CSE 230: Data Structures

Lecture 11 :Search Trees

Ritwik M

Based on the reference materials by Prof. Goodrich and Dr. Vidhya Balasubramanian

Search Trees

- Different search trees
  - Binary Search Trees
  - AVL Trees
  - Multi-way search Trees
  - (2,4) Trees
  - Red – Black Trees

Binary Search Trees

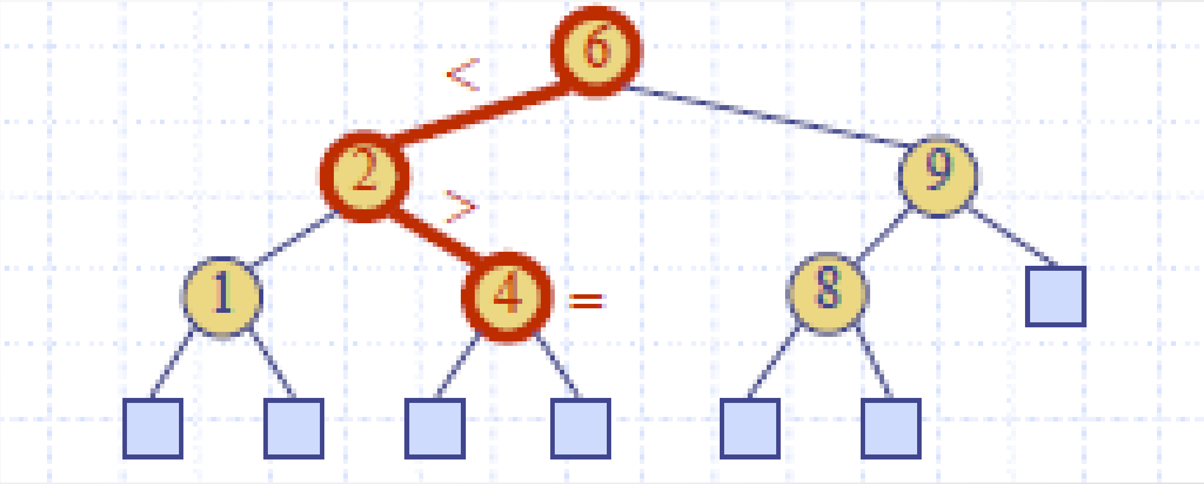
- It is a binary tree storing keys (or key-element pairs) at its nodes and satisfying the following properties:
  - The left subtree of a node contains only nodes with keys less than the node's key
  - The right subtree of a node contains only nodes with keys greater than the node's key
    - Let u, v, and w be three nodes such that u is in the left subtree of v and w is in the right subtree of v. key(u) ≤ key(v) ≤ key(w)
  - Both the left and right subtrees must also be binary search trees
  - Values are stored only in internal nodes (in the text book)
- Also called ordered or sorted binary tree
- Task: draw one such binary search tree!

Binary Search Trees

- Binary trees are very efficient for sorting and searching
- Fundamental data structure used to construct more abstract data structures
  - e.g sets, multisets, and associative arrays

Searching

- Can be recursive or iterative
- Start by examining the root and traverse
- If the key is less than the root, search the left subtree else search the right subtree
- Repeat until the key is found or remaining subtree is null
- Complexity :O(h)



Src: Goodrich notes

Searching: Iterative Algorithm

Algorithm find(k, root):

```
    curnode ← root
    while curnode is not None:
        if curnode.key == k:
            return curnode
        else if k < curnode.key:
            curnode ← curnode.left
        else
            curnode ← curnode.right
```

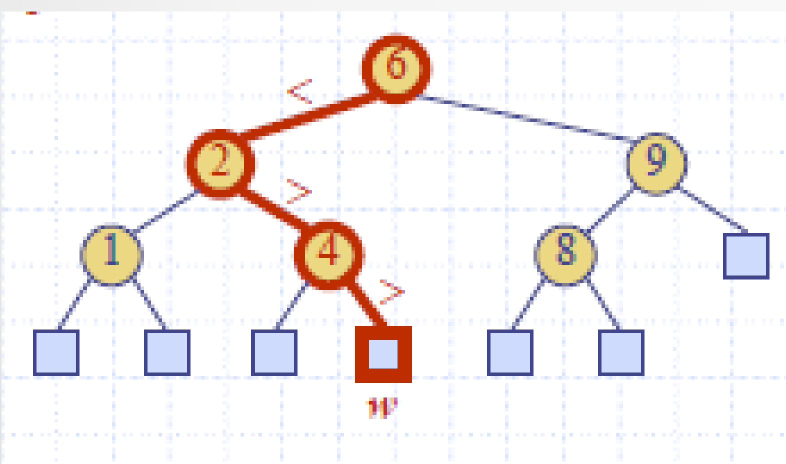
Searching: Recursive Algorithm

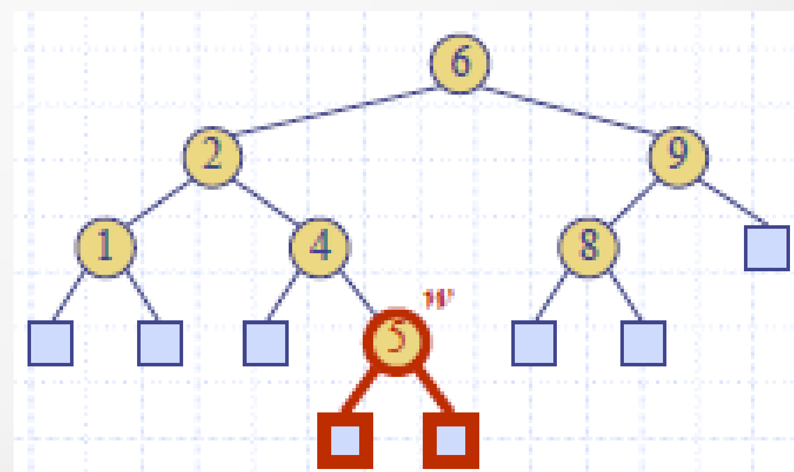
Algorithm find-recursive(k, node): //initially call with node = root

```
if node.key == k:
    return node
else if k < node.key:
    find-recursive(k, node.left)
else
    find-recursive(k, node.right)
```

Insertion

- insertItem(k,n) inserts a node with key k, into the tree with root node n
- Assume k is not already in the tree, and let let w be the leaf reached by the search
- We insert k at node w or add it as a child of w
  - Depending on the relative value it is a left child or right child





Insertion

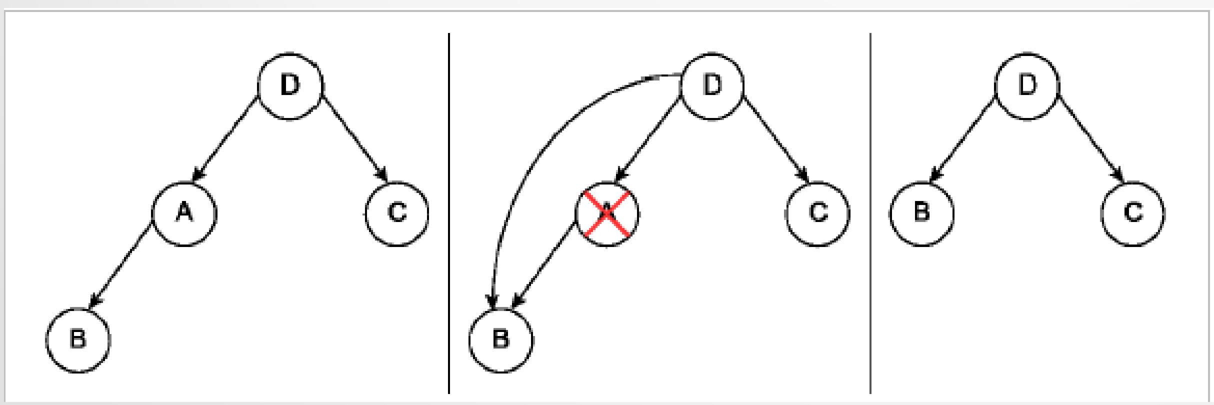
```
Procedure InsertItem(k,n) :
    if (k < n.key):
        if (n.left == null):
            n.left = Node(k)
        else:
            InsertItem(k,n.left)
        else if (k > n.key):
            if (n.right == null):
                n.right = Node(k)
            else:
                InsertItem(k,n.right)
```

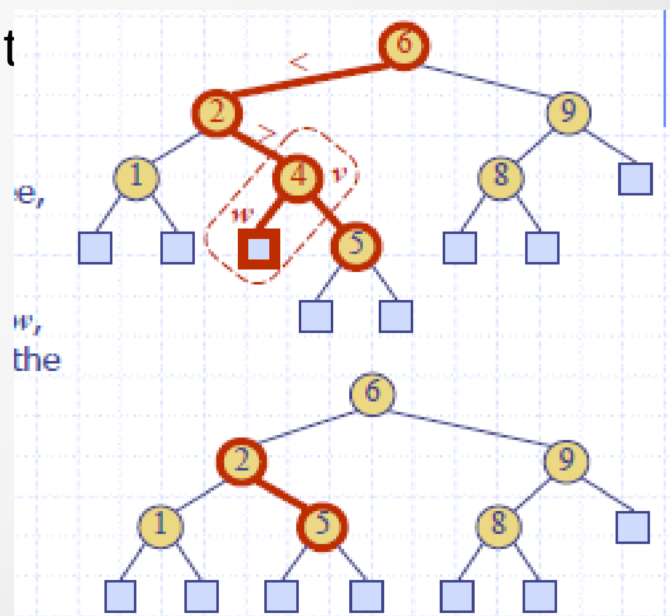
Deletion

- Three cases
  - Deleting a leaf or external node
    - Just remove the node
  - Deleting a node with one child
    - Remove the node and replace it with its child
  - Deleting a node with two children
    - Instead of deleting the node replace with its
      - inorder successor node
      - Inorder predecessor node

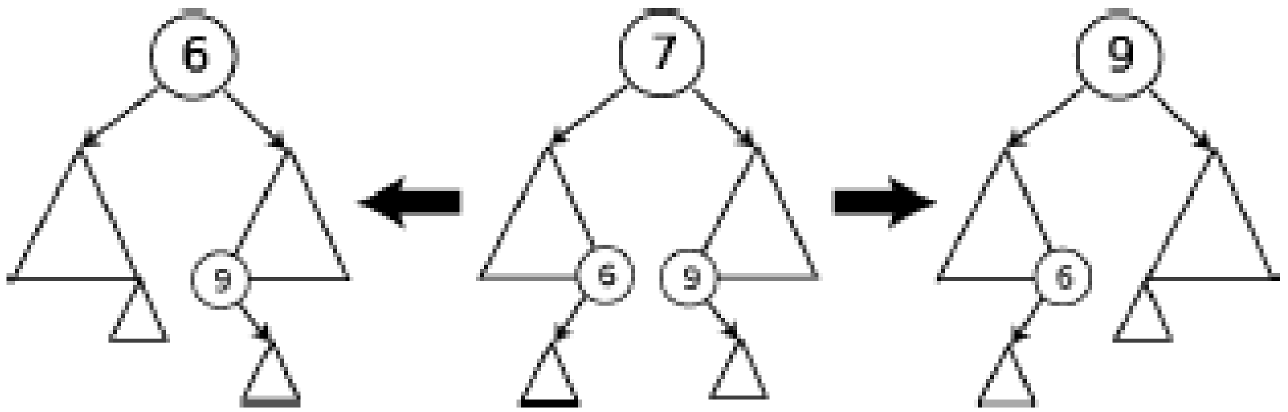
Deleting node with one child

- removeElement(k)
  - First find the node n with key k using the search method
    - Remove using removeAboveExternal(n.child)
      - set the parent of n's child to n's parent
      - set the child of n's parent to n's child





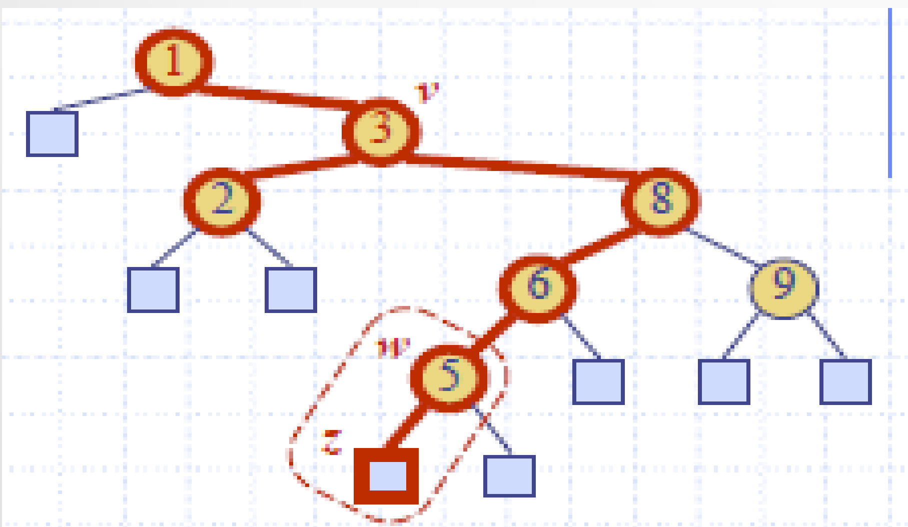
Deleting a node with two children

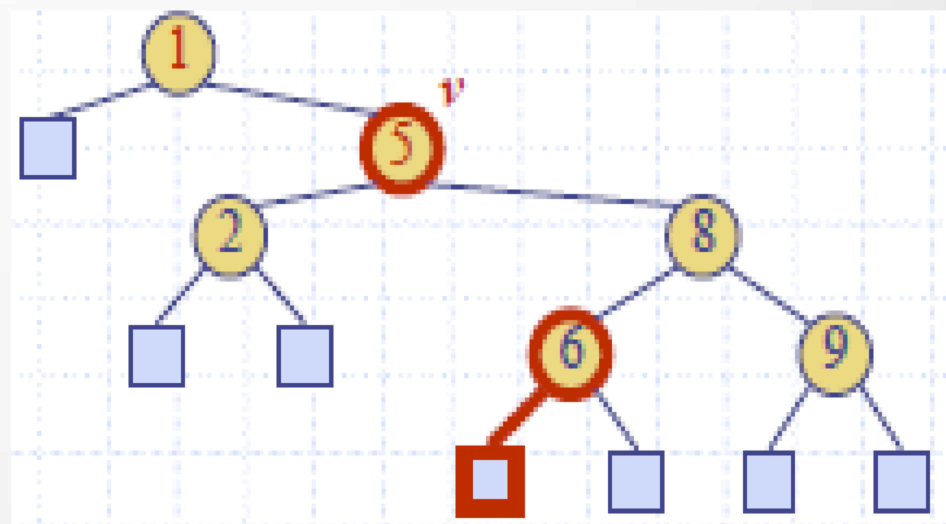


Deleting a node with two children from a binary search tree. The triangles represent subtrees of arbitrary size, each with its leftmost and rightmost child nodes at the bottom two vertices.

Deleting a node with two children

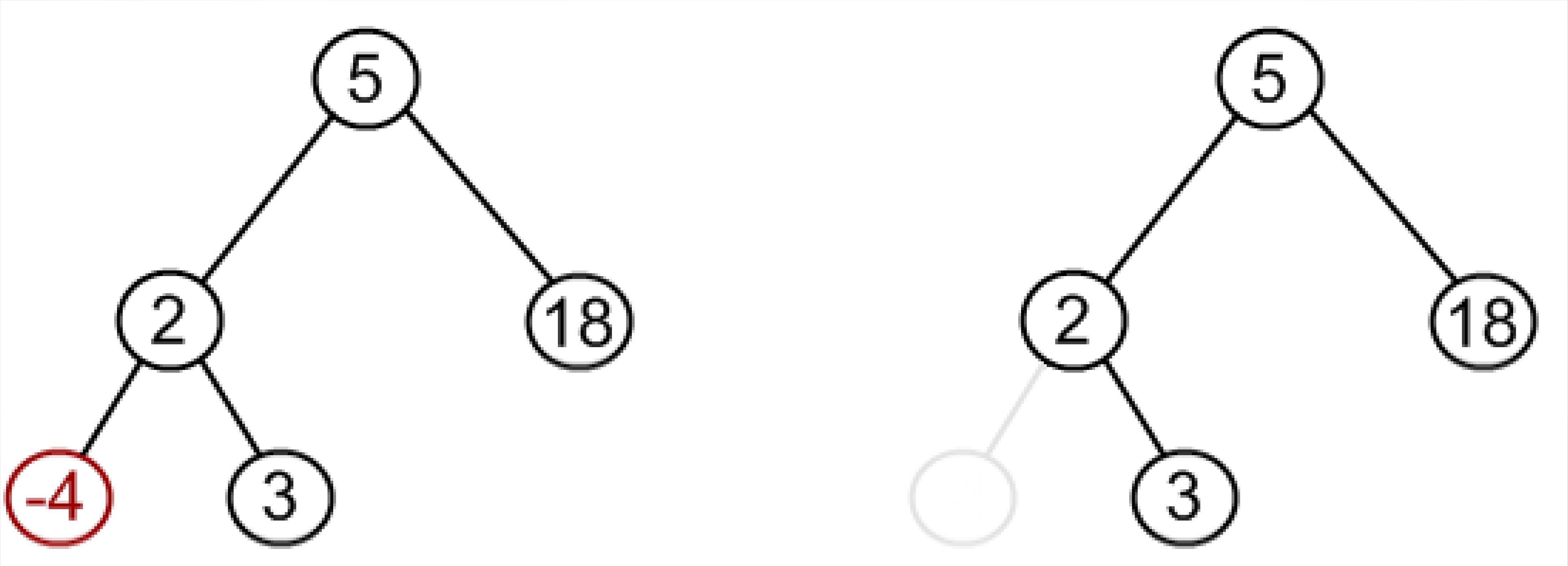
- find the node w that follows v in an inorder traversal
- copy key(w) into node v
- we remove node w and its left child z
  - Using the removeAboveExternal(z) method





Example: Case 1

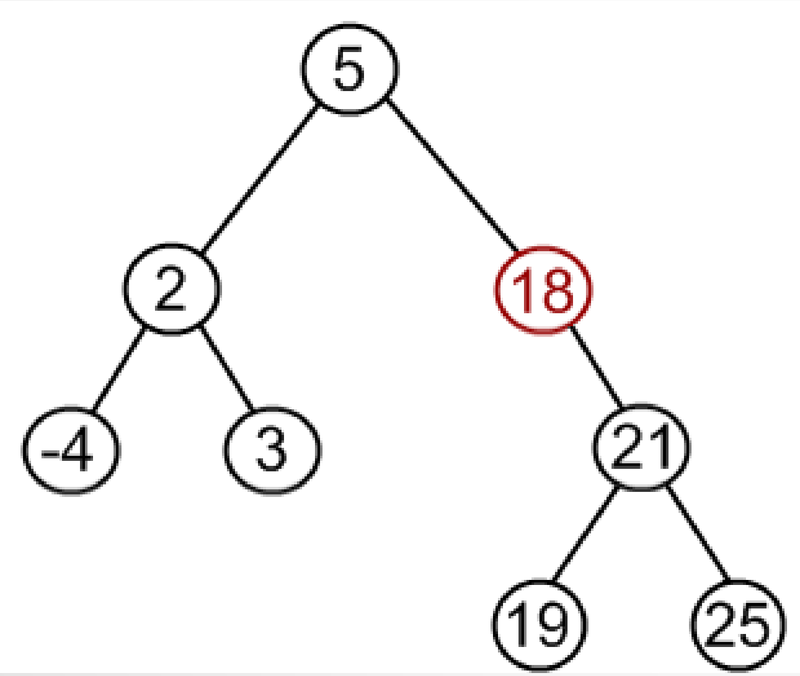
- Remove -4 from the BST

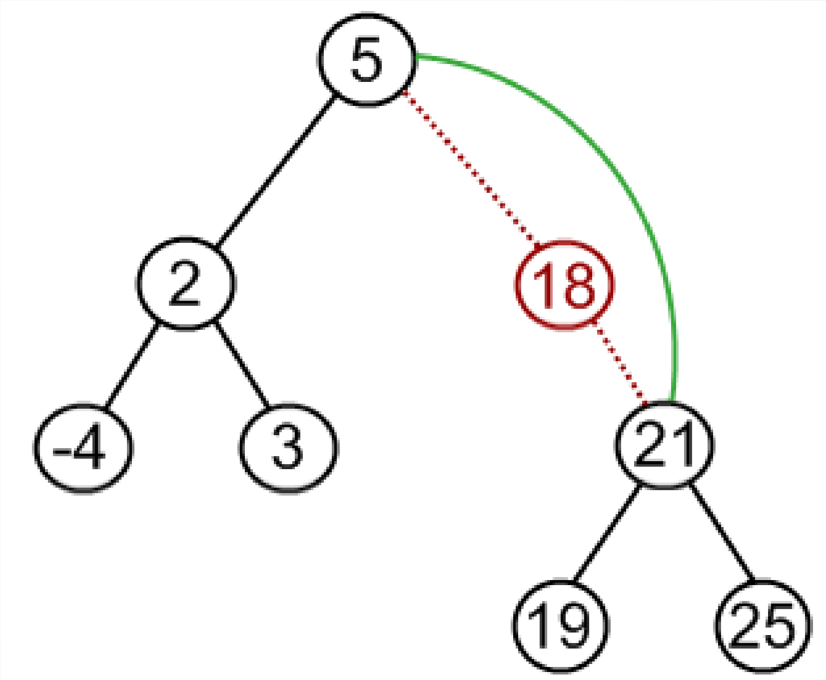


Source: http://www.algolist.net/Data_structures/Binary_search_tree/Removal

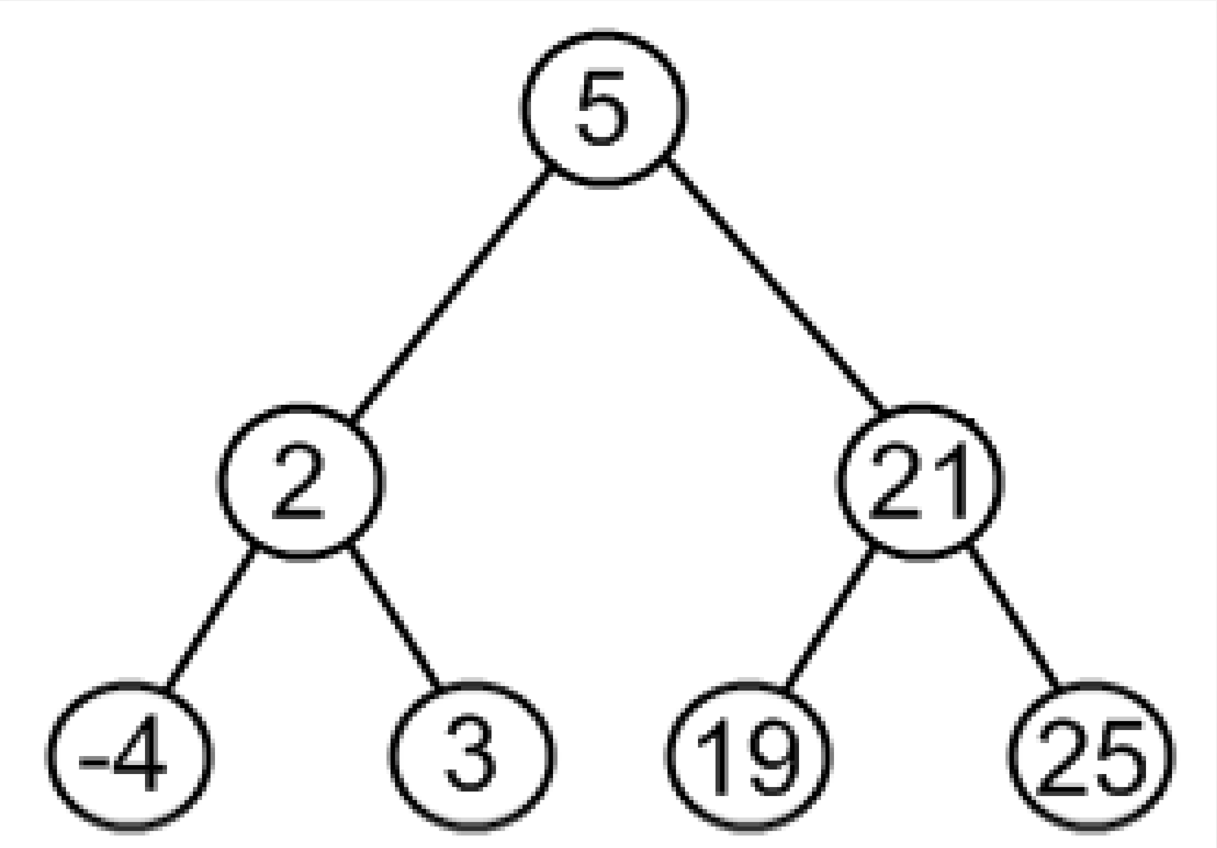
Example: Case 2

- Remove 18 from a BST



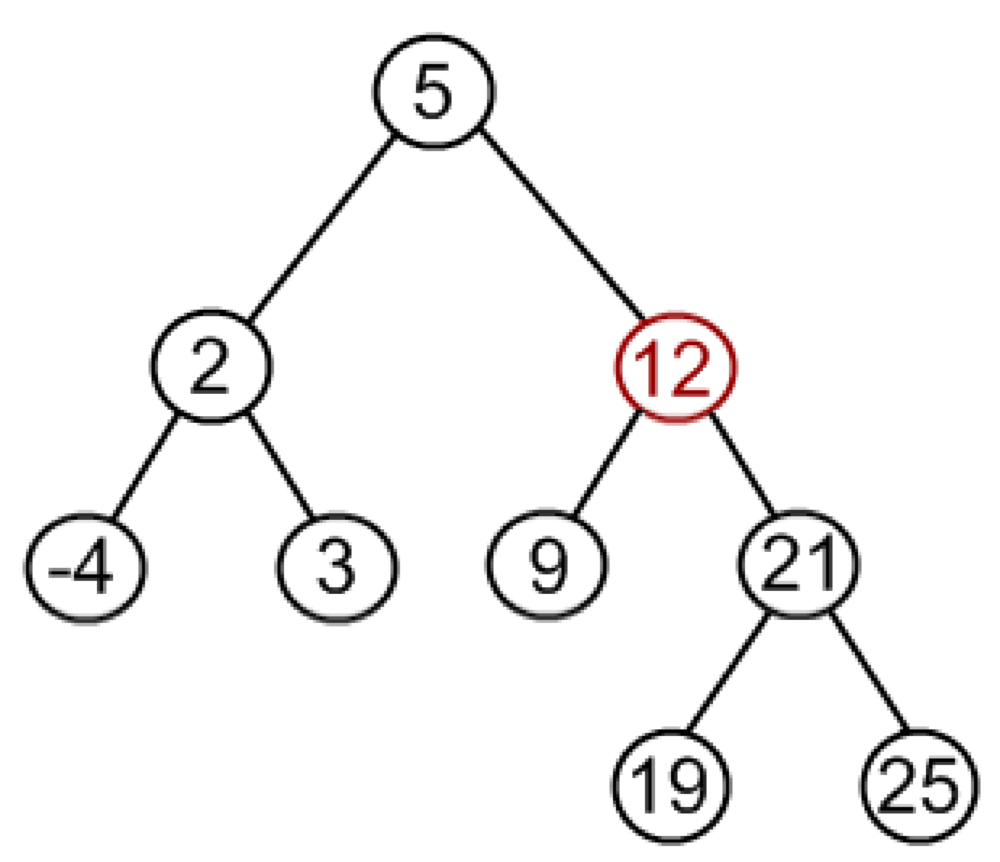


Example: Case 2



Example: Case 3

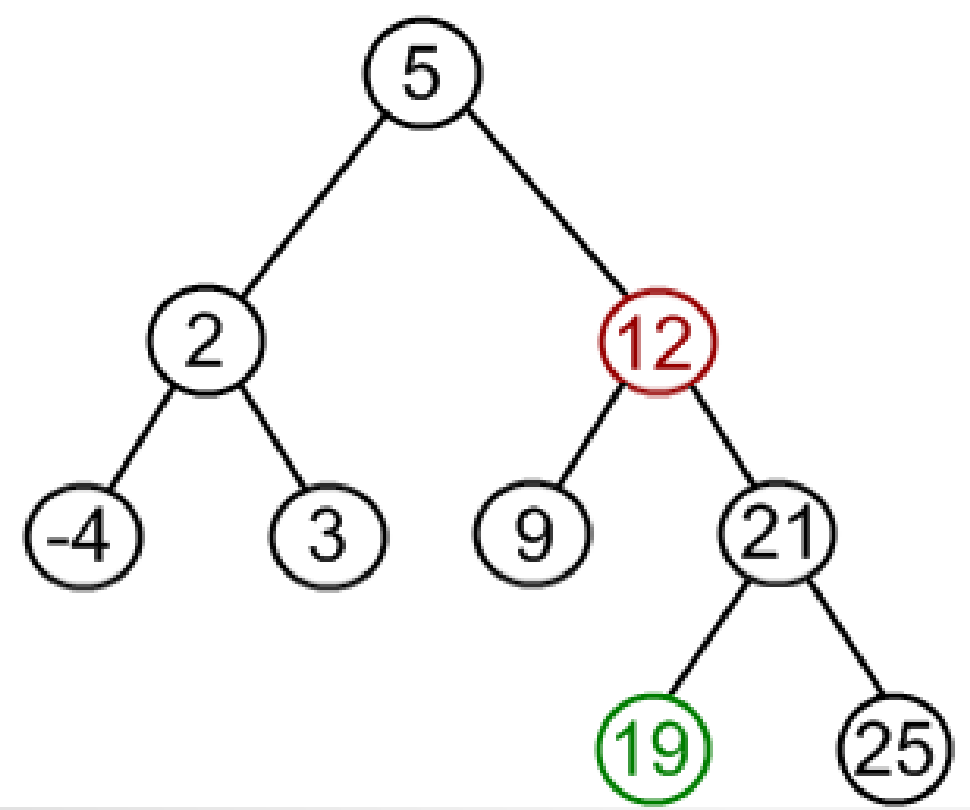
- Remove 12 from a BST.

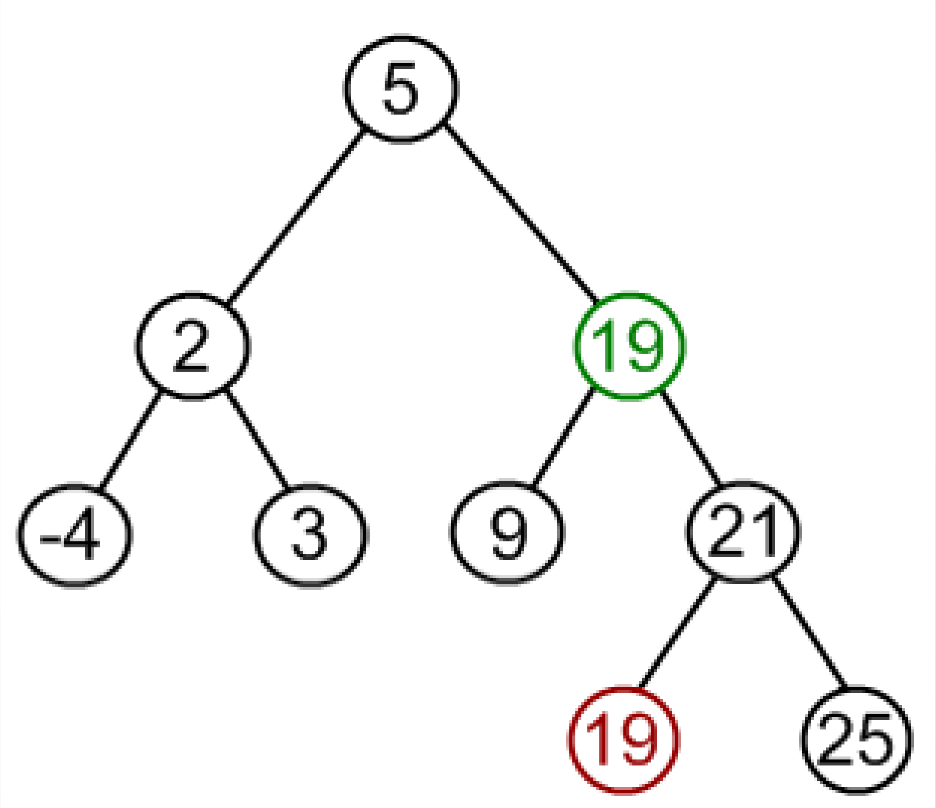


Flashback!

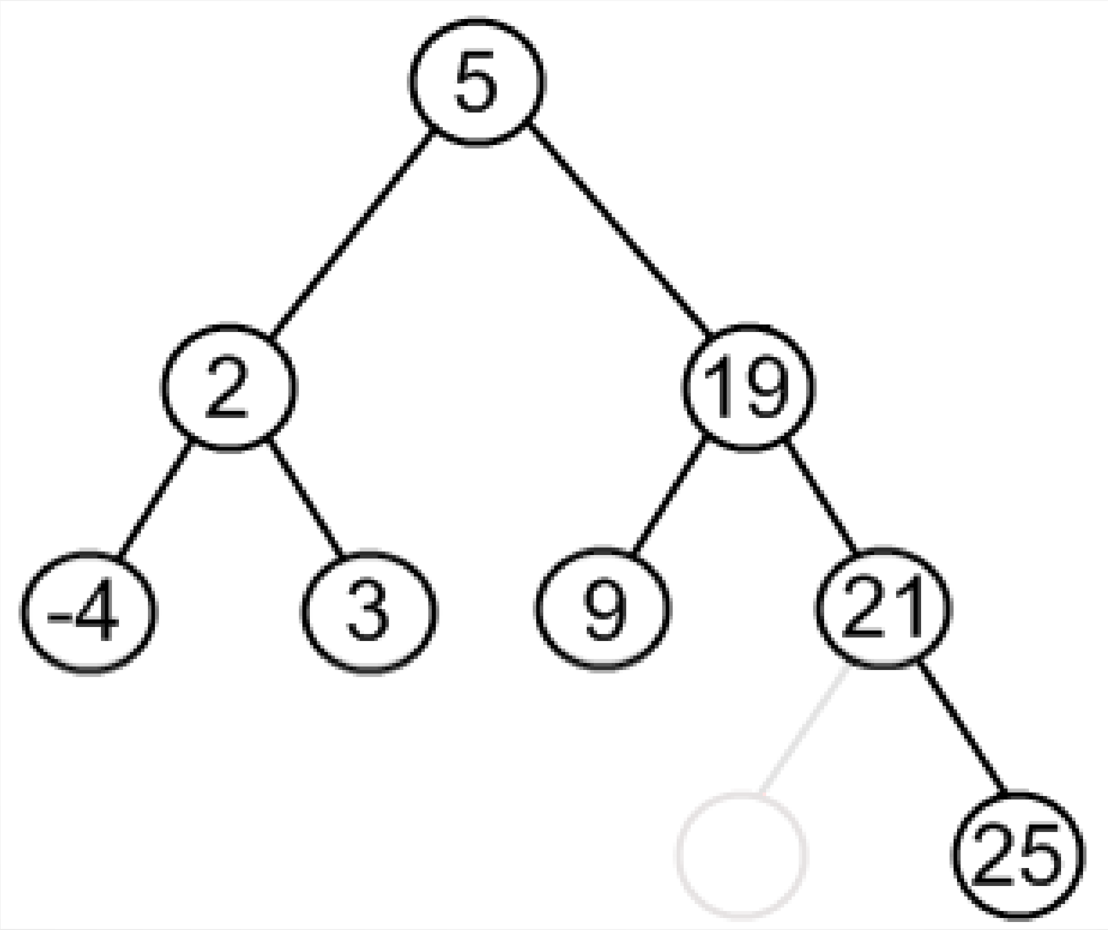
- Remember the algorithm:
  - Choose minimum element from the right subtree
  - Replace node to be deleted by that element

Example: Case 3





Example: Case 3

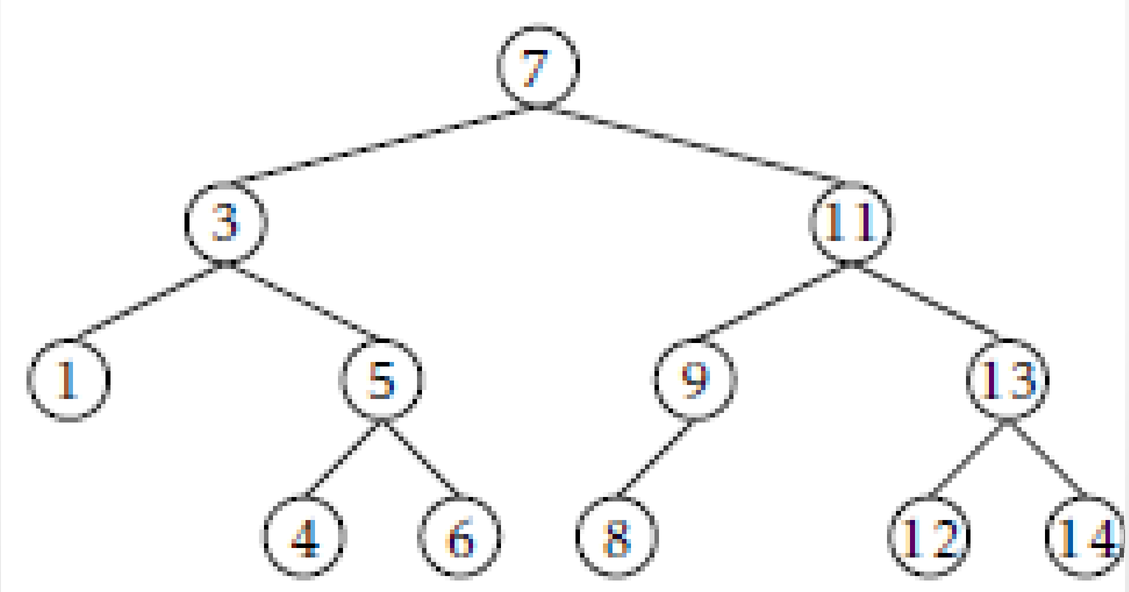


Exercise

- Insert into an initially empty binary search tree, items with the following keys (in the same order)
  - 30, 40, 24, 58, 48, 26, 11, 13
  - What happens if the values are entered in ascending order starting from 11
  - Try the reverse order: 13, 11, 26, 48, 58, 24, 40, 30

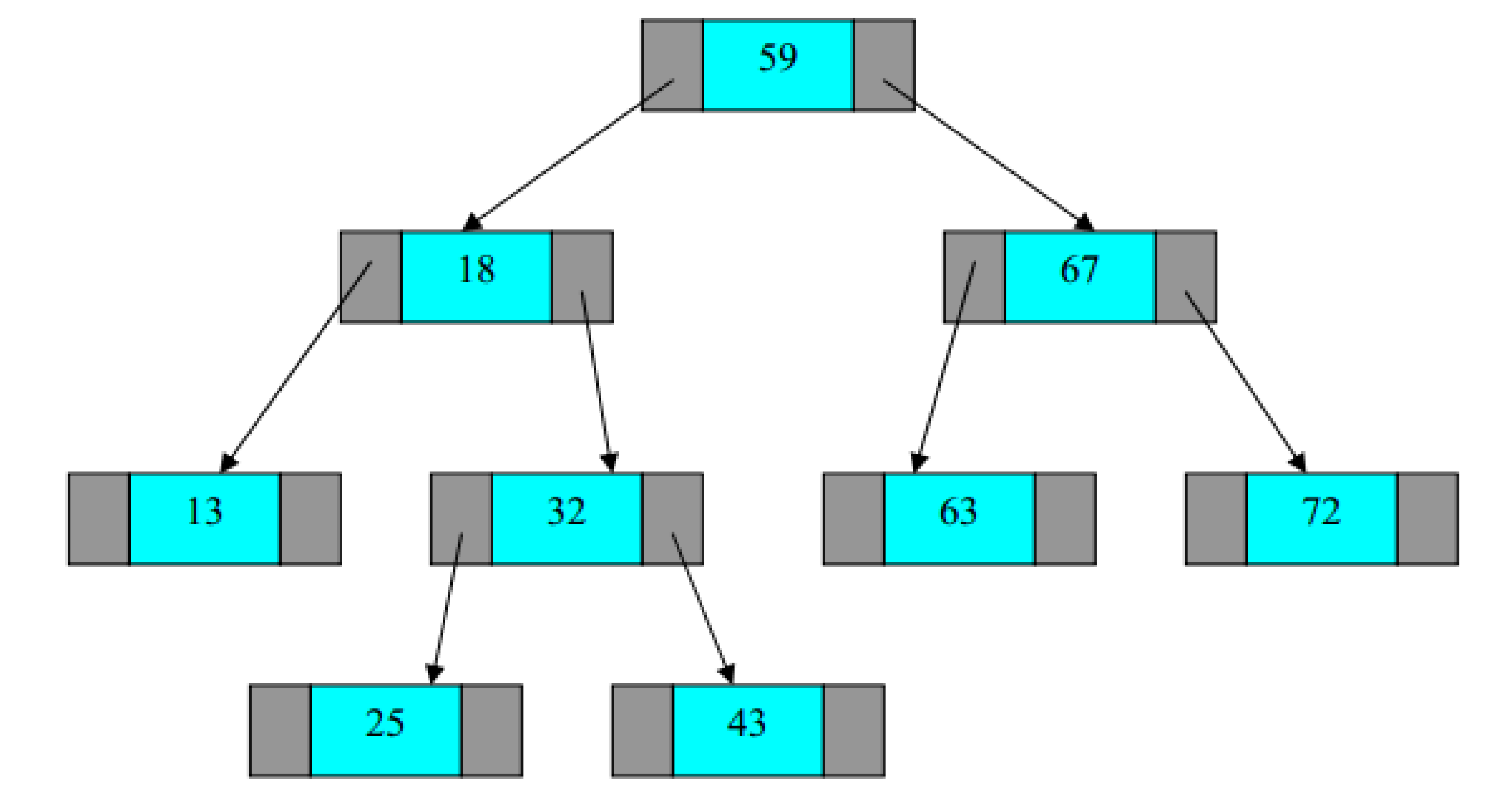
Exercise

- Consider the following binary search tree
  - Illustrate what happens when we add the values 3.5 and then 4.5 to this tree
  - Illustrate what happens when we remove the values 3 and then 5 from the tree



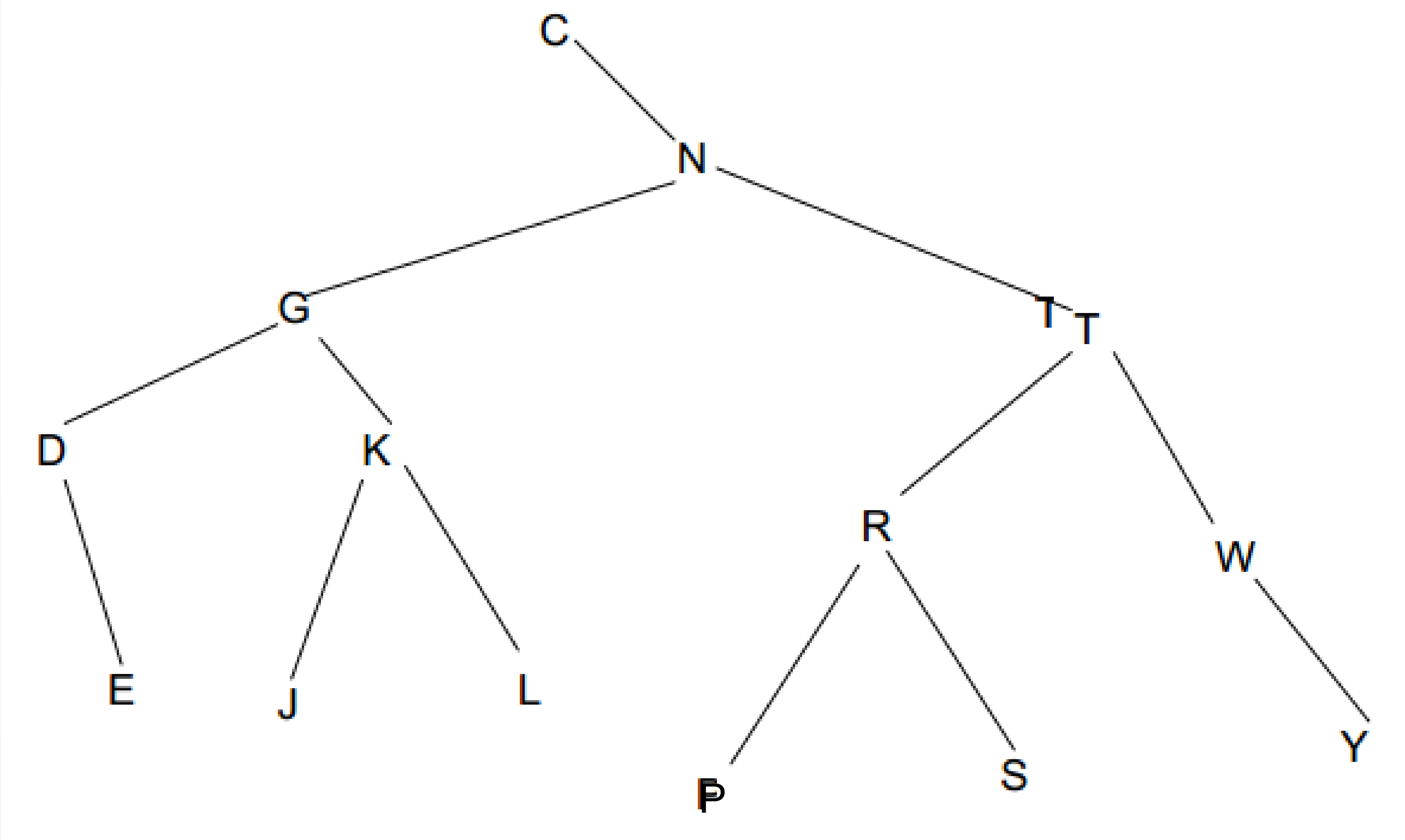
Exercise

- Delete 18 from this binary tree and illustrate the result



Exercise

- Delete node N



Exercise

- If we have some BinarySearchTree and perform the operations add(x) followed by remove(x) (with the same value of x) do we necessarily return to the original tree?
- In the case of deleting a node v with 2 children, why should we replace v with the child from the right sub tree why not the left? Is this possible? Justify your answer.

Height Balanced Trees

Next Class