
CS206

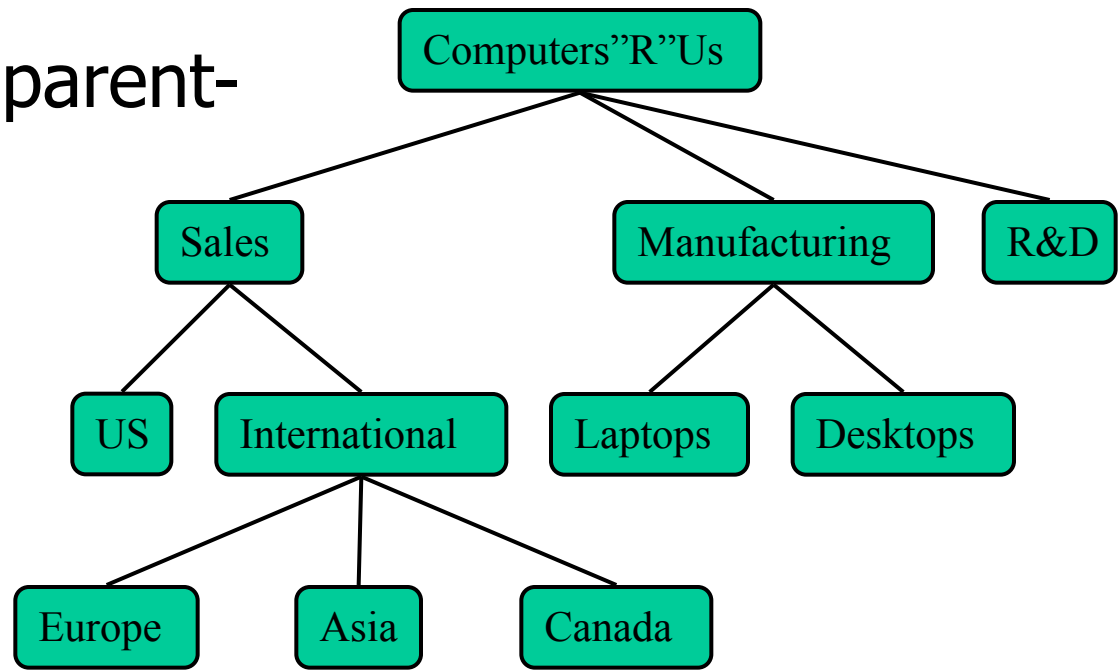
Trees

Lab

- If you have a sorted linked list, should you have
 - addFirst
 - addLast
 - removeFirst
 - removeLast

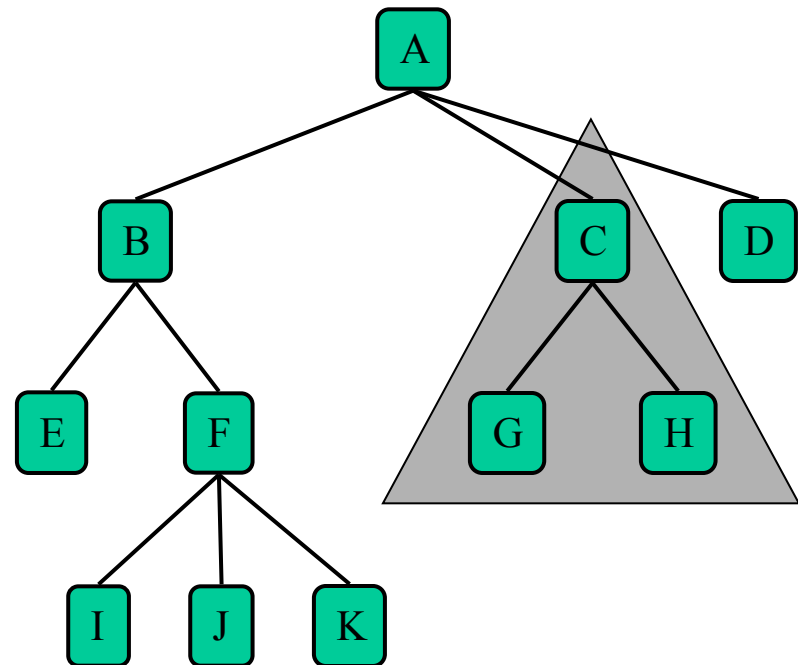
Tree

- A tree is an abstract model of a hierarchical structure
- Nodes have a parent-child relation
- No loops
- One Path



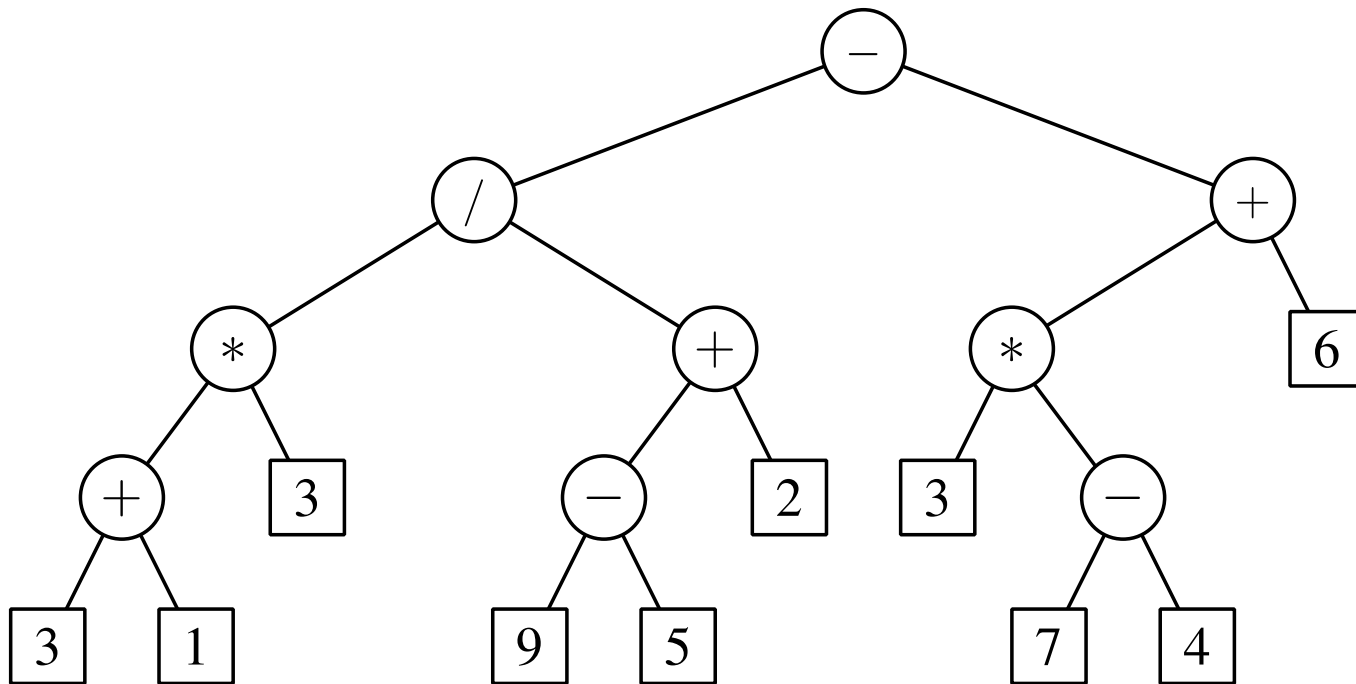
Terminology

- root: no parent – A
 - external node/leaf: no children – E, I, J, K, G, H, D
 - internal node: - node with at least one child - A, B, C, F
 - ancestor/descendent
 - depth - # of ancestors
 - Height - max depth
- Subtree: tree consisting of a node and its descendants



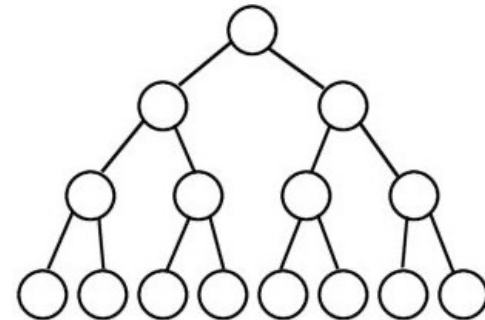
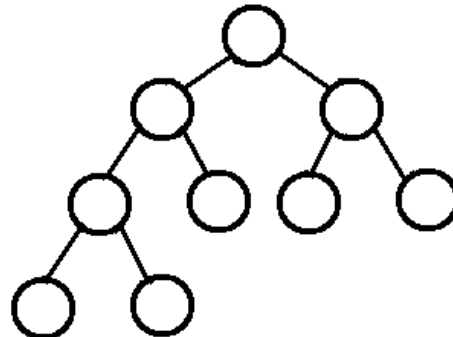
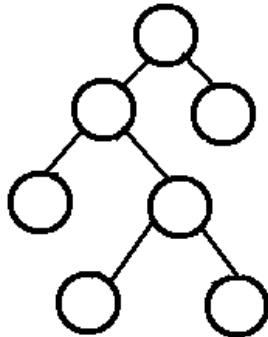
Binary Tree

- An ordered tree with every node having at most two children – left and right



Type of Binary Trees

- A binary tree is **proper** (or full) if each node has zero or two children
- A binary tree is **complete** if every level (except possibly the last) is filled
 - A (binary) heap is a complete binary tree



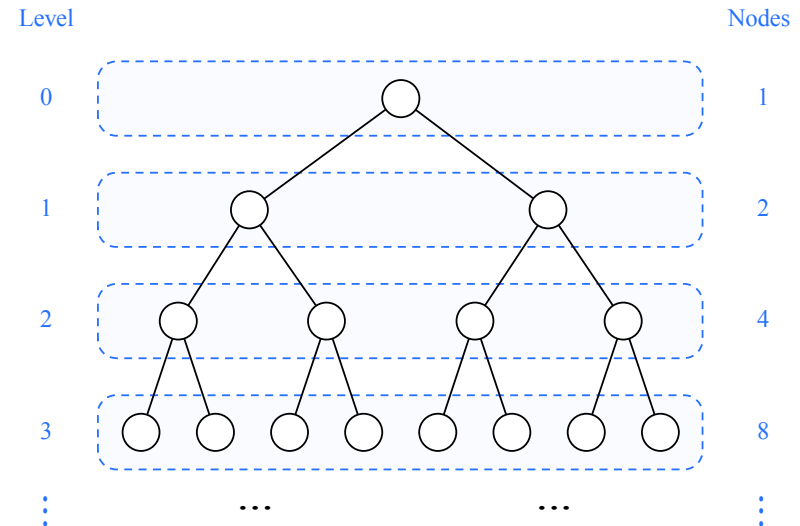
Binary Tree Properties

- Let n denote the number of nodes and h the height of a binary tree

- $h + 1 \leq n \leq 2^{h+1} - 1$

- $\log(n + 1) - 1 \leq h \leq n - 1$

- Height of a binary tree is usually (you hope) $O(\log n)$ of the max number of nodes — true worst case $O(n)$

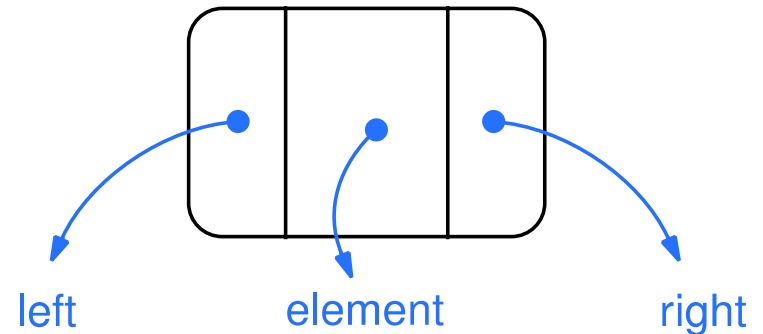


Interface

```
public interface TreeInterface<B>
{
    int size();
    int height();
    boolean isEmpty();
    boolean contains(B element);
    void insert(B element);
    B remove(B element);
}
```

Implementation

```
protected class Node<F> {  
    F payload;  
    Node<F> right;  
    Node<F> left;  
  
    public Node(F e) {  
        payload = e;  
        right = null;  
        left = null;  
    }  
  
    public String toString() {  
        return payload.toString();  
    }  
}
```



This looks a lot like a doubly linked list!!
So, is a doubly linked list a tree?

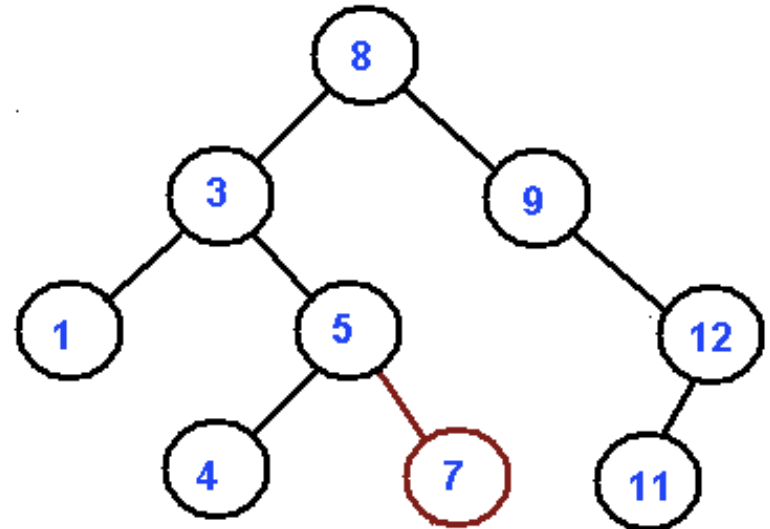
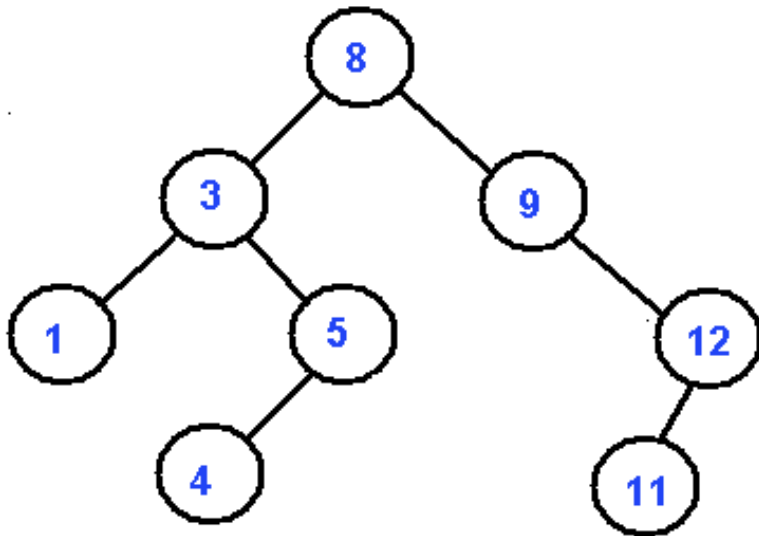
Class

```
public class LinkedBinaryTree<E extends Comparable<E>>  
implements TreeInterface<E> {  
  
    protected int size;  
    protected Node<E> root;
```

Class name violates Encapsulation!

Insertion

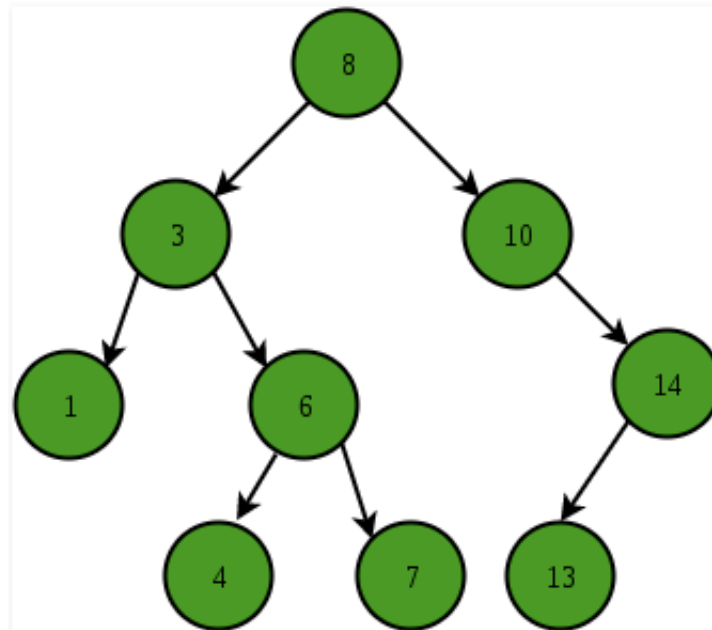
- smaller to the left, bigger to the right



Following this pattern creates a “Binary Search Tree”

contains

- `boolean contains(B element);`
- returns true if found in the tree, false otherwise



Contains Algorithm

- compare with root of **current subtree**
 - root is empty – return false
 - root == element – return true
 - root < element – recurse on right child
 - root > element - recurse on left child

- Comparisons are assumed to be done using Comparable interface (ie, the compareTo method)

Pseudo Code

```
findRec(root, key):  
    if root == null:  
        return false  
    if root.key == key:  
        return true  
    if root.key > key:  
        return findRec(root.left, key)  
    else  
        return findRec(root.right, key)
```

Contains Code

- Write using a recursive helper method

```
public boolean contains(E element) {  
    if (root==null) return false;  
    return containsUtil(root, element)!=null;  
}  
private Node containsUtil(Node treepart, E toBeFound) {  
    ... }  
}
```

Unordered Contains

- Suppose that you did not know relation among children
 - So thing being looked for could be either left or right
 - How would you change containsUtil function
 - Would a tree be a useful structure in this case?

insert

- `void insert(E element);`
- new node is always inserted as a leaf
- inserts to
 - left subtree if element is smaller than subtree root
 - right subtree if larger
- Pre-case: if `root=null` then `root=new Node`

```
public void insert(E element) {  
    if (root==null) {  
        root=new Node<E>(element);  
        size = 1;  
    } else  
        insertUtil(root, element);  
}
```

Groups

- Draw trees for
- 4, 5, 6, 49, 43, 31, 19, 10, 11, 8, 17
- 17, 31, 8, 19, 43, 11, 5, 49, 10, 6, 4

```
private void insertUtil(Node treepart, E toBeAdded) {  
    ... }  
}
```

size() without size

```
public int sizeAlt() {  
    return iSize(root);  
}  
private int sizeAltUtil(Node treepart) {  
    if (treepart==null) return 0;  
    return 1 + sizeAltUtil(treepart.left) +  
            sizeAltUtil(treepart.right);  
}
```

Height / maxDepth

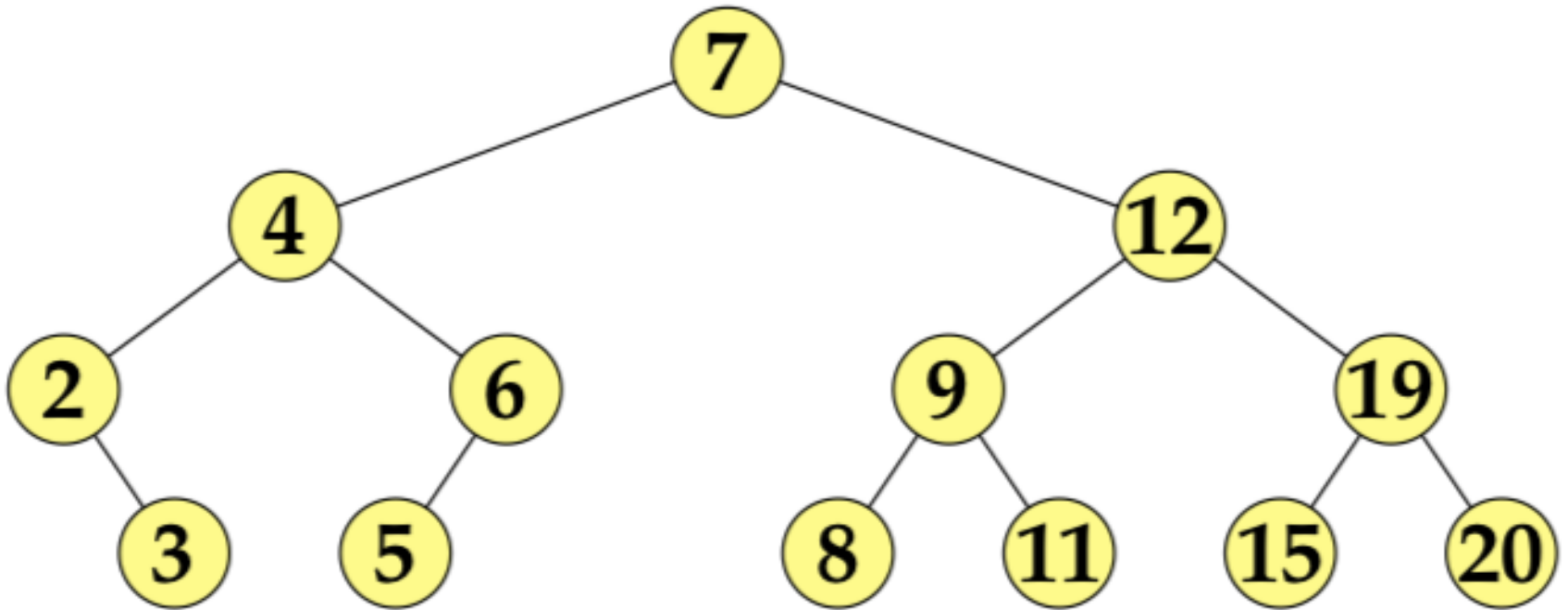
Again, using a recursive helper method

```
@Override
public int height()
{
    return maxDepthUtil(root, 1);
}

int maxDepthUtil(Node n, int depth) {
    ...
}
```

live write

Traversals / Printing



Postorder traversal

```
public void printPostOrder() {
    iPrintPostOrder(root, 0);
    System.out.println();
}

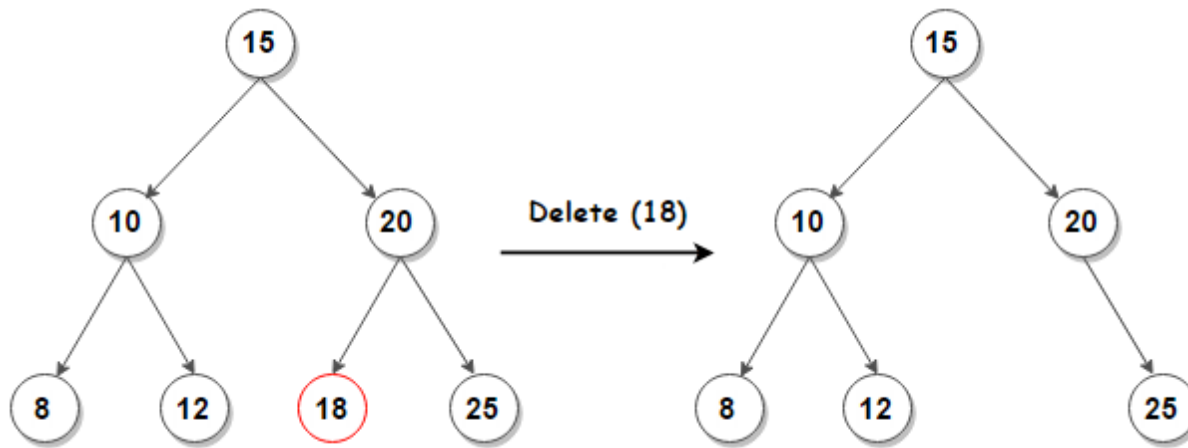
private void iPrintPostOrder(Node treePart, int depth) {
    if (treePart==null) return;
    iPrintPostOrder(treePart.left, depth+1);
    iPrintPostOrder(treePart.right, depth+1);
    System.out.print("[ "+treePart.payload+", "+depth+" ]");
}
```

Remove

- `boolean remove(E element);`
- returns true if element existed and was removed and false otherwise
- Cases
 - element not in tree
 - element is a leaf
 - element has one child
 - element has two children

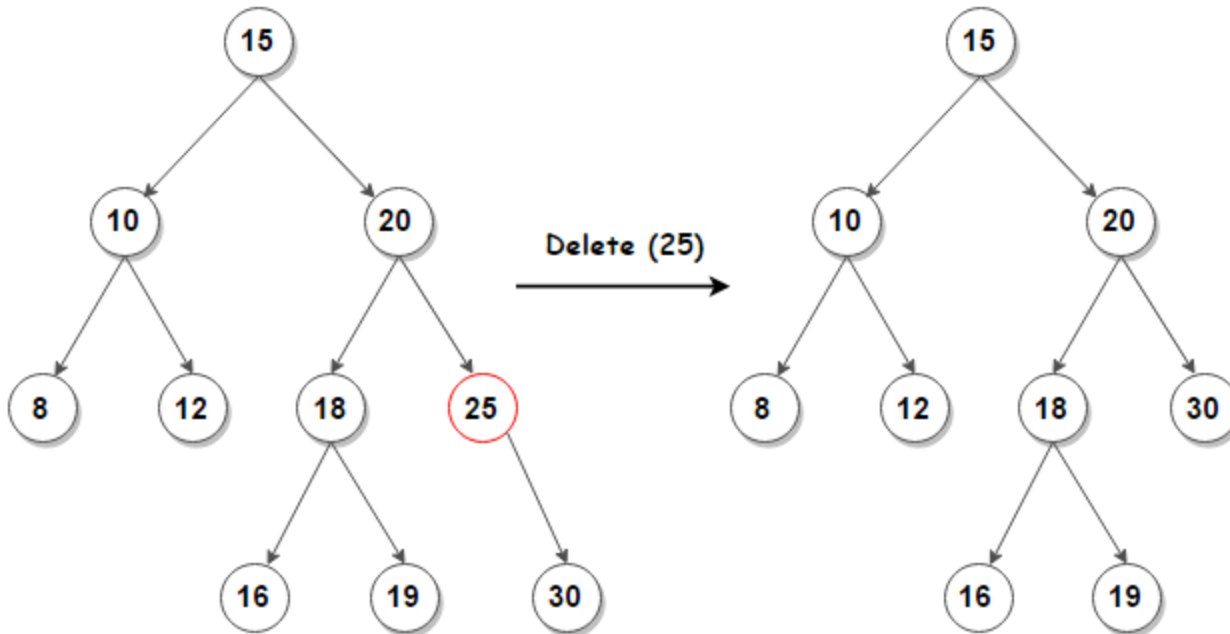
Leaf

- Just delete



One child

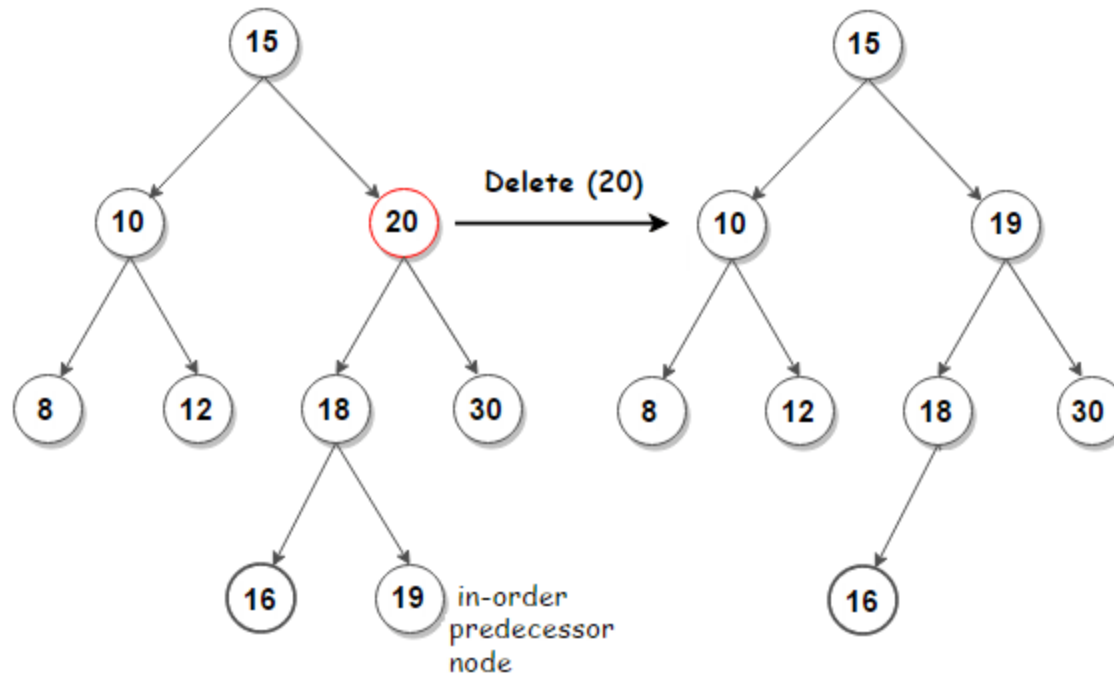
- Replace with child – skip over like in linked list



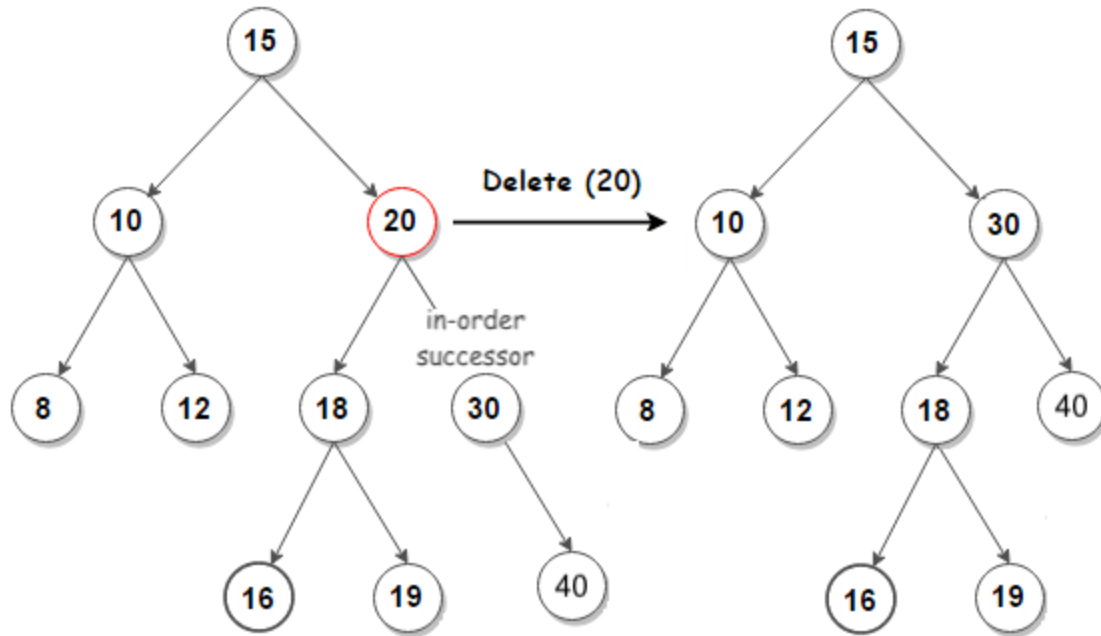
Two Children

- Replace with in-order predecessor or in-order successor
- in-order predecessor
 - rightmost child in left subtree
 - max-value child in left subtree
- in-order successor
 - leftmost child in right subtree
 - min-value child in right subtree

Replace with Predecessor



Replace with Successor



Lab

- Given the data:
- 6, 19, 10, 5, 43, 31, 11, 8, 4, 17, 49, 36
- Draw the binary tree
- Write the preorder traversal of your tree
- What the height of the tree?
- If the data were re-arranged, what is the shortest possible tree?

```
private Node containsUtil(Node treepart, E toBeFound)
{
    if (treepart==null) return null;
    int cmp = treepart.element.compareTo(toBeFound);
    if (cmp==0)
    {
        return treepart;
    }
    else if (cmp<0)
    {
        return containsUtil(treepart.left, toBeFound);
    }
    else // cmp>0
    {
        return containsUtil(treepart.right, toBeFound);
    }
}
```

Draw some Binary Search Trees

- 11, 6, 8, 19, 4, 10, 5, 17, 43, 49, 31
- 6, 19, 10, 5, 43, 31, 11, 8, 4, 17, 49
- 4, 5, 6, 49, 43, 31, 19, 10, 11, 8, 17
- 17, 31, 8, 19, 43, 11, 5, 49, 10, 6, 4