Red-Black Trees

A Tree

- Why Trees? O(logN) operations
- A tree is a collection of nodes:
  - The collection can be empty;
  - Otherwise consists of,
  - a distinguished node r, called the root
  - zero or more (sub)trees T_1, T_2, …, T_k, each of whose roots are connected by a directed edge from r.
  - The root of each subtree is said to be a child of r, and r is the parent of each subtree root.

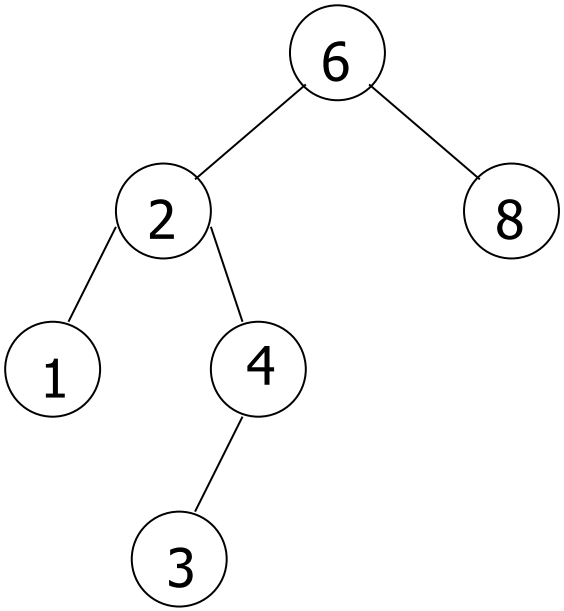
Concepts on Trees

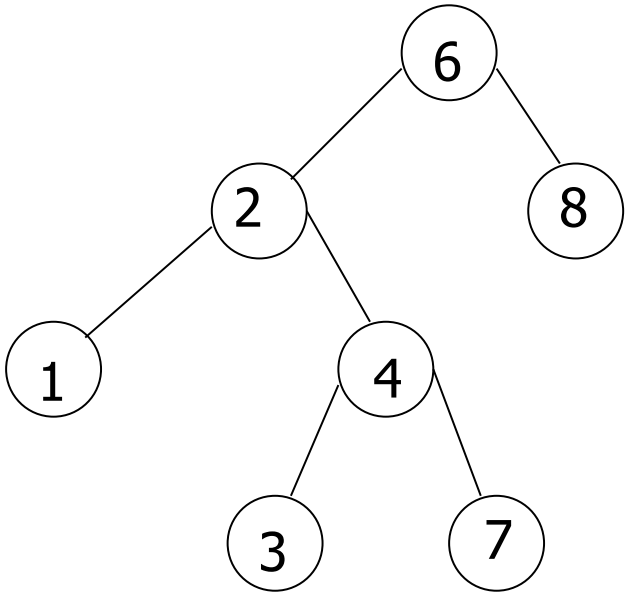
- The root is the only node without parent.
- Nodes with no children are known as leaves.
- Nodes with the same parent are siblings.
- A path from node n_1 to n_k is defined as a sequence of node $n_1, n_2, \ldots, n_k$ such that n_i is the parent of n_{i+1} for $1 \leq i < k$.
- The length of the path is the number of edges on the path
- There is a path of length zero from every node to itself.
- In a tree there is exactly one path from the root to each node.

Binary Search Trees

- Assumptions:
  - each node in the tree stores an item.
  - those items are integers.
  - all the items are distinct.
- The properties:
  - For every node, X, in the tree, the values of all the items in its left subtree are smaller than the item in X, and the values of all the items in its right subtree are larger than the item in X.

Which one is a Binary Search Tree?





Operations on a Binary Search Tree

- Contains(int x): Is x an element of BST
- FindMax() find the maximum values
- FindMin(): find the minimum value
- Adding and deleting elements
- Keep the tree balanced

Find Minimum Element

- private BinaryNode<AnyType> findMin (BinaryNode<AnyType>)
- {
  - if (t==null)
    - return null;
  - else
    - if (t.left==null)
    - return t;
  - return findMin(t.left);
- }

Search for An Element

- private boolean contains (AnyType x, BinaryNode<AnyType>) {
- If(t==null) return false;
- int compareresults=myCompare(x, t.element)
- If(compareResult <0)
  - return contains(x,t.left)
- Else if (compareResut >0)
  - return contains(x,t.right)
- Else return true}

Insert An Element

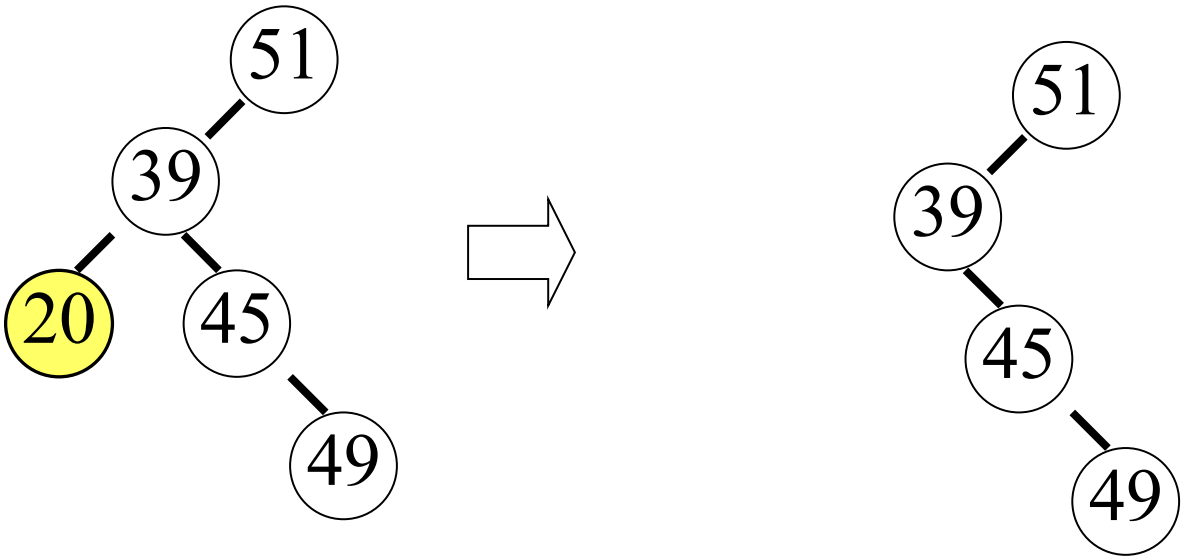
- private boolean contains (AnyType x, BinaryNode<AnyType>) {
- If(t==null) return new BinaryNode;
- int compareresults=myCompare(x, t.element)
- If(compareResult <0)
  - t.left= insert(x,t.left)
- Else if (compareResut >0)
  - t.right= insert(x,t.right)
- Else Do nothing}

Delete

- The node to be deleted has three conditions
  - The node is a leaf
  - The node has one child
  - The node has two children

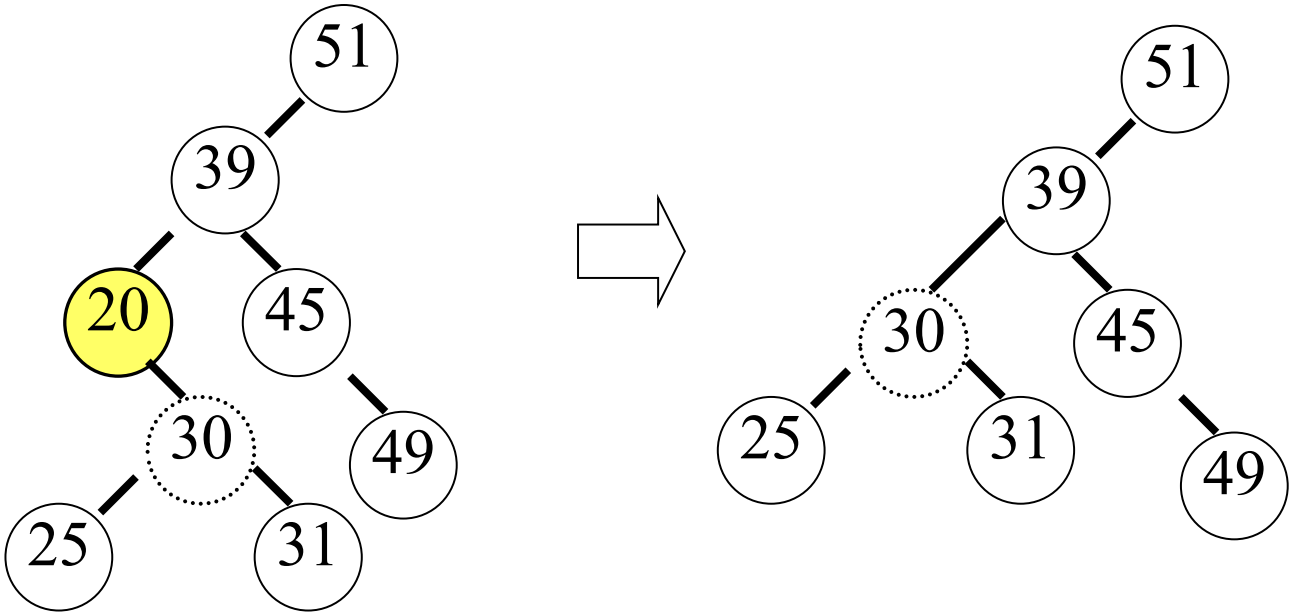
How to Remove a Node?

- What if the node is a leaf?
  - It can be deleted immediately.



How to Remove a Node?

- What if the node has one child?
  - It can be deleted after its parent adjusts a pointer to bypass the node

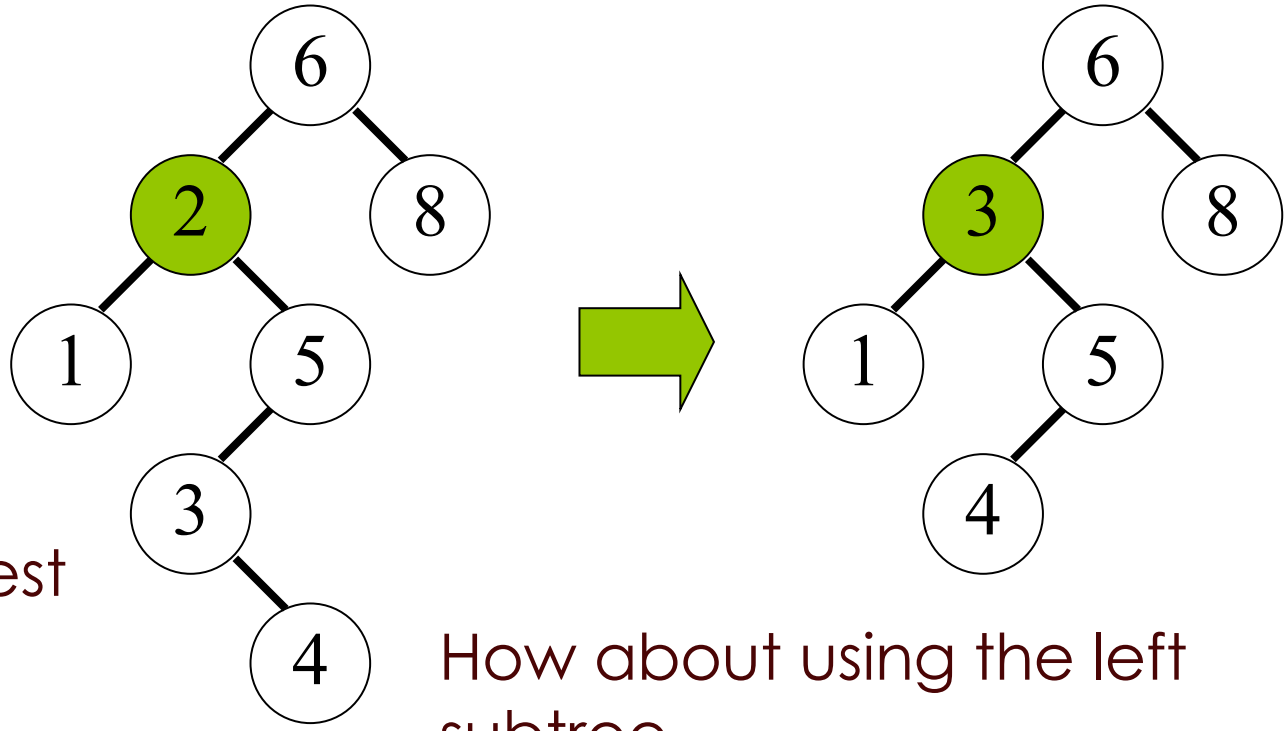


- What if the node has two children?
  - The general strategy is to replace the data of this node with the smallest data of the right subtree, and recursively delete the node.

Find smallest data in right tree

Replace this value in the node

Delete smallest data



How about using the left subtree

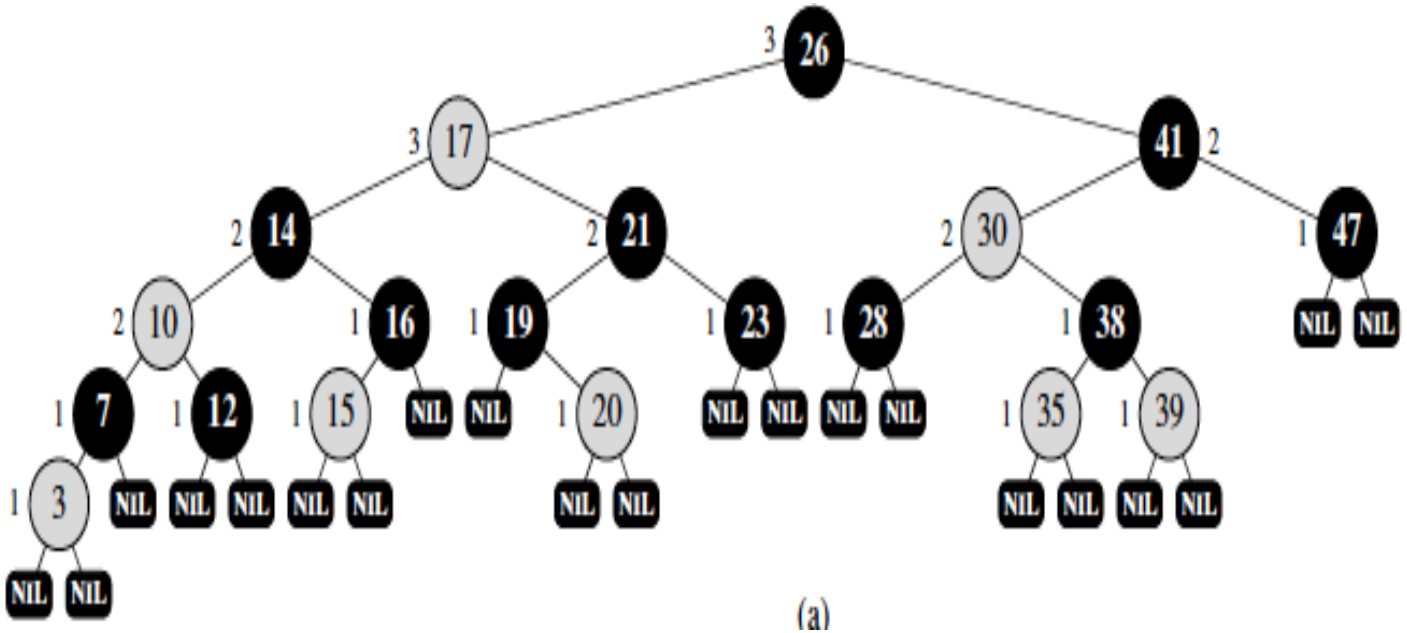
Balanced Trees

- Best complexity for addition/deletion and search in binary trees is O(logn).
- This is true only if the tree is balanced---nearly equal height on each side.
- Different types of balanced trees proposed
  - AVL, B-Trees, Red-Black Trees

Red-Black Trees

- A binary search tree where…
- Every node is either red or black
- The root is black
- Every leaf (NIL) is black
- If a node is red then both its children are black
- For each node, all paths from the node to descendant leaves contain the same number of black nodes
- Black height of a node is the number of black nodes from the node to the leaf. The node itself is not counted in the path

Example



(a)

Black nodes colored black
Red nodes colored red

Height of Red Black Tree

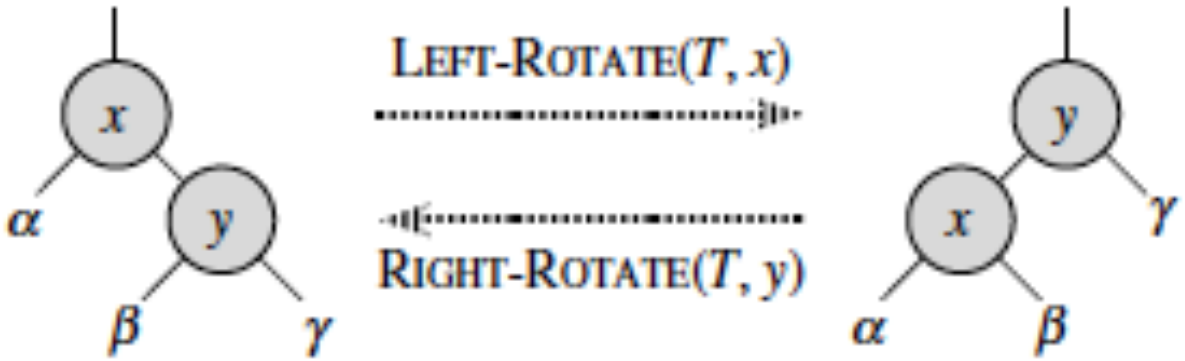
- A R_B tree with n internal nodes has height at most 2(log n+1)
  - The subtree rooted at node x contains at least $2^{bh(x)}-1$ internal nodes. Proof by Induction
  - Let the height of the tree be h
  - At least half the nodes from the root to leaf (not counting the root) are black
  - Thus the black height of the tree must be at least h/2

Height of Red-Black tree

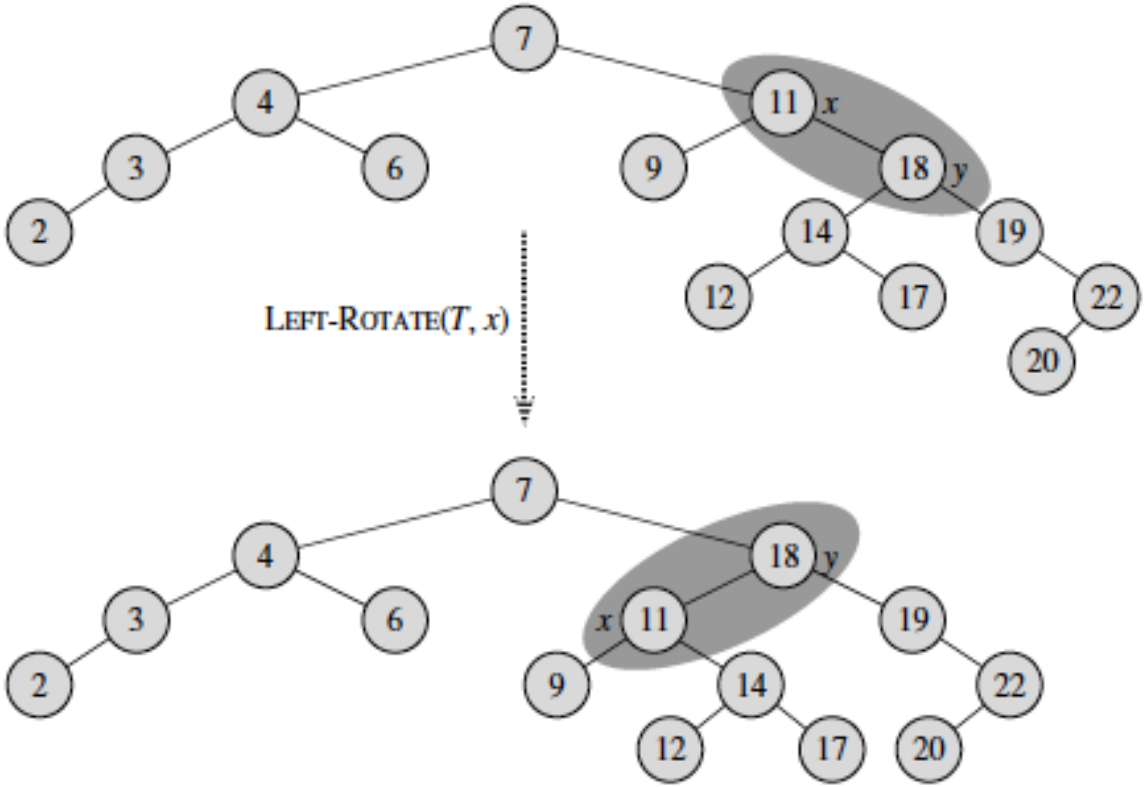
- $N >= 2^{h/2} - 1$
- $h <= 2\lg(n+1)$
- Thus search, finding maximum, finding minimum can all be done in O(lgn)
- Insertion and Deletion can also be run in O(lgn)

Rotation

- Need to rotate newly inserted nodes to preserve red-black property



Rotation Example



Insertion

- First find the position of the node in the R-B Tree
- Color new node red

- Adjust the tree to maintain R-B property

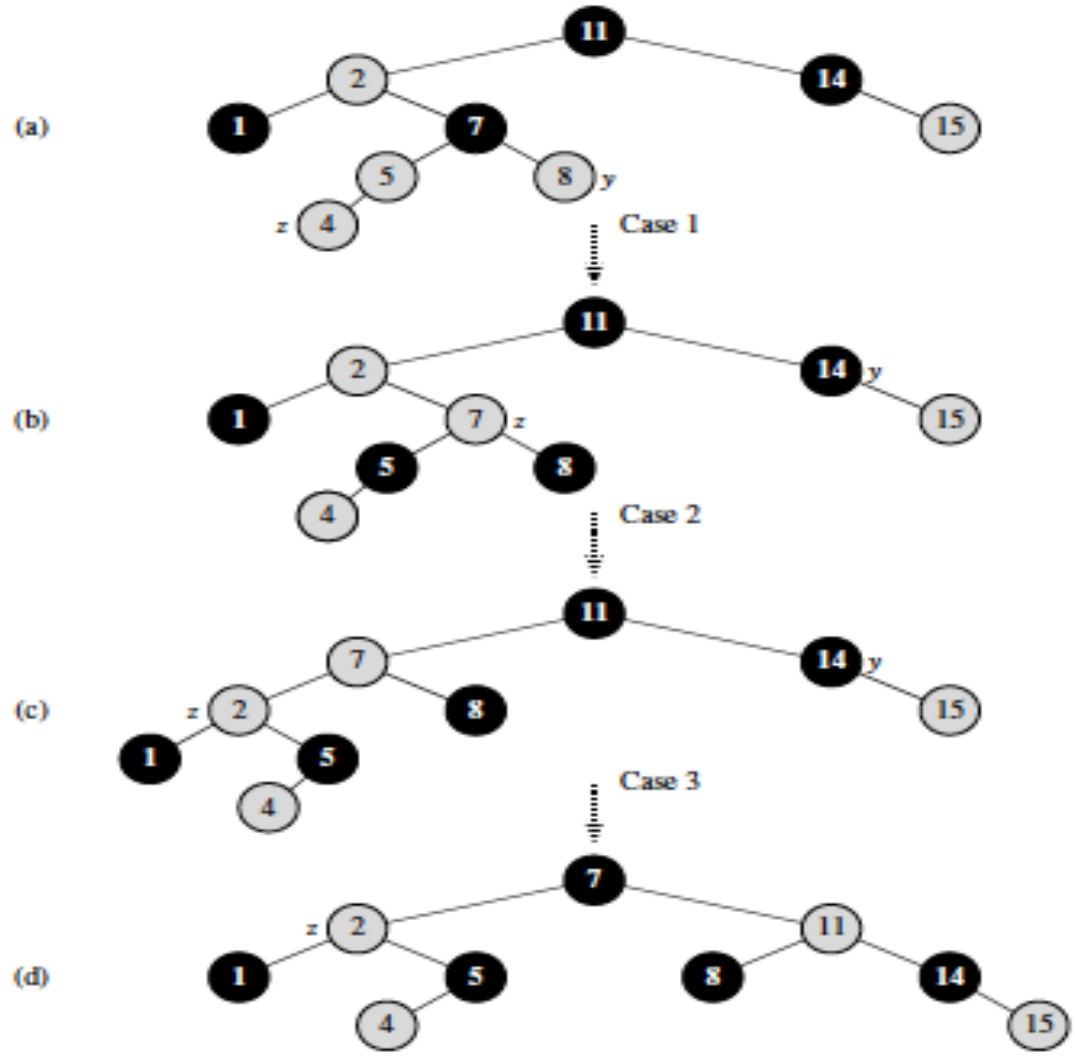
Tree Adjusting

- z is the instance where a red node has a red parent
- If parent is root then make parent black

- Case 1: Parent and uncle of z also red
  - Make grandparent red
  - Make parent and uncle black

- Case 2: Uncle is black
  - Color parent black
  - Color grandparent red
  - Do rotation

- Complexity of Insertion is O(lgn)—why ?

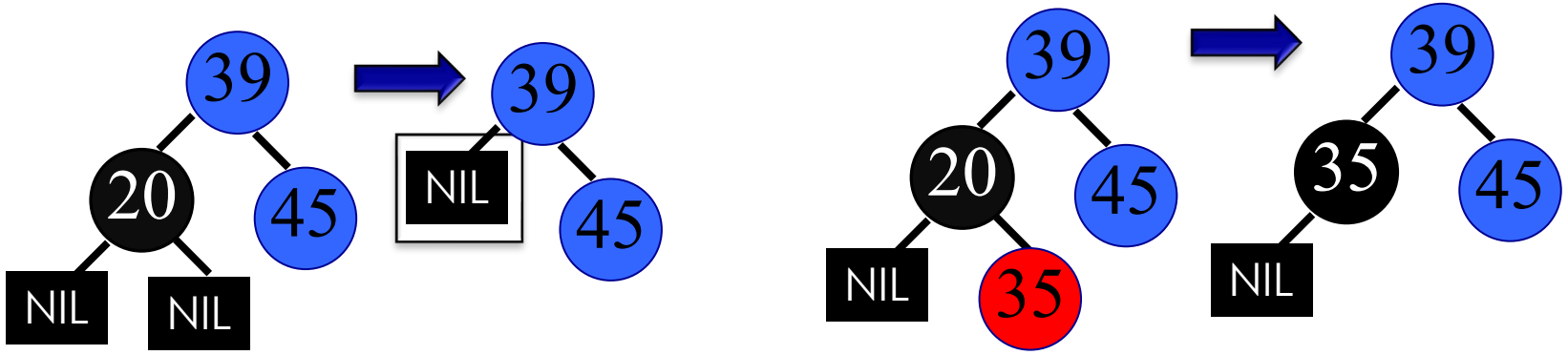

(a)
11
2
14
1
7
15
5
8
y
z
4
Case 1
(b)
11
2
14
y
1
7
z
15
5
8
4
Case 2
(c)
11
7
14
y
z
2
8
15
1
5
4
Case 3
(d)
7
z
2
11
1
5
8
14
4
15

Deletion

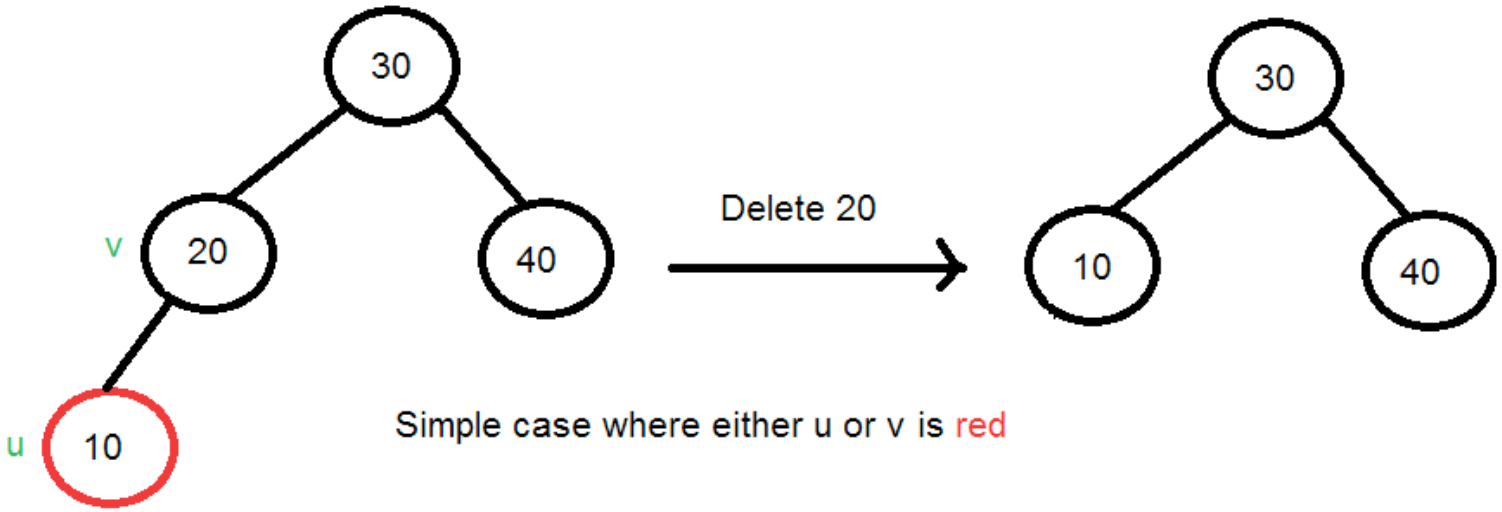
- Similar to deleting in binary search trees
- Except---
- Leaves are NIL nodes, so we do not delete at leaves, and adjust NIL nodes accordingly
- If we delete a red node—no problem
- If we delete a black node, then we will reduce the number of black nodes in a path. Have to fix it

Deletion—Double Black Node

- Recall that you are deleting either a node with NILs or node with one child
- The NIL node will be a double black
- Has to stand for two blacks to keep black height same in all paths
- Fix by adjusting the black nodes



Case 1
Either child or parent is red

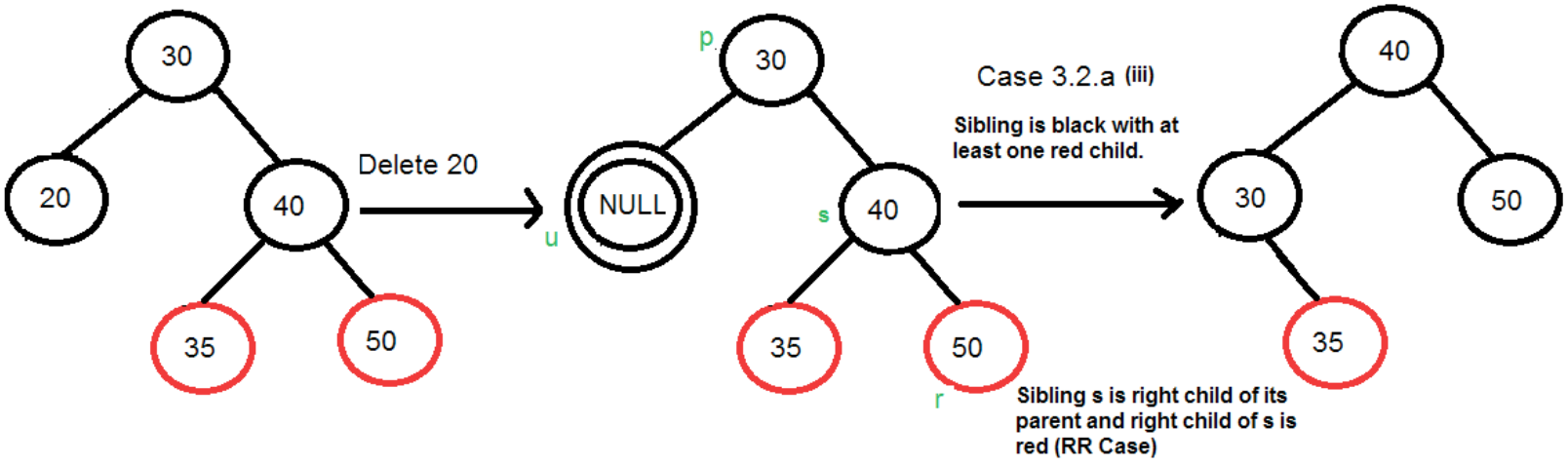


Images from :http://www.geeksforgeeks.org/red-black-tree-set-3-delete-2/

Case 2:

Node and parent are black. Sibling of node is black . At least one child of sibling is red

Rotate sibling branch

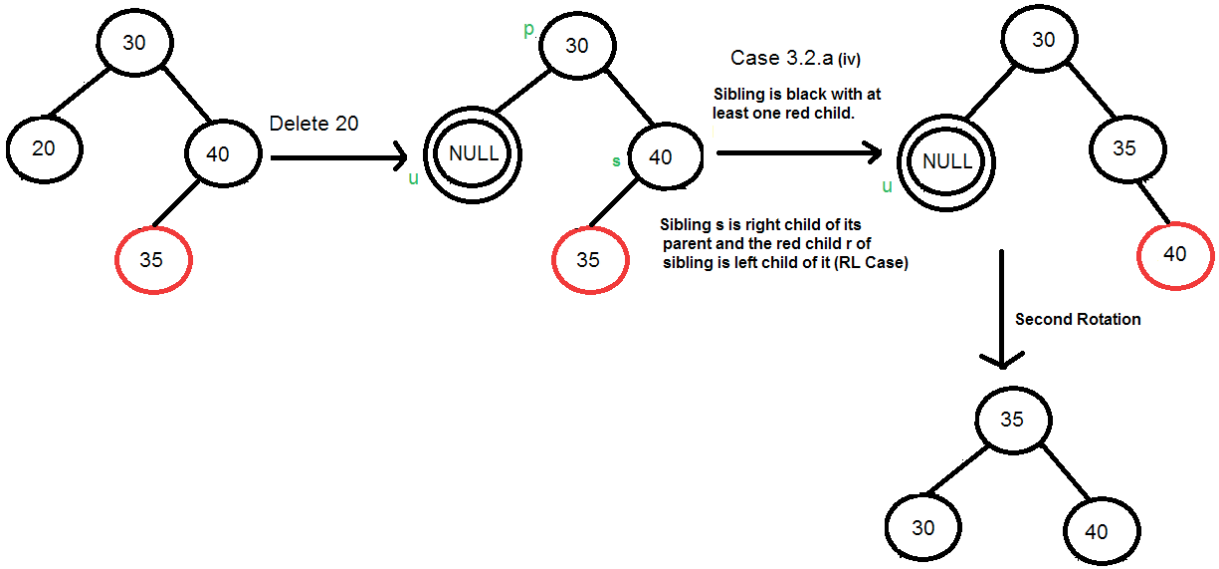


Example with right child

Case 2:

Node and parent are black. Sibling of node is black . At least one child of sibling is red

Rotate sibling branch

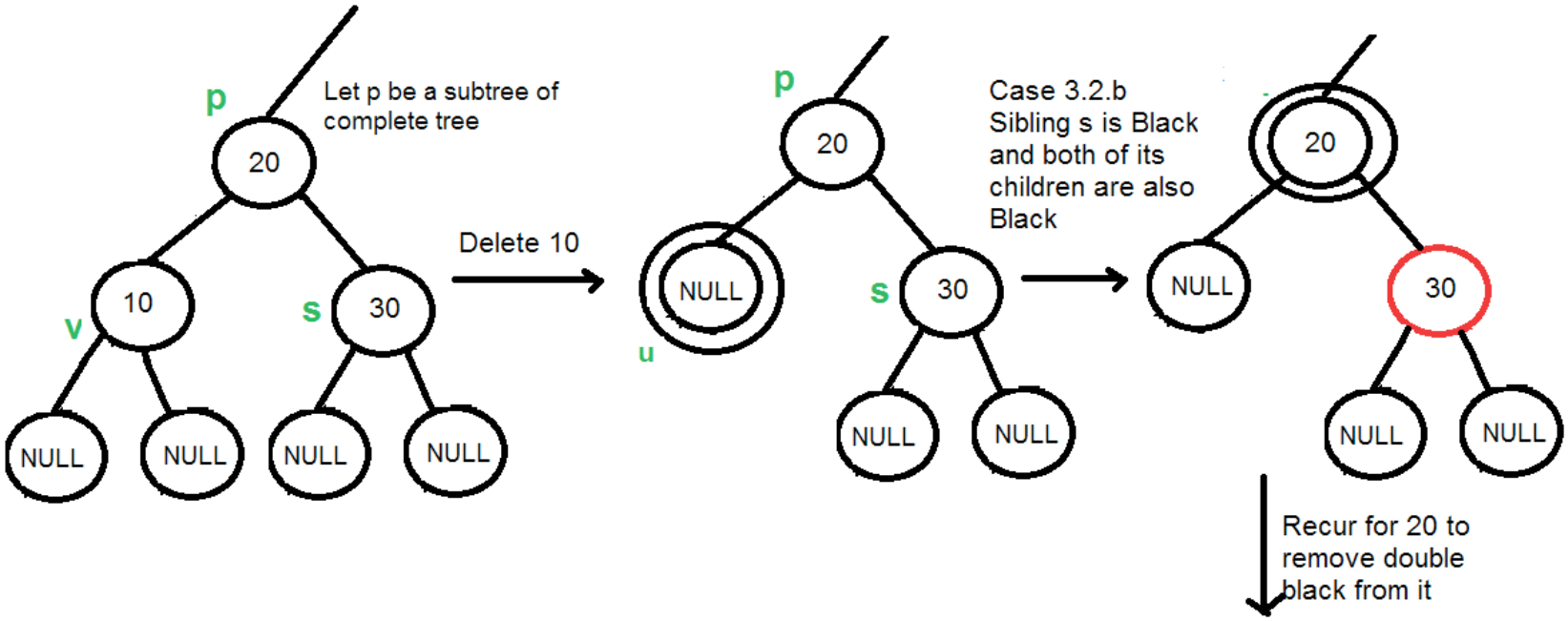


Example with left child

Case 3:

Node and parent are black. Sibling of node is black .
Both children are black.

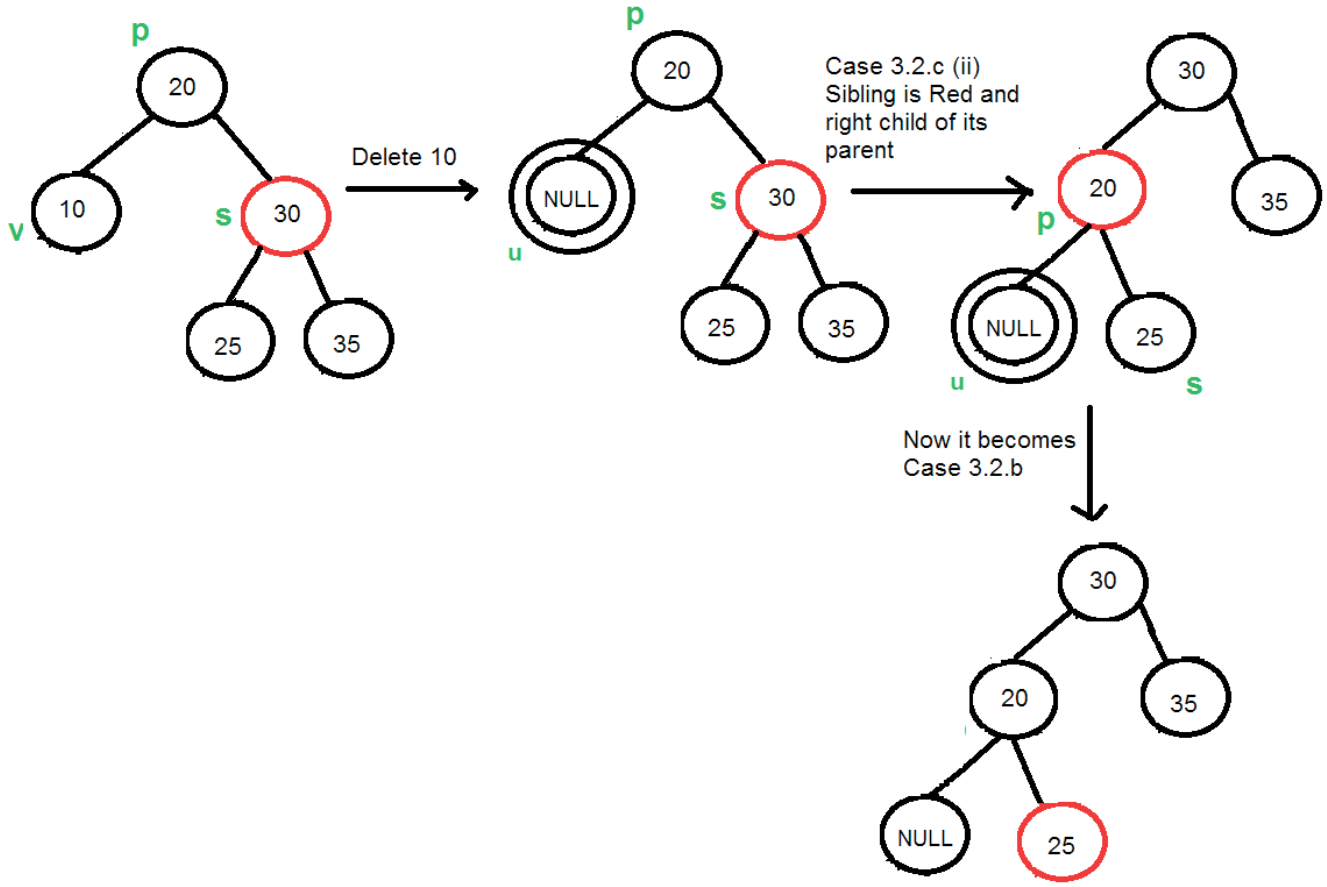
Move double black upto parent—and continue



Case 4:

Node and parent are black. Sibling of node is red.

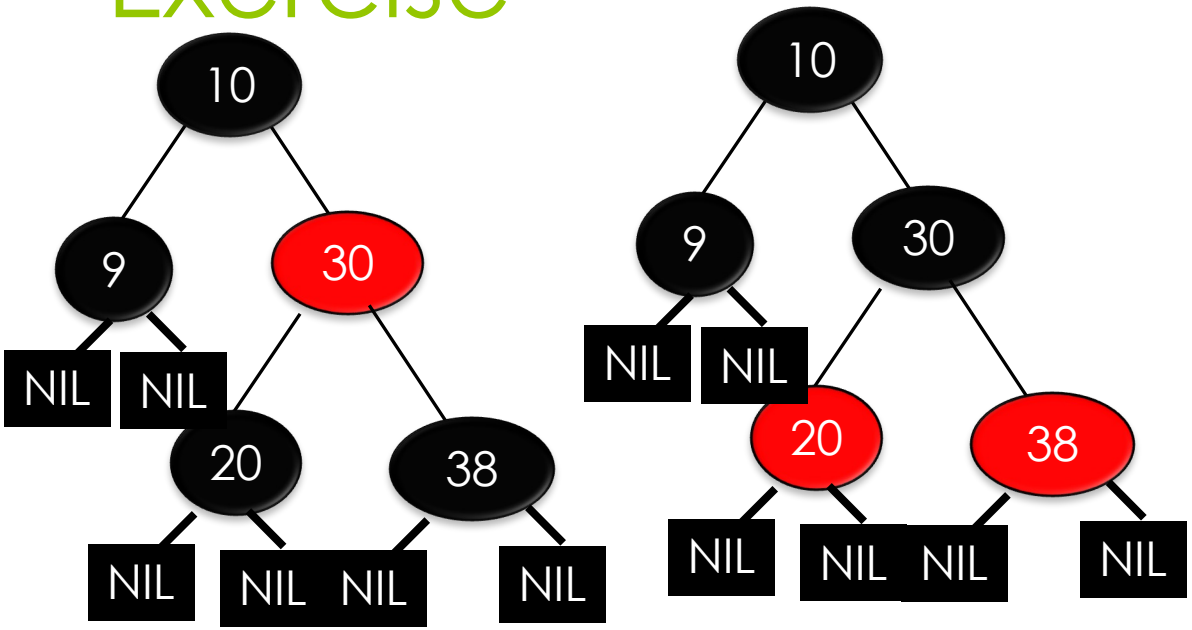
Rotate to move sibling up. Change color of sibling and parent.



Summary of Insertion and Deletion Rules

- If problem at root-color root black
- Insertion can cause change in black height, rotate tree to maintain black height after each insertion
- Deletion will cause double black nodes
  - Find appropriate red node (parent / sibling/ child of sibling) to make it black

Exercise



Delete 9 in all the tree
Delete 10 in all the trees