CH-231-A

Algorithms and Data Structures

ADS

Lecture 25

Dr. Kinga Lipskoch

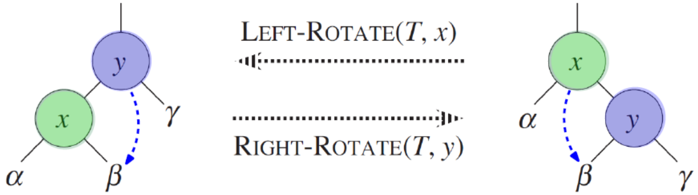
Spring 2022

Operations

- Querying
  - Search, Minimum & Maximum, Successor & Predecessor
  - Just as in normal BST
  - $O(\lg n)$
- Modifying
  - Tree-Insert, Tree-Delete $\rightarrow O(\lg n)$
  - But, need to guarantee red-black tree properties:
    - Must change color of some nodes
    - Change pointer structure through rotation

Rotations (1)

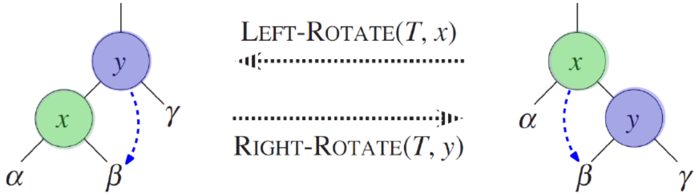
- *Right-Rotate*(T, y):
  - Node y becomes right child of its left child x
  - New left child of y is former right child of x
- *Left-Rotate*(T, x):
  - Node x becomes left child of its right child y
  - New right child of x is former left child of y



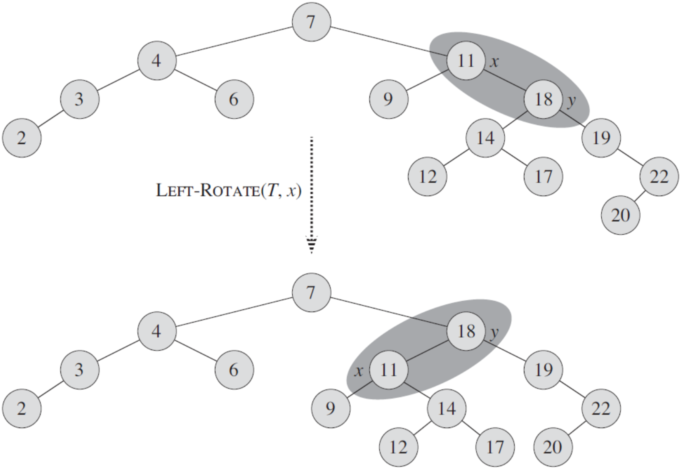
Rotations (2)

BST property is preserved:

- (left): $\mathit{key}(\alpha) \leq x.\mathit{key} \leq \mathit{key}(\beta) \leq y.\mathit{key} \leq \mathit{key}(\gamma)$
- (right): $\mathit{key}(\alpha) \leq x.\mathit{key} \leq \mathit{key}(\beta) \leq y.\mathit{key} \leq \mathit{key}(\gamma)$



Rotation: Example



Rotation Pseudocode

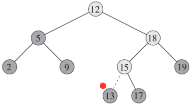
```
LEFT-ROTATE(T, x)
y = x.right                      // set y
x.right = y.left                 // turn y's left subtree into x's right subtree
if y.left ≠ T.nil
    y.left.p = x
y.p = x.p                        // link x's parent to y
if x.p == T.nil
    T.root = y
elseif x == x.p.left
    x.p.left = y
else x.p.right = y
y.left = x                       // put x on y's left
x.p = y
```

Time complexity: O(1)

Insertion

TREE-INSERT(T, z)

```
y = NIL
x = T.root
while x ≠ NIL
    y = x
    if z.key < x.key
        x = x.left
    else x = x.right
z.p = y
if y == NIL
    T.root = z
elseif z.key < y.key
    y.left = z
else y.right = z
```



RB-INSERT(T, z)

```
y = T.nil
x = T.root
while x ≠ T.nil
    y = x
    if z.key < x.key
        x = x.left
    else x = x.right
z.p = y
if y == T.nil
    T.root = z
elseif z.key < y.key
    y.left = z
else y.right = z
z.left = T.nil
z.right = T.nil
z.color = RED
RB-INSERT-FIXUP(T, z)
```

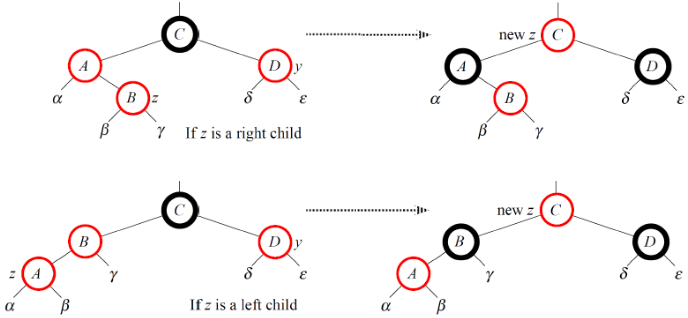
Fixing Red-Black Tree Properties

- We are inserting a **red** node to a valid red-black tree.
- Which properties may be violated?
  1. Duh: Cannot be violated. ✓
  2. RooB: Violated if inserted node is root. ✗
  3. LeaB: Inserted node is not a leaf, i.e., no violation. ✓
  4. BredB: Violated if parent of inserted node is red. ✗
  5. BH: Not affected by red nodes, i.e., no violation. ✓

Fixing BredB

- BredB for node z is violated, if $z.p$ is red.
- Then, $z.p.p$ is black. (BredB property)
- We need to consider different cases depending on the uncle y of z, i.e., the child of $z.p.p$ that is not $z.p$.
- There are 6 cases:
  - $z.p$ is left child of $z.p.p$
    - y is red (Case 1)
    - y is black
      - z is right child of $z.p$ (Case 2)
      - z is left child of $z.p$ (Case 3)
  - $z.p$ is right child of $z.p.p$
    - y is red (symmetric to Case 1)
    - y is black
      - z is right child of $z.p$ (symmetric to Case 3)
      - z is left child of $z.p$ (symmetric to Case 2)

Case 1 (Red Uncle)



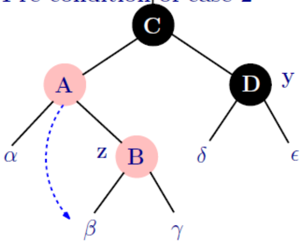

```
if z.p == z.p.p.left
    y = z.p.p.right
    if y.color == RED
        z.p.color = BLACK
        y.color = BLACK
        z.p.p.color = RED
        z = z.p.p
```

Case 1

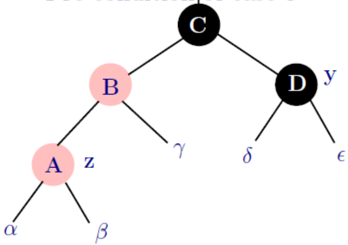
There is a new problem, if *z.p.p.p* is red.
Algorithm needs to continue with *z.p.p*.

Case 2 (Black Uncle, z Right Child)

Pre-condition of case 2

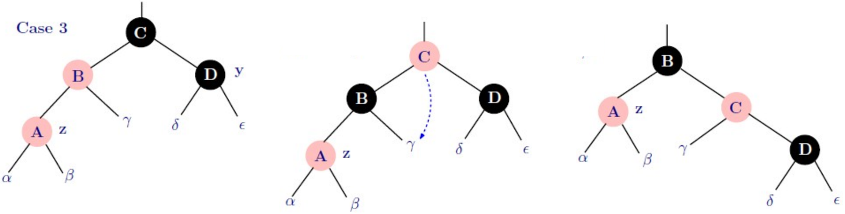


Pre-condition of case 3




```
else if z == z.p.right
    z = z.p                  Case 2
    LEFT-ROTATE(T, z)
```

Case 3 (Black Uncle, z Left Child)




```
z.p.color = BLACK
z.p.p.color = RED
RIGHT-ROTATE(T, z.p.p)
```

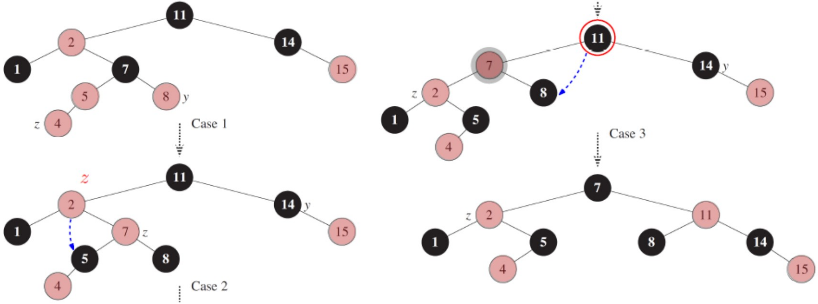
Case 3

Putting It All Together

- We need to put the 3 cases (and the 3 symmetric cases) together
- Moreover, we need to propagate the considerations upwards (see Case 1)
- Finally, we have to fix RooB

```
RB-INSERT-FIXUP(T, z)
while z.p.color == RED
    if z.p == z.p.p.left
        y = z.p.p.right
        if y.color == RED
            z.p.color = BLACK           Case 1
            y.color = BLACK             Case 1
            z.p.p.color = RED           Case 1
            z = z.p.p                   Case 1
        else if z == z.p.right
                z = z.p                 Case 2
                LEFT-ROTATE(T, z)       Case 2
            z.p.color = BLACK           Case 3
            z.p.p.color = RED           Case 3
            RIGHT-ROTATE(T, z.p.p)      Case 3
    else (same as then clause
            with "right" and "left" exchanged)
T.root.color = BLACK
```

Insert Example



Time Complexity

- In worst case, we have to go all the way from the leaf to the root along the longest path within the tree
- Hence, running time is $O(h) = O(\lg n)$ for the fixing of the red-black tree properties
- Overall, running time for insertion is $O(h) = O(\lg n)$
- Example for building up a red-black tree by iterated node insertion:
  http://www.youtube.com/watch?v=vDHFF4wjWYU