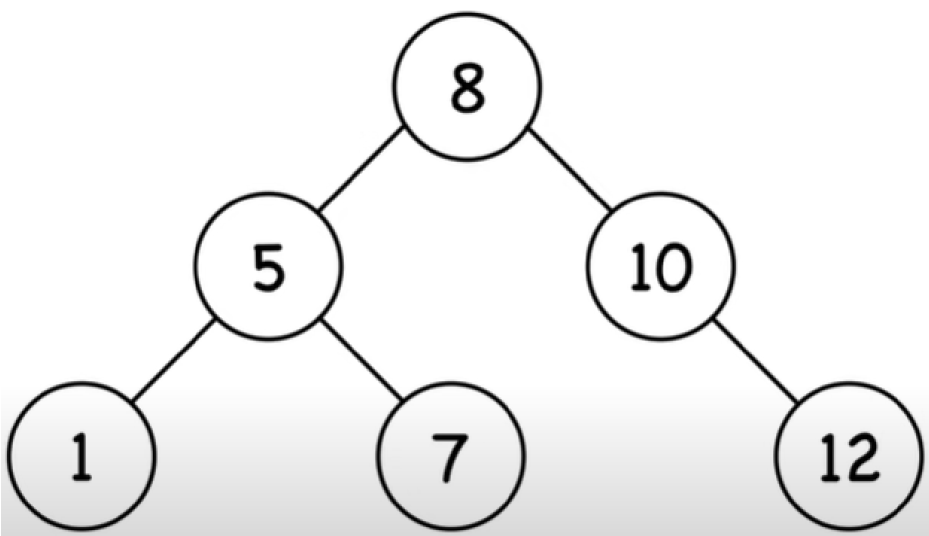
Lecture 9

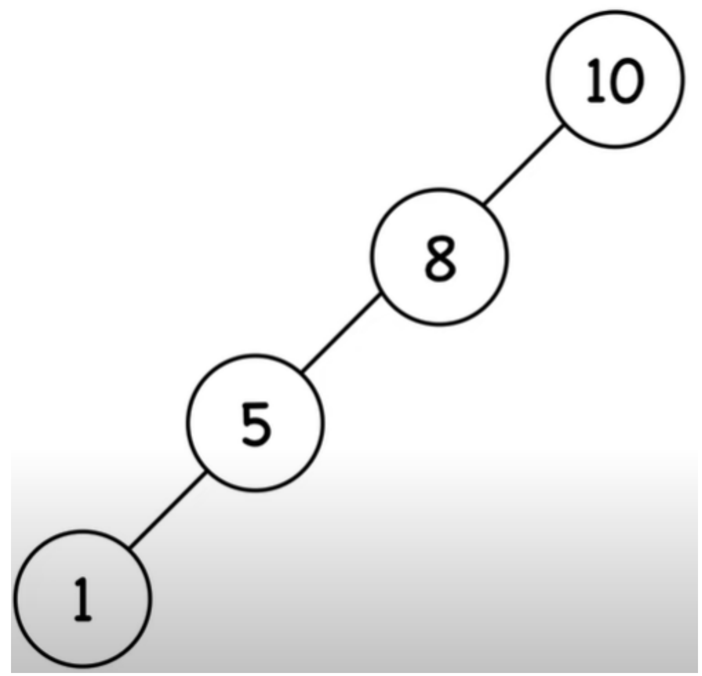
Red-Black Trees

Department of Computer Science
Hofstra University

Binary Search Trees

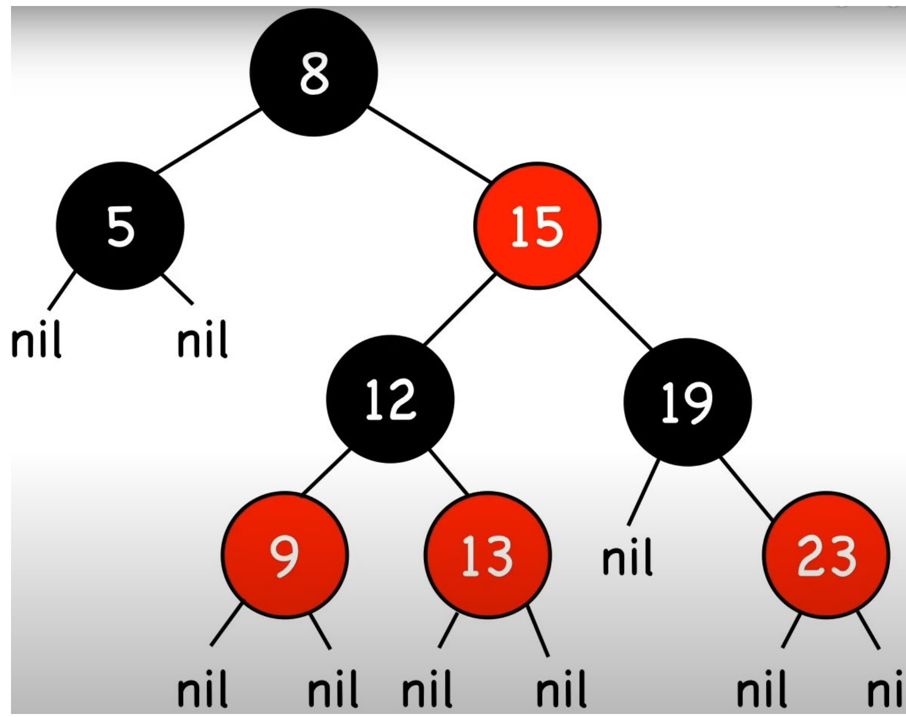
- Ordered, or sorted, binary trees.
- Each node can have 2 subtrees.
- Items to the left of a given node are smaller.
- Items to the right of a given node are larger.
- Balanced search trees have guaranteed height of O(log n) for n items
  - Red-Black Tree is a type of balanced search tree





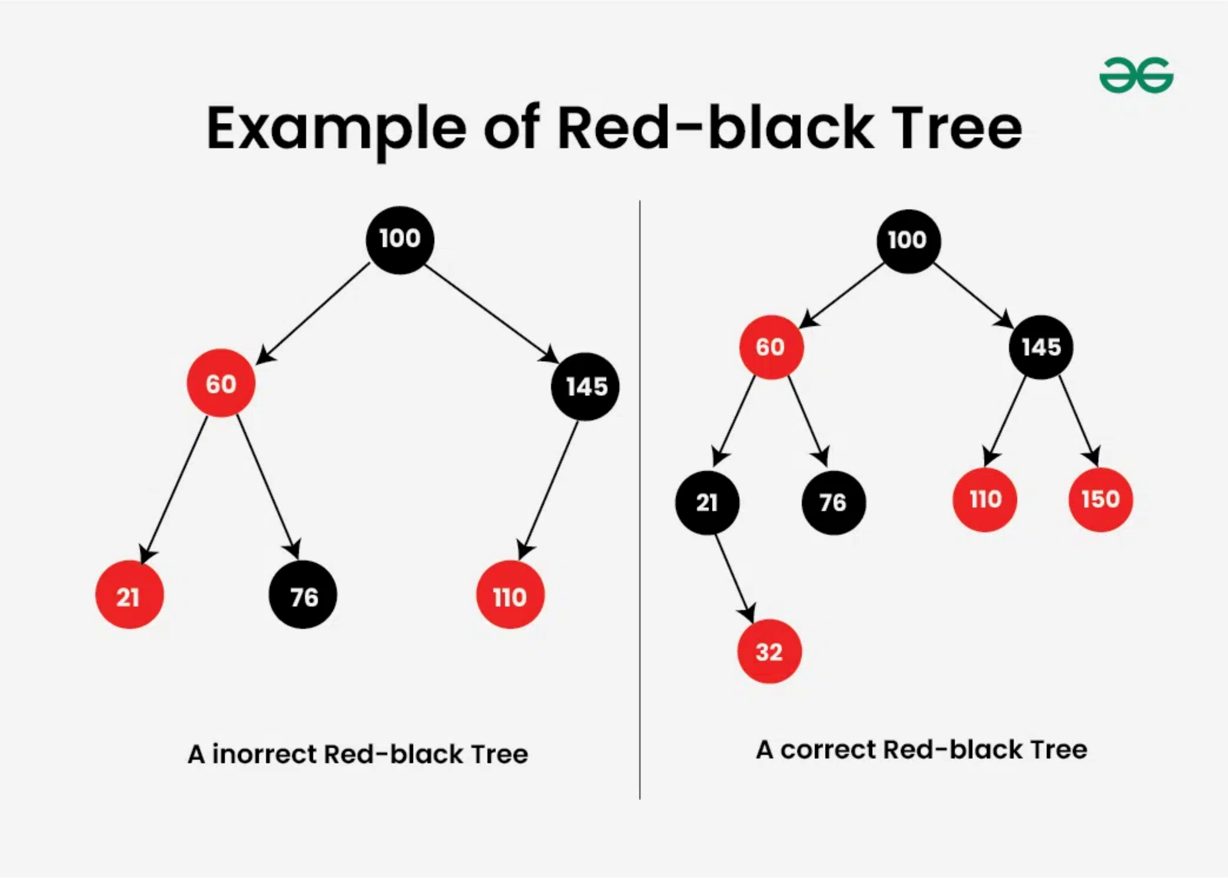
Red-Black Tree

- 1. Node Color: A node is either red or black.
- 2. Root Property: The root and leaves (NIL) are black.
- 3. Red Property: If a node is red, then its children are black.
- 4. Black Property: All paths from a node to its NIL descendants contain the same number of black nodes.
  - Path length excludes root node itself, so here each path contains 1 black node



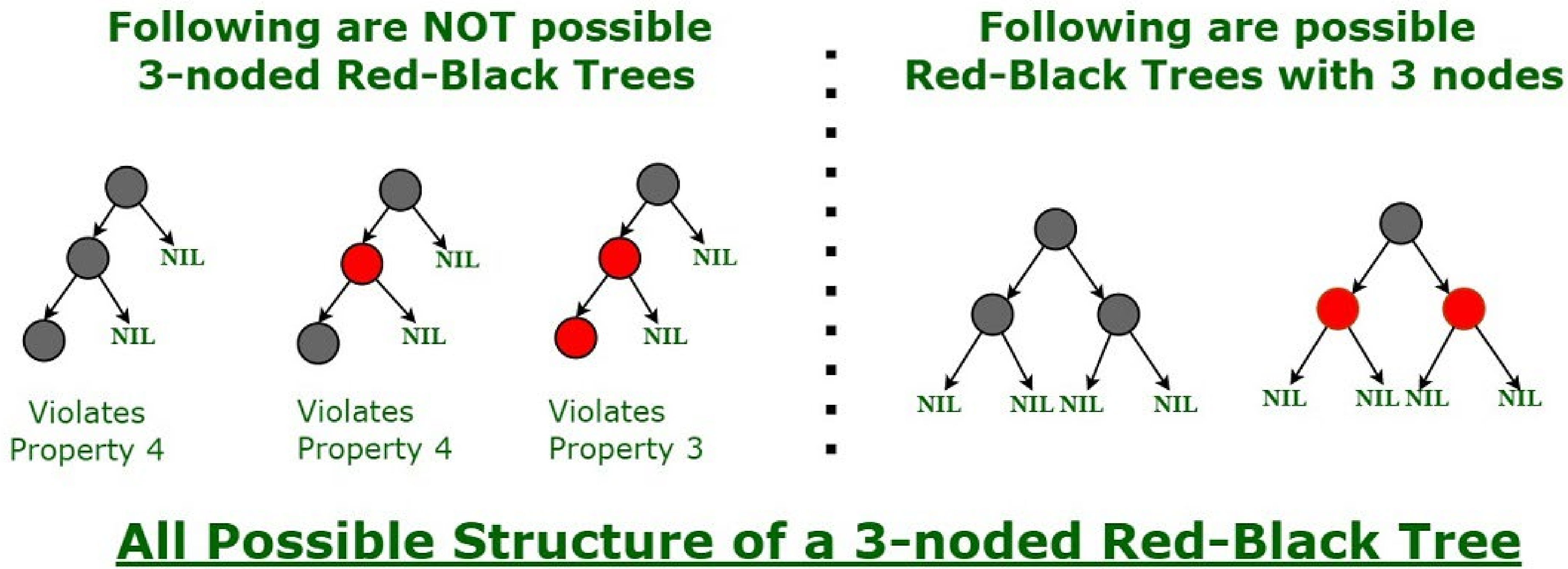
Example

- Tree on the left: Incorrect Red Black Tree.
  - Two red nodes are adjacent to each other.
  - One of the paths to a leaf node has zero black nodes, whereas the other two paths contain 1 black node each.



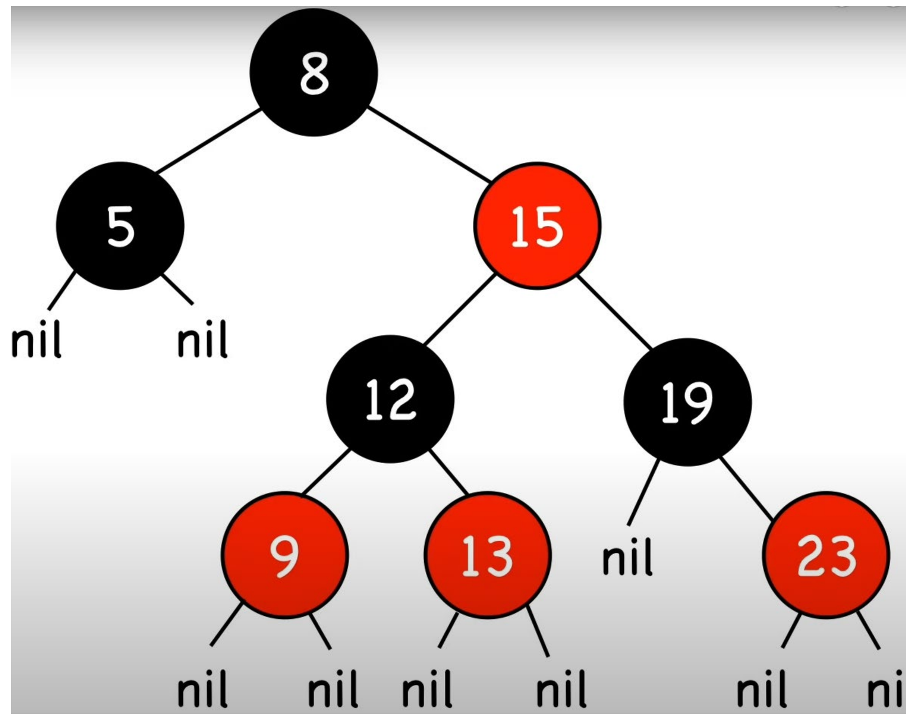
Red-Black tree ensures balancing

- A chain of 3 nodes is not possible in a Red-Black tree



Additional Properties

- Balanced search tree: the longest path (root to farthest NIL) is no more than twice the length of the shortest path (root to nearest NIL).
  - Shortest path: all black nodes (=2)
  - Longest path: alternating red and black (=4)
- Operations: search, insert, remove, each with time complexity O(log(n)).
  - Insert and remove may result in violation of red-black tree properties, use rotations to fix it



Rotations

- Alters the structure of a tree by rearranging subtrees
- Goal is to decrease the height of the tree to maximum height of O(log n)
  - Larger subtrees up, smaller subtrees down
- Does not affect the order of elements
- Time complexity O(1)

Before Rotation:

```
x
 \
  y
 / \
a   b
```

Before Rotation:

```
    x
   /
  y
 / \
a   b
```

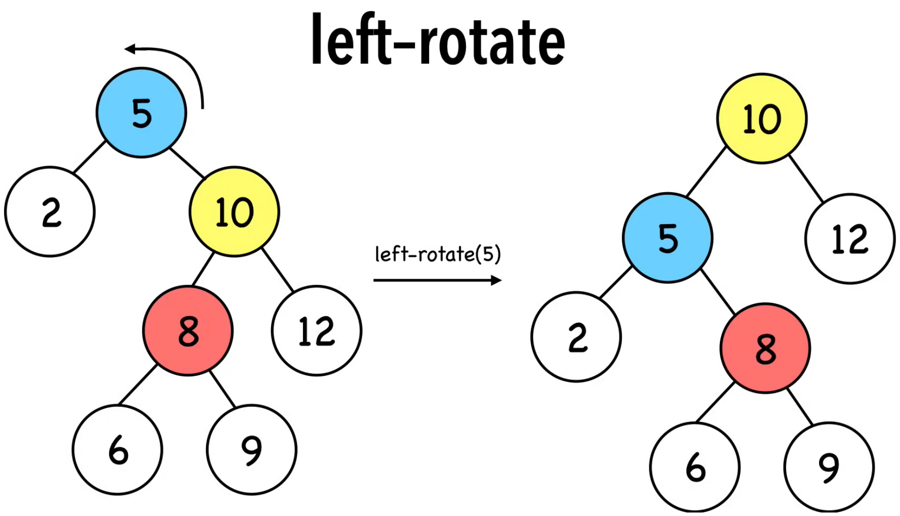
After Left Rotation:

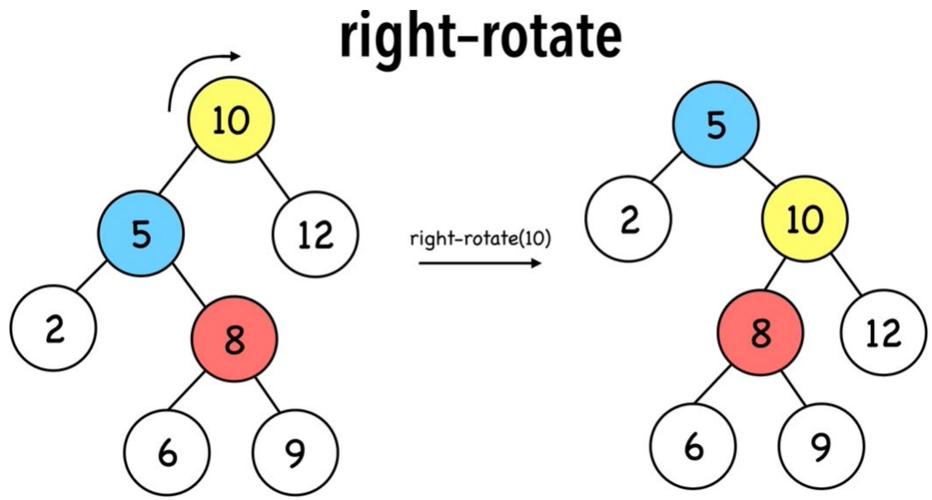
```
  y
 / \
x   b
 \
  a
```

After Right Rotation:

```
  y
 / \
a   x
   /
  b
```

Rotations Examples





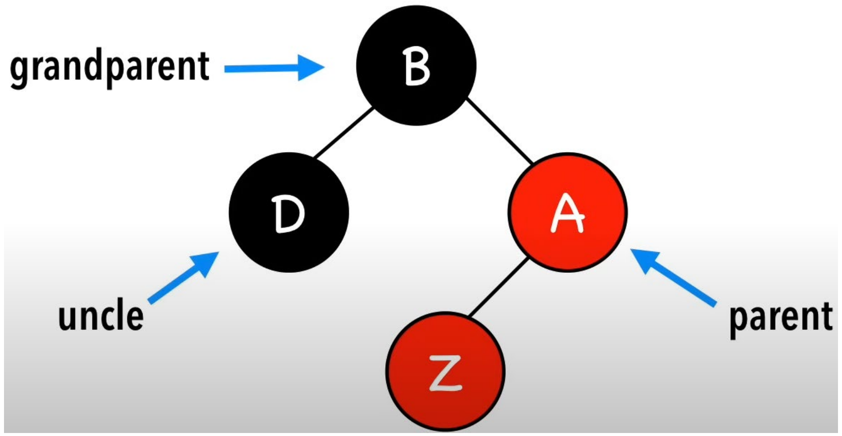
Insertion

- Inserting a new node in a Red-Black Tree involves a two-step process: performing a standard binary search tree (BST) insertion, followed by fixing any violations of Red-Black properties.
- **Insertion Steps**

1. **BST Insert**: Insert the new node into BST and color it red.
2. **Fix Violations**:
    2. If the parent of the new node is **black**, no properties are violated.
    3. If the parent is **red**, the tree might violate the Red Property, requiring fixes.

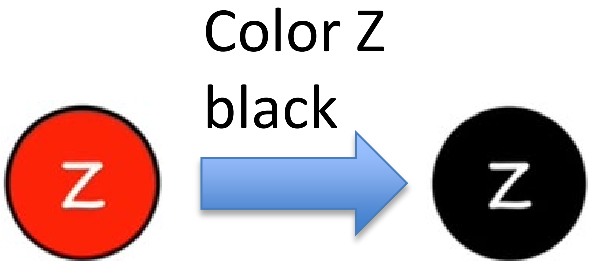
Insertions

- Step 1. Insert Z and color it red
- Step 2. Recolor and rotate nodes to fix violations
- 4 scenarios after inserting node Z
- Case 0. Z = root
  - Color Z black
- Case 1. Z.uncle = red
  - Recolor Z's parents and grandparent
- Case 2. Z.uncle = black (triangle)
  - Rotate Z.parent
- Case 3. Z.uncle = black (line)
  - Rotate Z.grandparent & Recolor Z's parents and grandparent



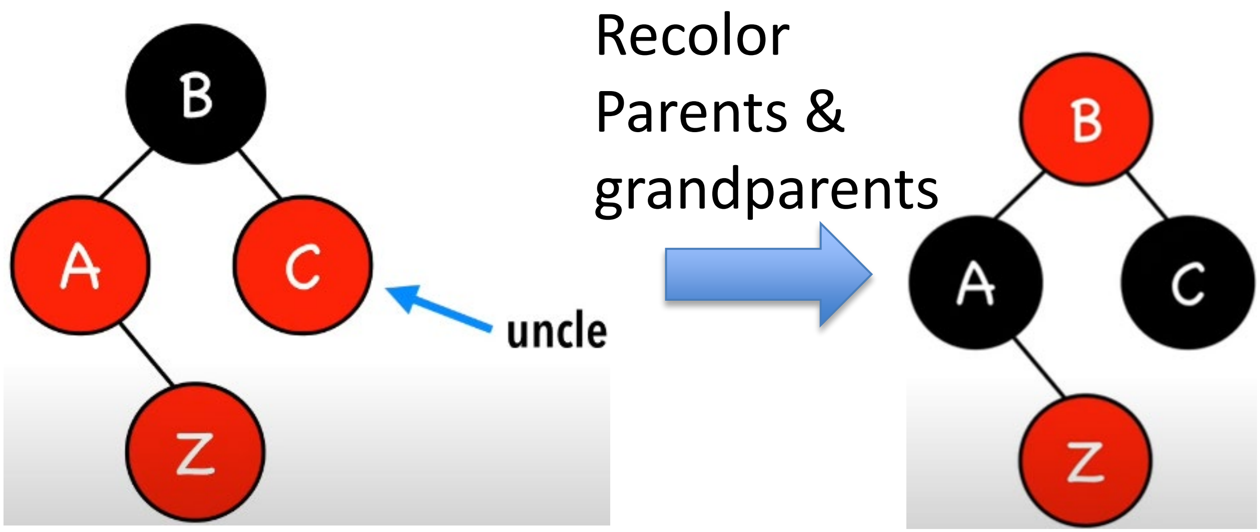
Case 0. Z = root

- Color Z black



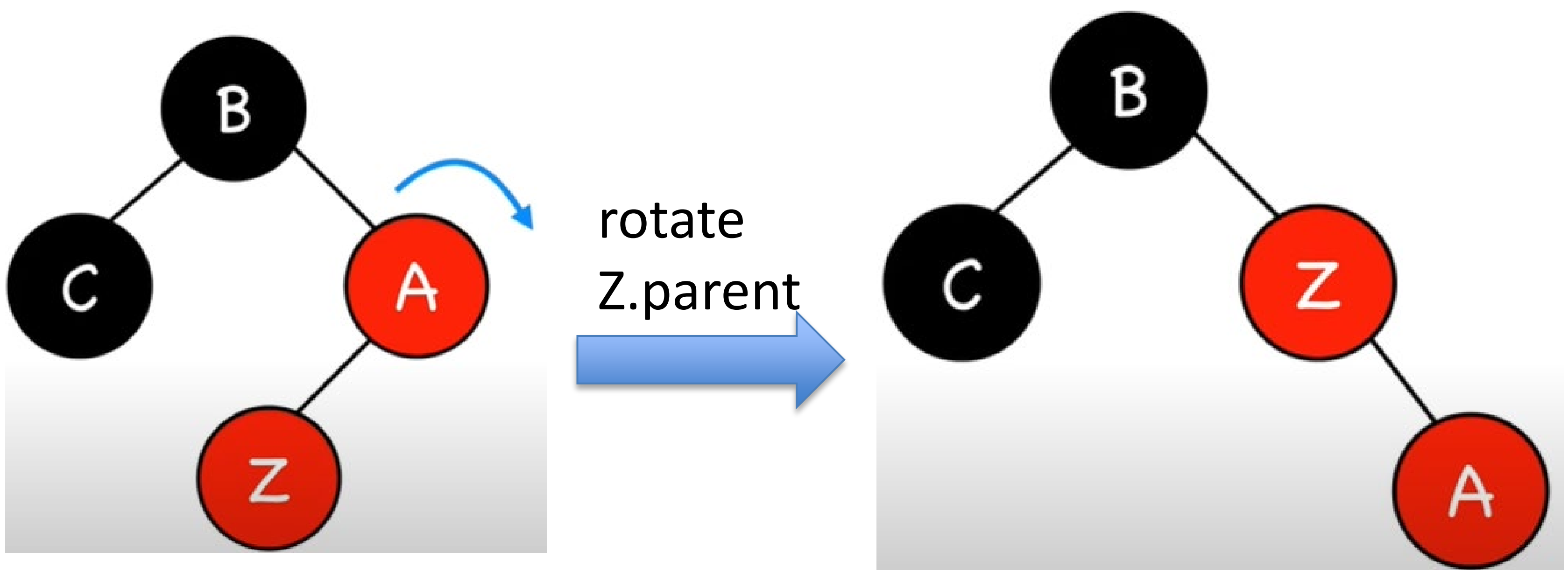
Case 1. Z.uncle = red

- Recolor Z's parents and grandparent

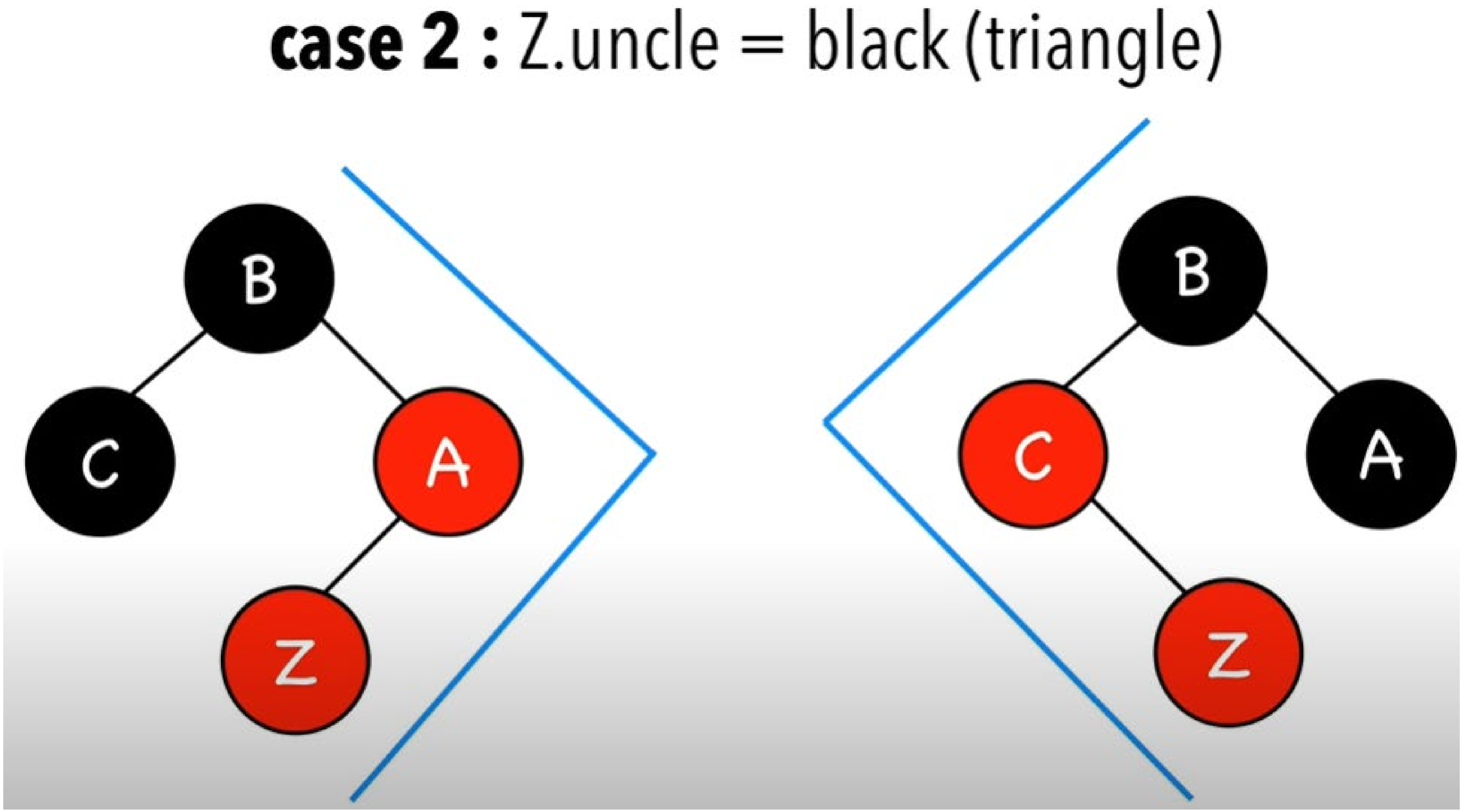


Case 2. Z.uncle = black (triangle)

- Rotate Z.parent

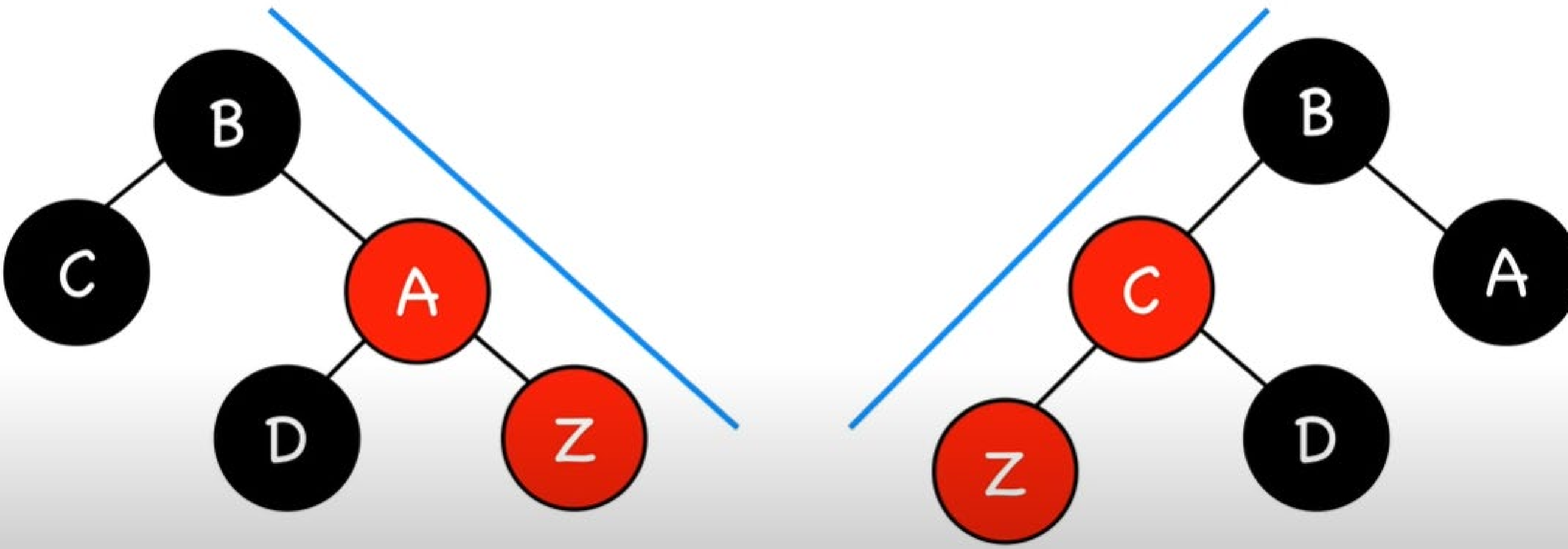


Case 2. Z.uncle = black (triangle)



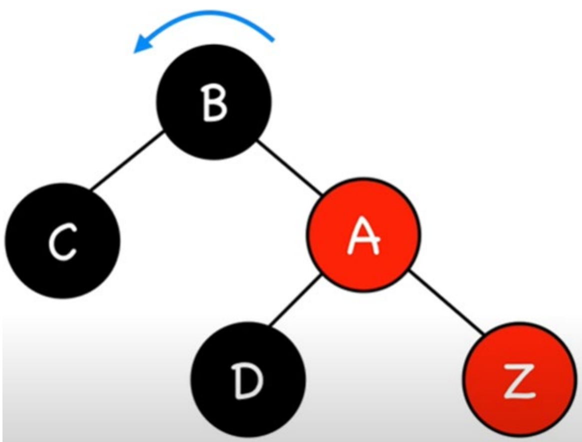
Case 3 Z.uncle = black (line)

case 3 : Z.uncle = black (line)

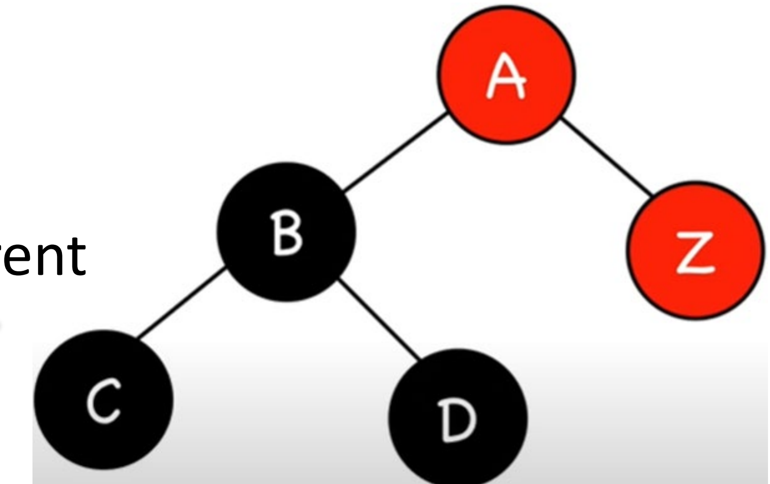


Case 3 Z.uncle = black (line)

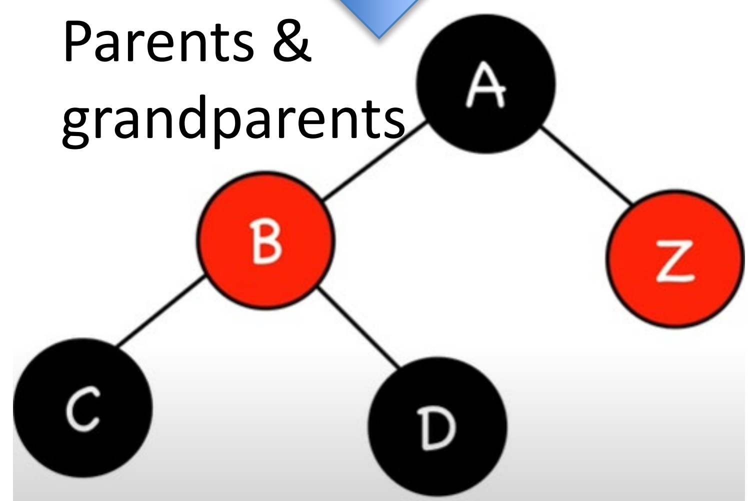
- Rotate Z.grandparent



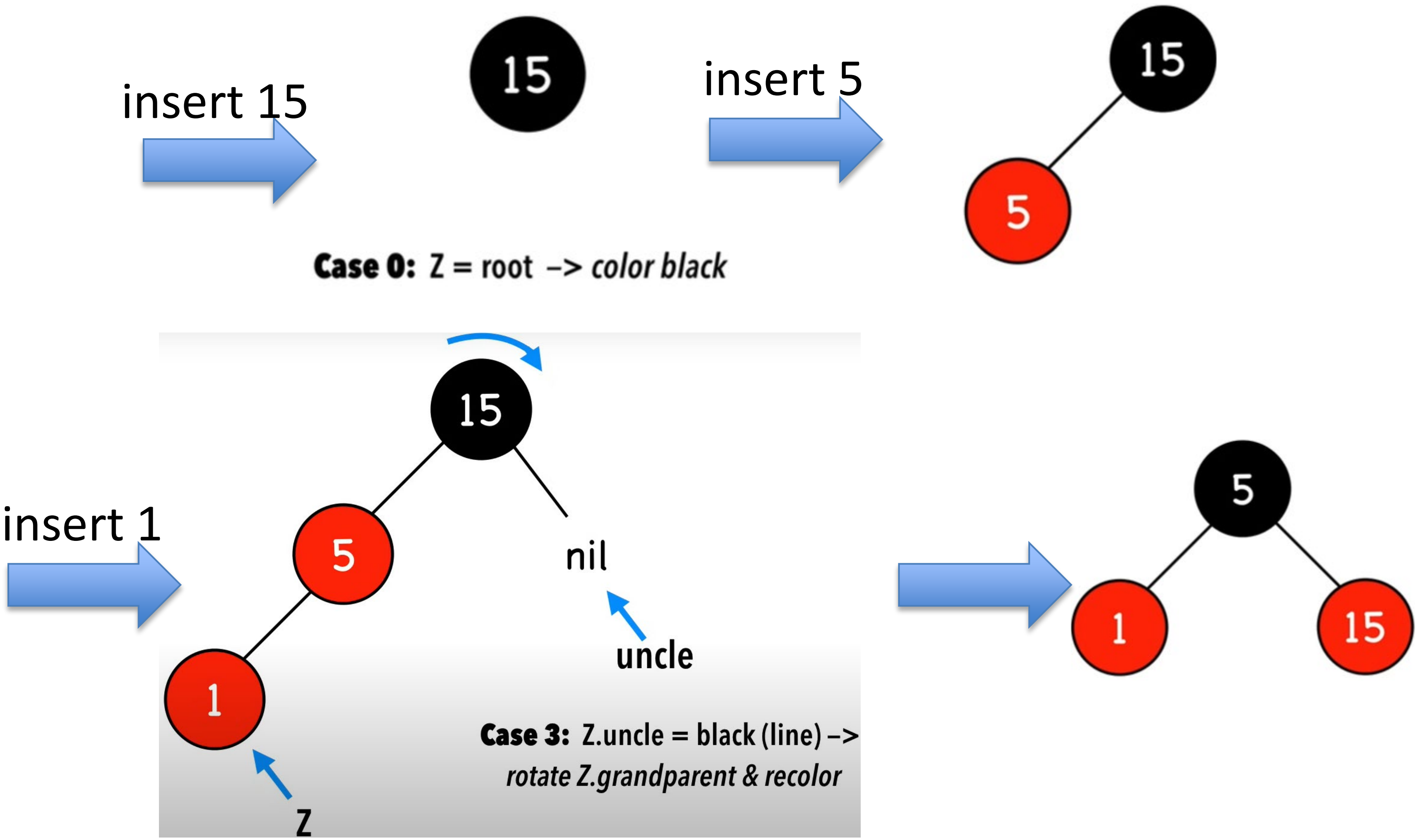
rotate
Z.grandparent



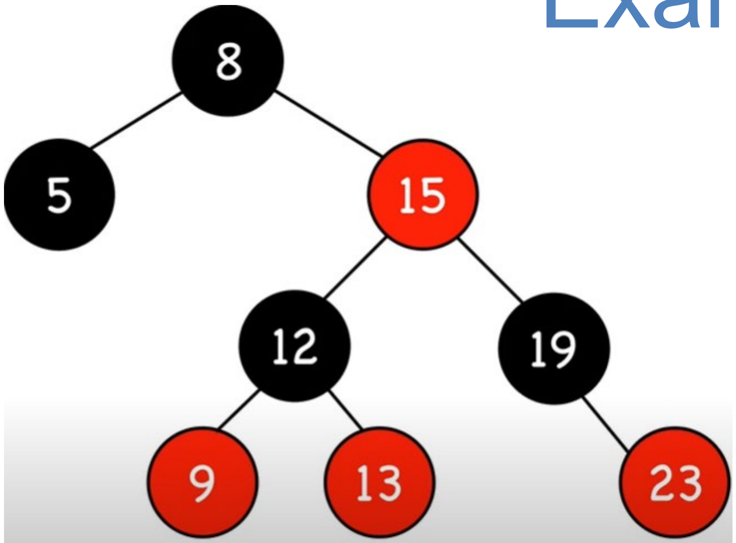
recolor
Parents &
grandparents



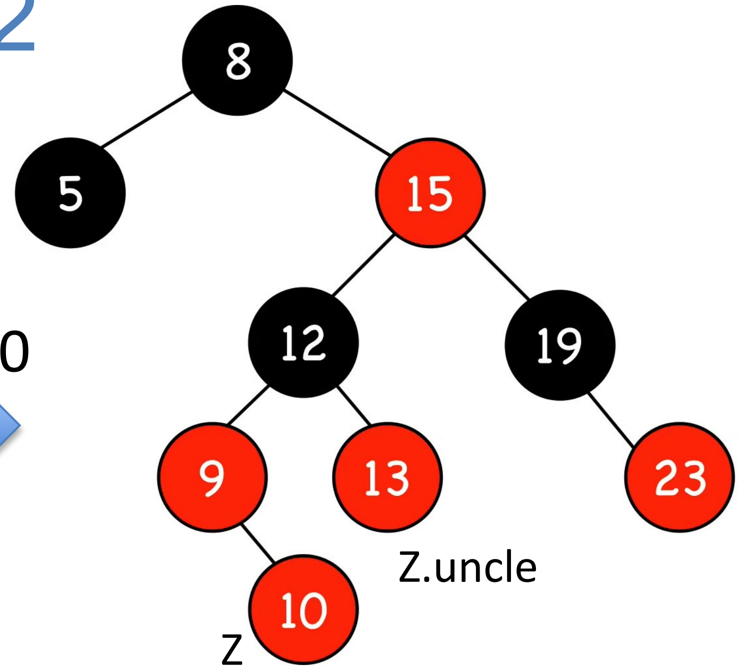
Example 1


insert 15
15
insert 5
15
5
Case 0: Z = root –> color black
15
insert 1
5
nil
uncle
1
Z
Case 3: Z.uncle = black (line) –>
rotate Z.grandparent & recolor
5
1
15

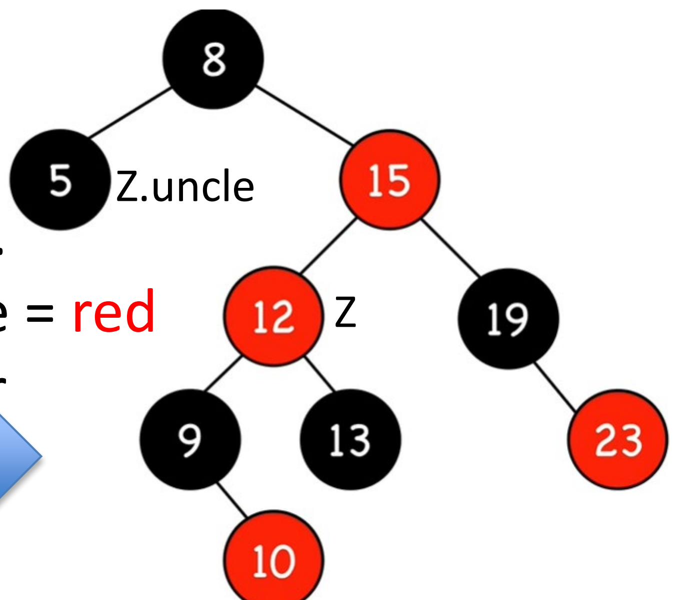
Example 2



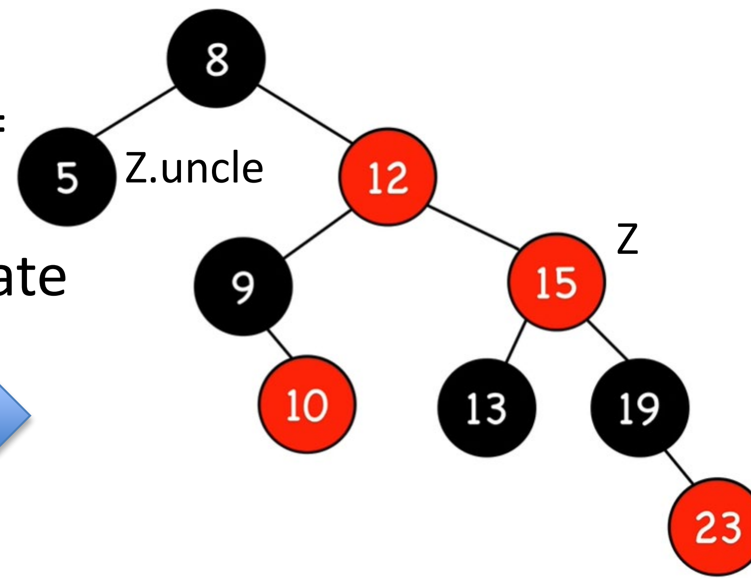
insert 10



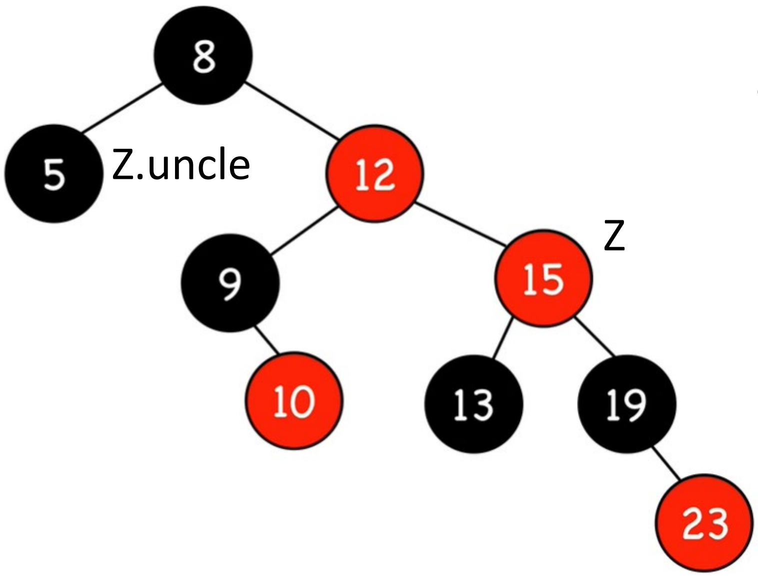
Case 1.
Z.uncle = red
recolor



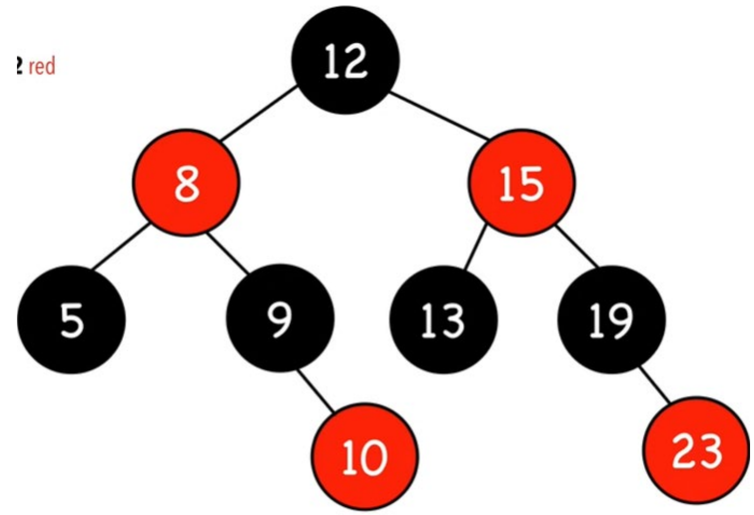
Case 2.
Z.uncle = black
right rotate on 15



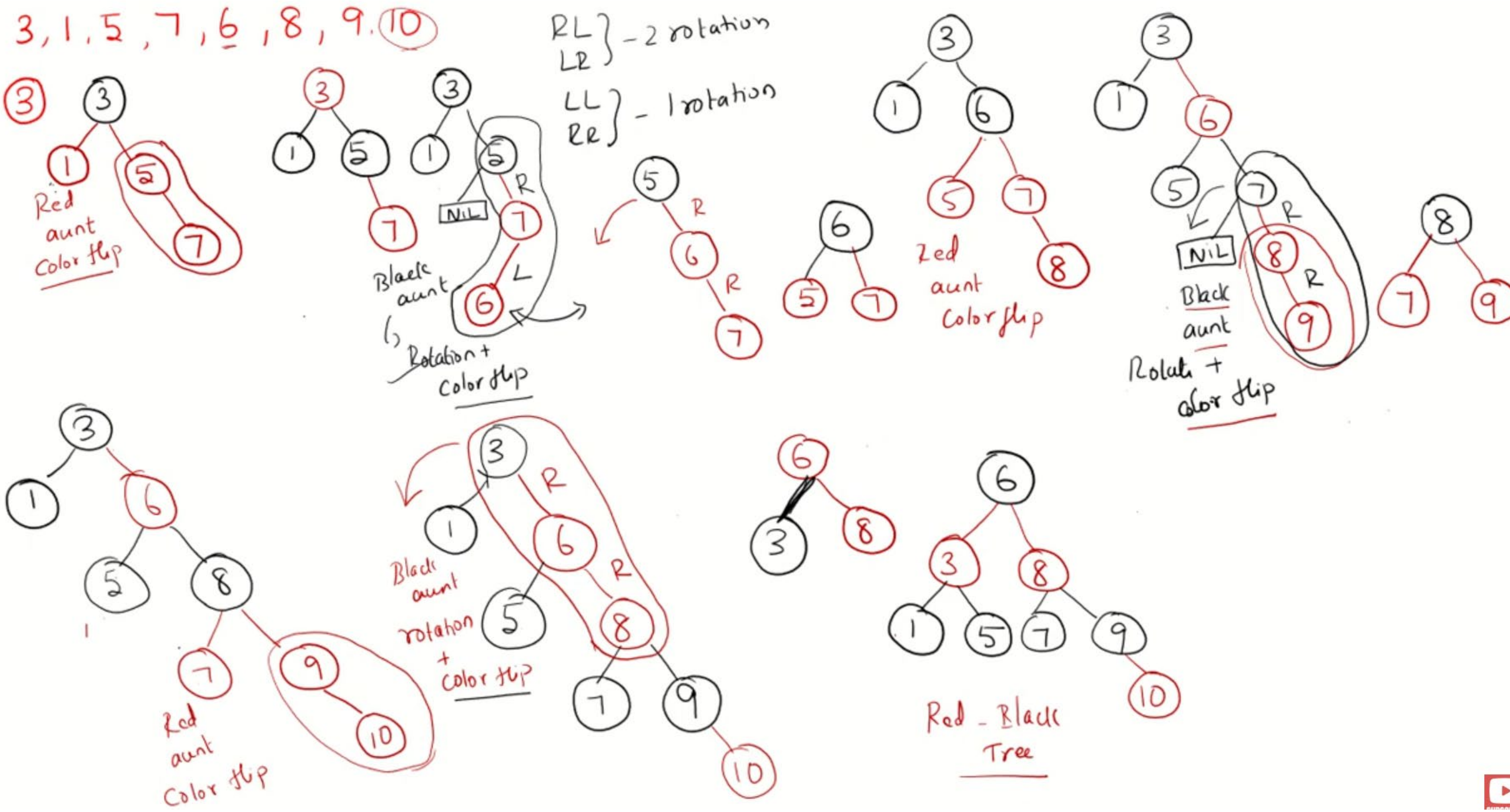
Example 2 Con’t



Case 2.
Z.uncle = black
left rotate on 8 &
recolor



Another Example



Red Black Tree – Insertion
https://www.youtube.com/watch?v=9ubIKipLpRU

Time Complexity

- 1. Insert : O(log(n))
  - maximum height of red-black trees
- 2. Color red : O(1)
- 3. Fix violations :
  - Constant # of:
  - a. Recolor : O(1)
  - b. Rotation: O(1)
- Overall time complexity: O(log(n))

Applications

- Red–black trees are widely used as system symbol tables.
  - Java: java.util.TreeMap, java.util.TreeSet.
  - C++ STL: map, multimap, multiset.
  - Linux kernel: completely fair scheduler, linux/rbtree.h.
  - Emacs: conservative stack scanning.

Video Tutorials

- Red-Black Trees // Michael Sambol
  - https://www.youtube.com/playlist?list=PL9xmBV_5YoZNqDI8qfOZgzbqahCUmUEin
  - Lecture slides based in this video series
- Red Black Tree – Insertion
  - https://www.youtube.com/watch?v=9ubIKipLpRU
- https://www.geeksforgeeks.org/introduction-to-red-black-tree/
  - https://www.geeksforgeeks.org/introduction-to-red-black-tree/