Red-Black Trees

Introduction

Introduction

- A **Red-Black Tree (RBT)** is a **self-balancing binary search tree**.
- Each node stores an extra bit for color, either **Red** or **Black**.
- **Red-Black Tree Properties:**
  - Every node is either red or black.
  - The root is always black.
  - Red nodes cannot have red children (no two red nodes can be adjacent).
  - Every path from a node to its descendant NULL nodes must have the same number of black nodes (called the **black-height**).
  - New insertions are always red nodes.

Structure

- **Root:** Black**Red**
- **Nodes:** Must have black children (no two red nodes can be adjacent).
- **Black-Height:** Every path from a node to its NULL descendant must have the same number of black nodes.

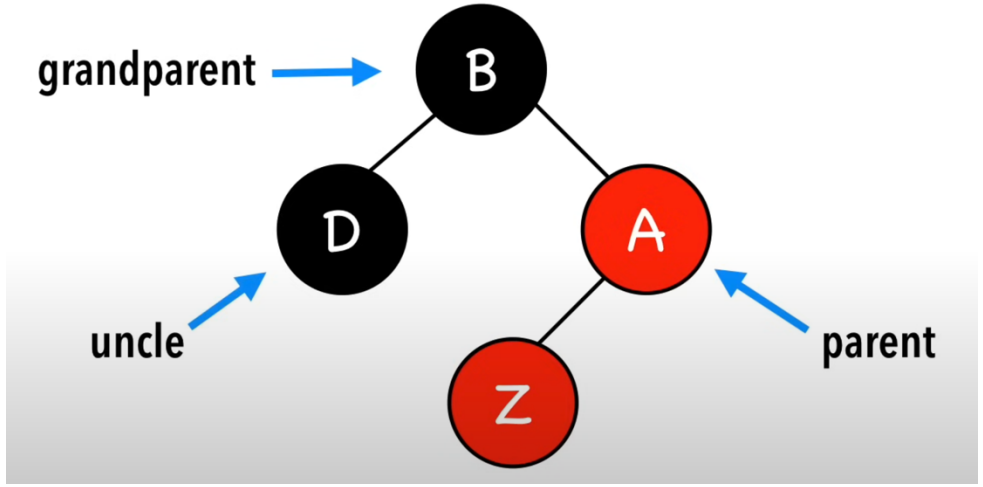
Properties in Summary

- **Root is black.**
- **Red nodes cannot have red children** (no two consecutive red nodes).
- **Every path from a node to a NULL node must contain the same number of black nodes.**
- **Leaf nodes (NULL nodes)** are always black.

Insertion in a Red-Black Tree

- **Insertion Steps:**
  - Insert the new node as a red node.
  - If the red node violates any Red-Black properties, perform **recoloring** or **rotations** (left or right).
  - Ensure the tree still satisfies all Red-Black properties after insertion.
- **Rotation Example:**
  - If the new node's **parent** is **red** and the **uncle** is also **red**, **recolor** the nodes.
  - If the new node's **parent** is **red** but the **uncle** is **black** (or NULL), perform **rotations**.

Insertion Details



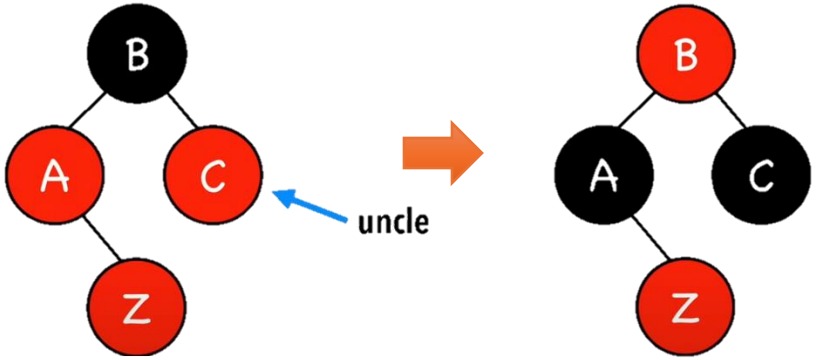
1. Insert Z and color it **red**
2. Recolor and rotate nodes to fix violation

4 scenarios

0. Z = root
1. Z.uncle = **red**
2. Z.uncle = **black** (triangle)
3. Z.uncle = **black** (line)

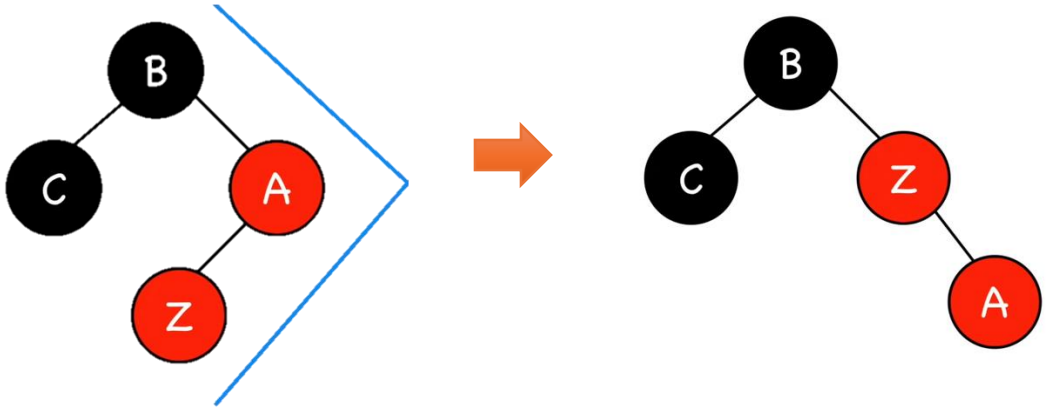
Insertion Details Cont.

Case 1 → Solution - Recolor



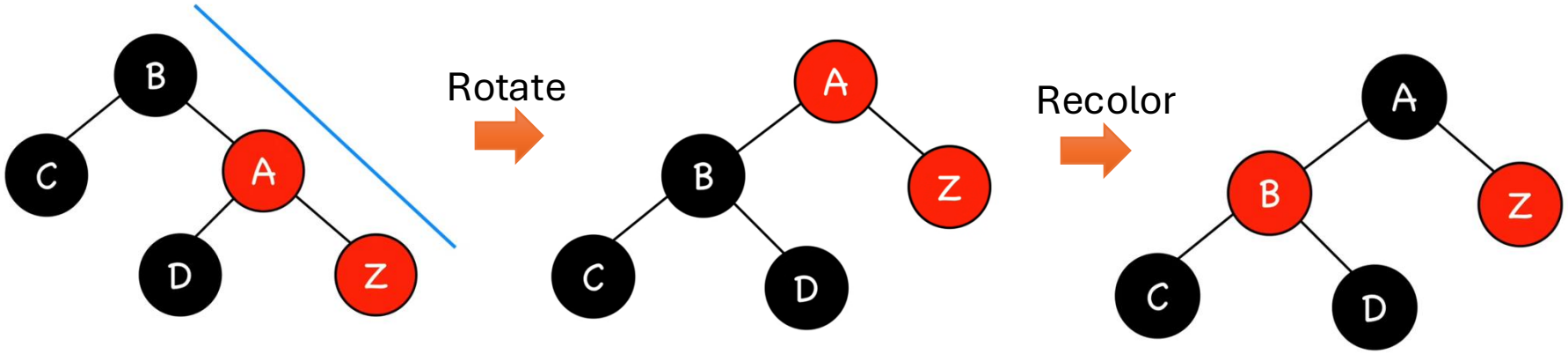
Case 2: Z.uncle = black (triangle)

→ Soulution – Rotate opposite direction of Z on Z.parent

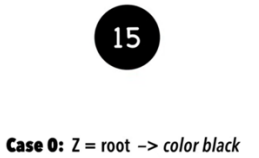


Case 3: Z.uncle = black (line)

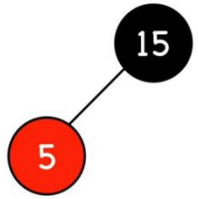
→ Soulution – Rotate opposite direction of Z on Z.grandparent, and recolor original parent and grandparent

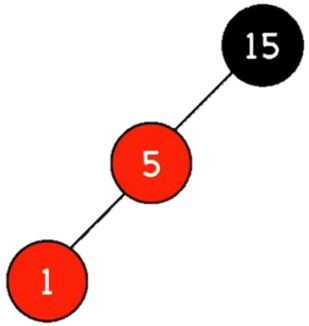


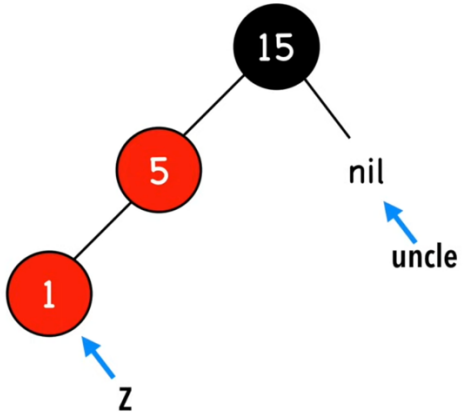
Example

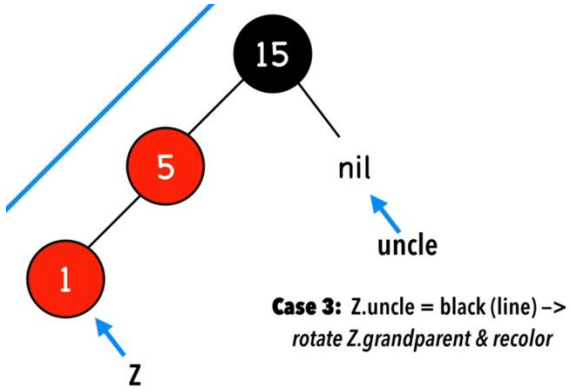


Case 0: Z = root –> *color black*

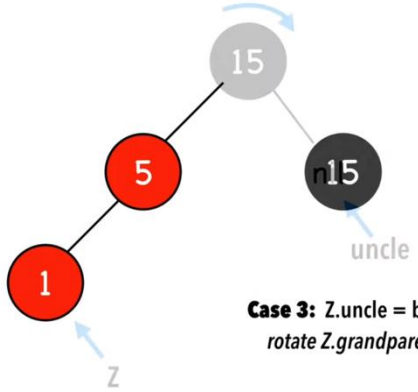




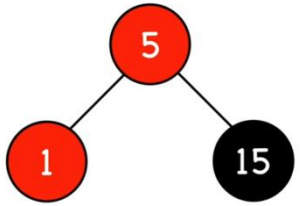


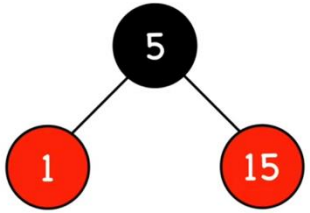


Case 3: Z.uncle = black (line) –>
rotate Z.grandparent & recolor

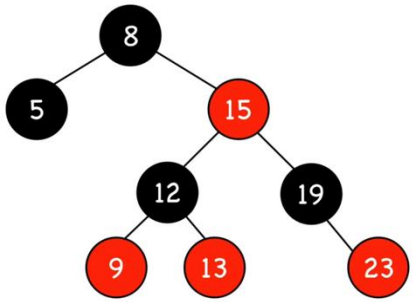


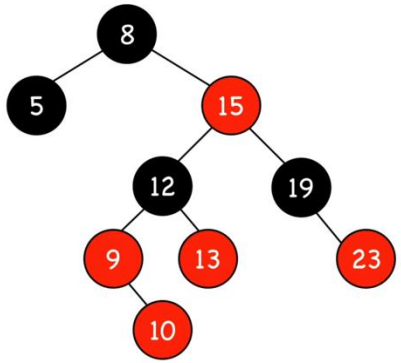
Case 3: Z.uncle = black (line) –>
rotate Z.grandparent & recolor

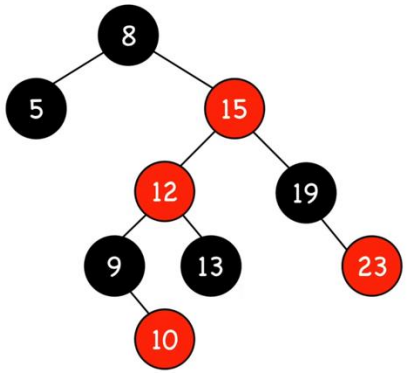


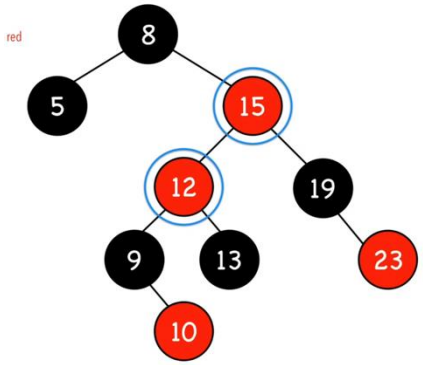


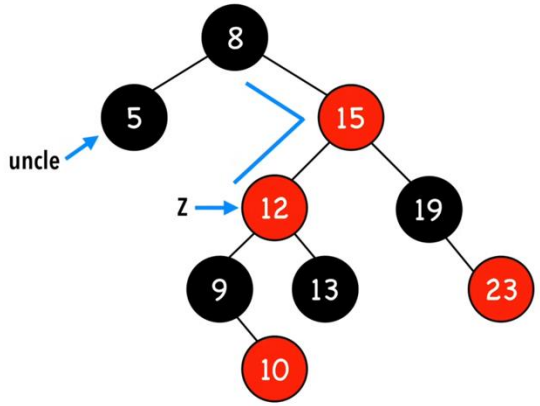
Example

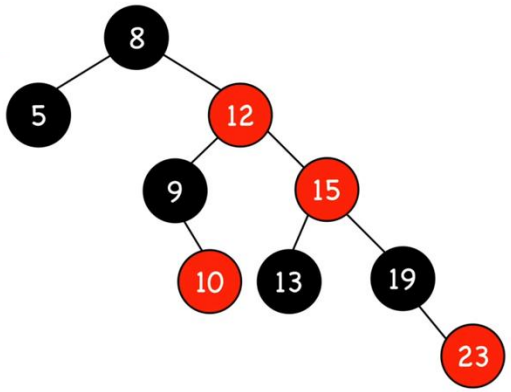

8
5
15
12
19
9
13
23

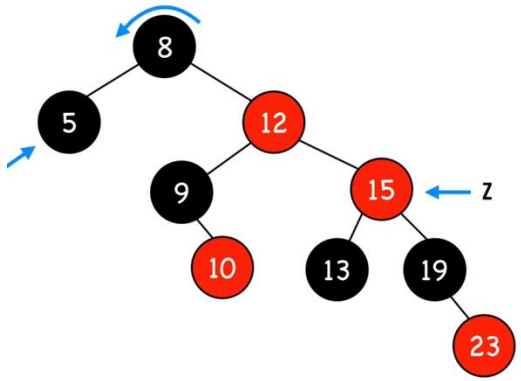

8
5
15
12
19
9
13
23
10

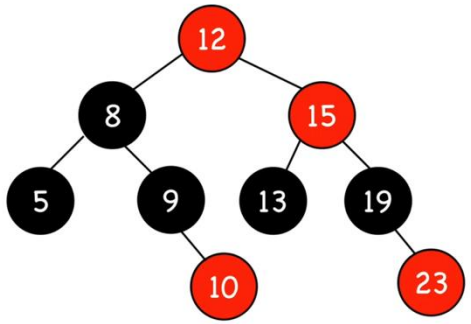

8
5
15
12
19
9
13
23
10

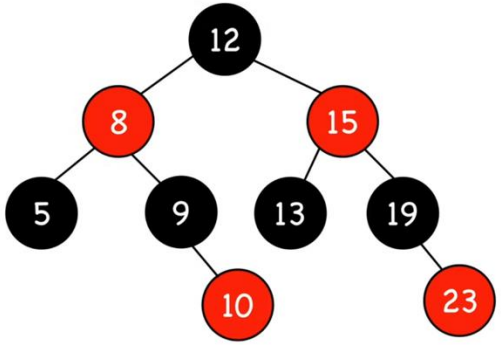

red
8
5
15
12
19
9
13
23
10


uncle
z
8
5
15
12
19
9
13
23
10


8
5
12
9
15
10
13
19
23


8
5
12
9
15
z
10
13
19
23


12
8
15
5
9
13
19
10
23


12
8
15
5
9
13
19
10
23

Time Complexity

- Insertion : O(log n)
- Search : O(log n)
- Deletion : O(log n)

Applications

- in the Java Collections Library
  - TreeSet
  - TreeMap
  - HashMap
- Linux
  - The Completely Fair Scheduler in the Linux kernel
  - mmap and munmap operations for file/memory mapping
- To implement finite maps and Standard Template Libraries (STL) in C++.
- MySQL also uses the Red-Black tree for indexes on tables.
- Use for geometric range searches, k-means clustering, and text-mining.