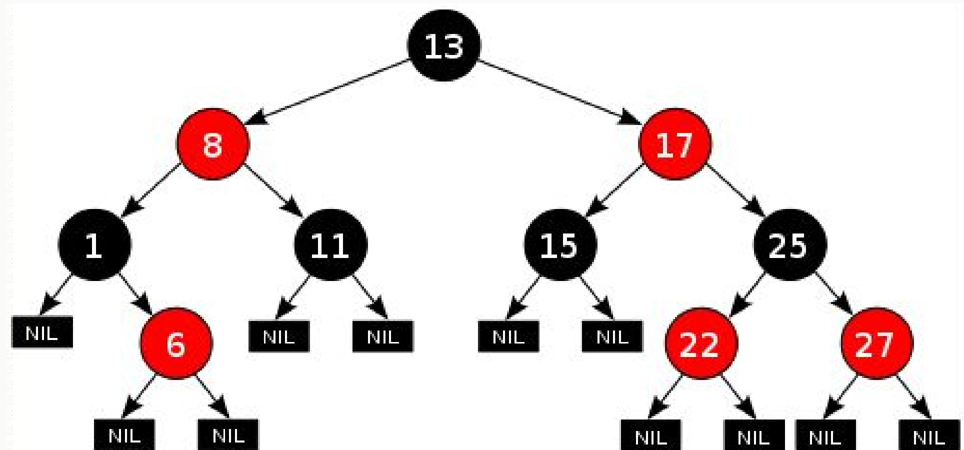
Red Nodes

- Every **Red** node can only have two black child nodes:
  - A red node cannot have a child node that has the color red
  - Can't have two reds in a row
  - Every red node has a black parent



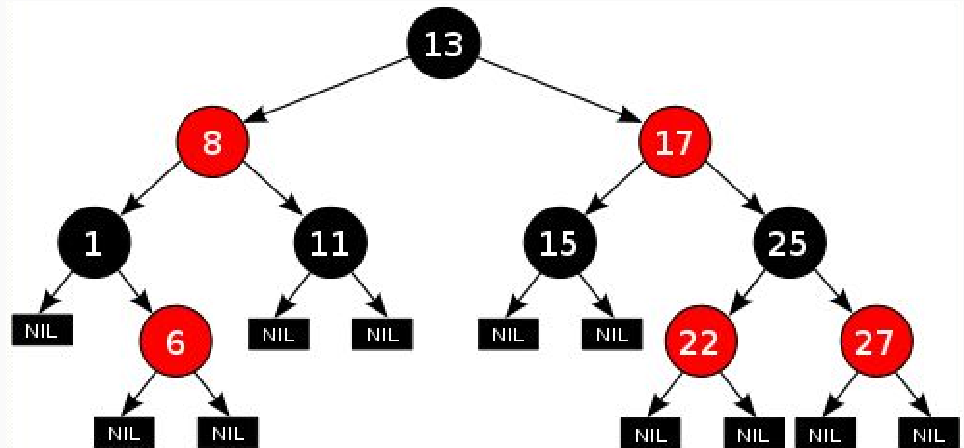
Black nodes

- **Leaf** nodes are always **null**.
  - Leaf nodes never store values.
- **Leaf** nodes are always **black**.
- The **root** node is always **black**.



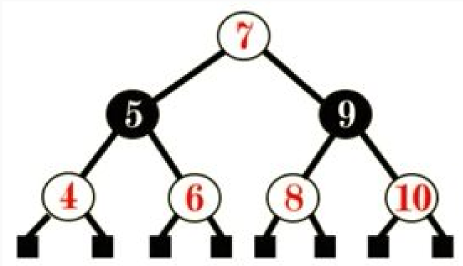
Paths

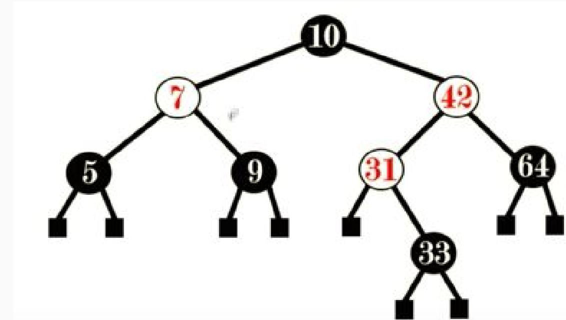
- **Every** path from any node to any descendant leaf node must contain the same number of black nodes.



ICE: 7.1 Identification

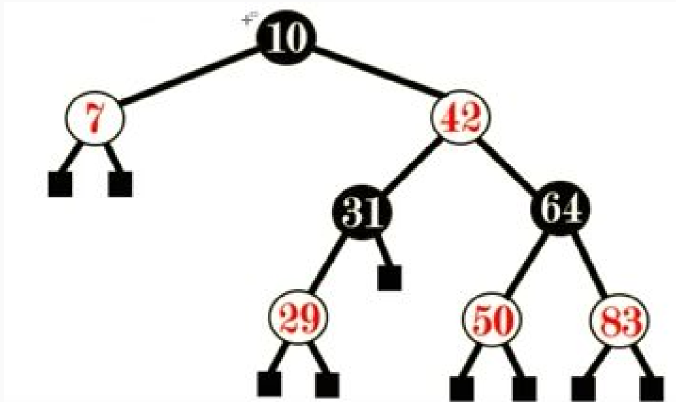
Which of the following are **not** Red-Black Trees?

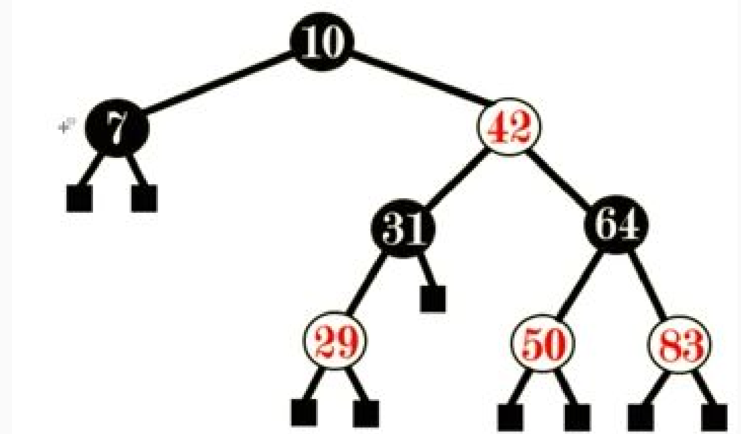




ICE: 7.1 Identification Contd.

Which of the following are **not** Red-Black Trees?





Observations

1. The longest path from the root to any leaf is no more than twice as long as the shortest path from the root to any other leaf in that tree.
2. Tree is roughly balanced.
3. Insertion, deletion, and search lean towards log(n).

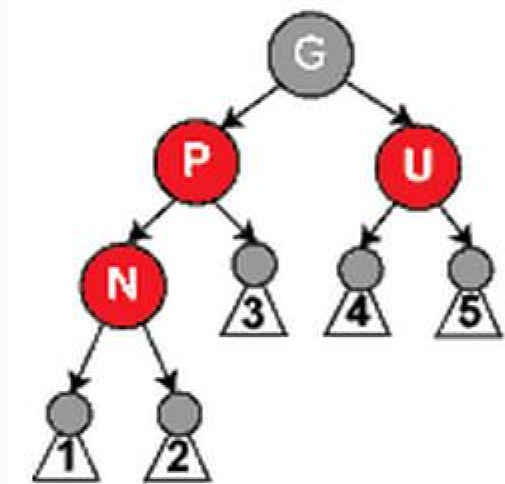
Terms

N: Inserted Node

G: grandparent Node

P: parent Node

U: uncle Node



Insertion

1) Insert a value according to the rules of BST.
2) All inserted nodes start colored as **Red** by default.
3) From the node inserted, and examine the relevant nodes (parent, grandparent, uncle nodes) to determine if a case has been violated.
4) Resolve violation according to rules for the case
5) Reorder/Recolor nodes as needed in accordance to the rules

Violation Case 1

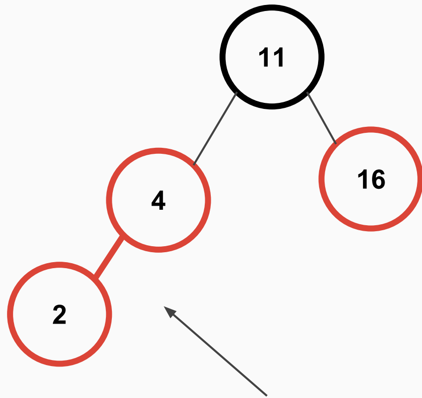
1. Inserted node is child of a red node
2. Uncle Node is red

How to Fix:

1. Swap the color of the parent, grandparent and uncle
2. Check from grandparent upwards for further violations (recursive)

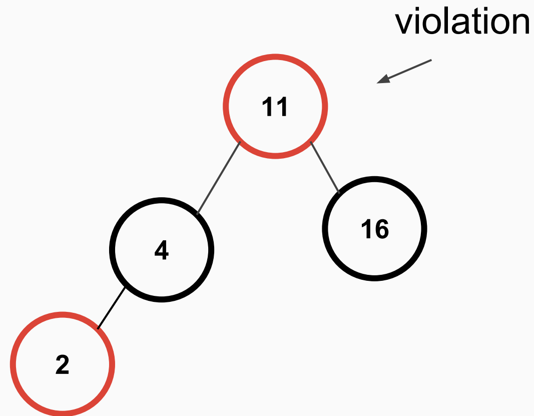
Case 1 Contd.

2 is inserted

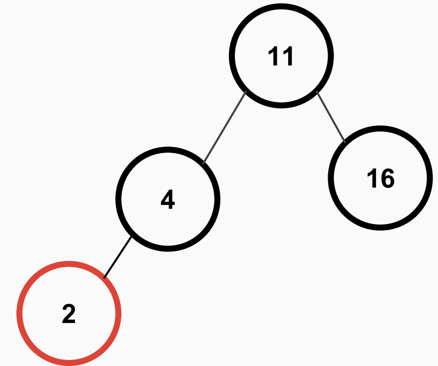


violation

Swap Colors



Check Grandparent



Violation Case 2

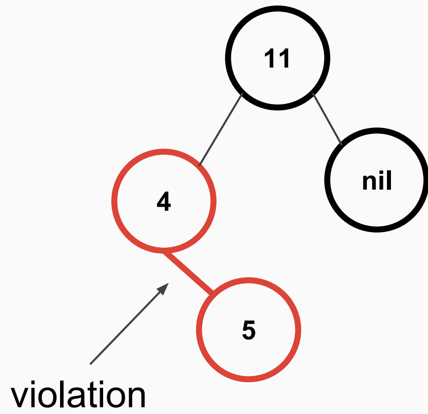
1. Inserted node is child of a red node
2. Uncle Node is black
3. The inserted node/violating node is on the (inside of the subtree)

How to Fix:

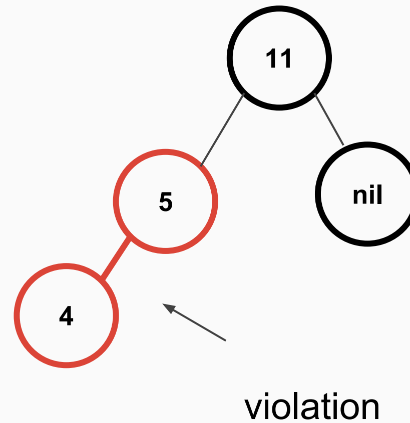
1. Rotate the parent node downward (think AVL rotations)..
   a. Turns Case 2 violation into a Case 3 violation
2. Follow rules for Case 3

Case 2 Contd.

5 is inserted



Rotate (convert to Case 3 violation)



Violation Case 3

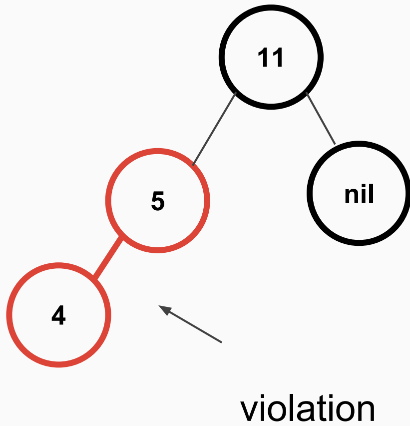
1. Inserted node is child of a red node
2. Uncle Node is black
3. The inserted node/violating node is on the (outside of the subtree)

How to Fix:

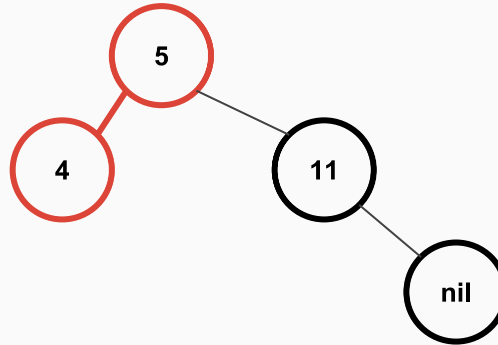
1. Rotate around the grandparent node (think AVL rotations)..
2. Swap the colors of the parent and the grandparent

Case 3

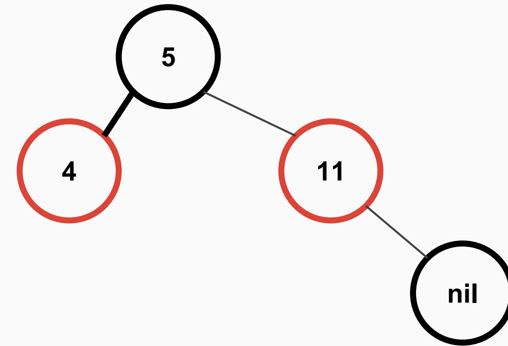
Remember: 5 was inserted



Rotate around grandparent



Swap Colors

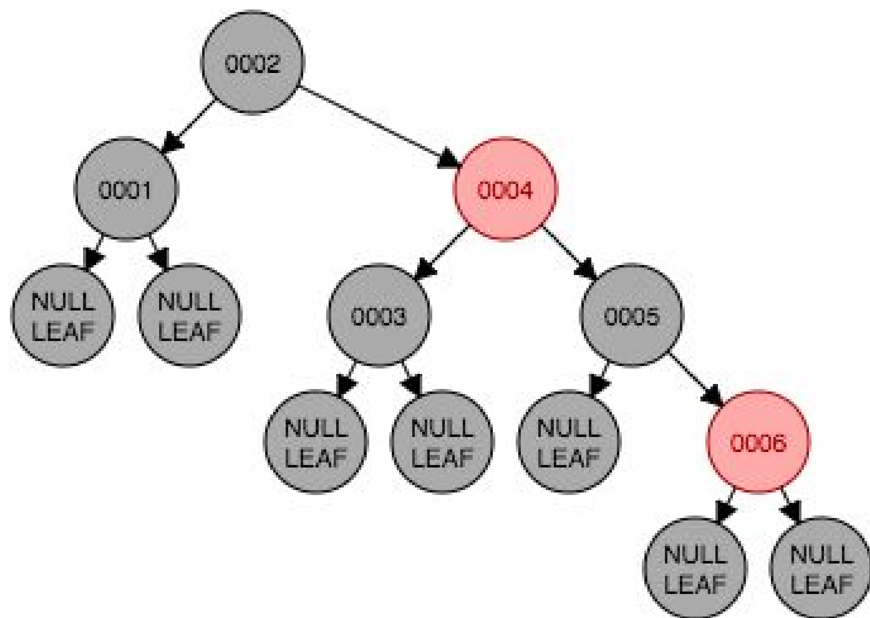


ICE: 7.2 Red-Black Tree Insertion

Insert the following values, rebalance the tree according to the Red-Black tree rules.

1 2 3 4 5 6

Solution



ICE: 7.3 Red-Black Tree Insertion

Insert the following values, rebalance the tree according to the Red-Black tree rules as you are inserting the values.

14, 17, 11, 7, 53, 4, 13, 12, and 8

Deletion

- Deletion process is fairly complicated
- There are a few basic states, and numerous edge cases that must be identified and followed to successfully delete and rebalance the tree.
    - Roughly 10 cases or so depending on implementation
    - BST only had 3 cases...
- Checkout Wikipedia for a more information
    - https://en.wikipedia.org/wiki/Red%E2%80%93black_tree

Performance

Red Black Trees	Average Case	Worst Case
Insert	O(log(n))	**~ O(log(n))**
Delete	O(log(n))	**~ O(log(n))**
Search	O(log(n))	**~ O(log(n))**

AVL vs Red-Black Trees

1. Generally more balanced: **AVL**
2. Greater rotational operations/overhead: **AVL**
3. Trying to perform more insertions/deletions than searches: **Red-Black Trees**
4. Trying to perform more searches than insertions/deletions: **AVL** (AVL is more balanced)

R-B Applications

- Used in:
  - Java:
    - java.util.TreeMap
    - java.util.TreeSet
  - C++:
    - STL map,
    - multimap,
    - multiset

Resources

https://en.wikipedia.org/wiki/Red%E2%80%93black_tree

https://www.cs.usfca.edu/~galles/visualization/RedBlack.html

http://docs.oracle.com/javase/7/docs/api/java/util/TreeMap.html