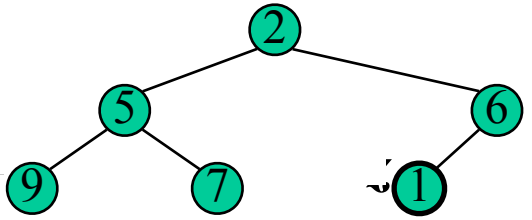
Trees

mostly chapter 26

Binary Tree — terms



Term	Definition
Node	A part of a tree.
Parent	A node that has children
Child	A node that has parents. Child nodes have
Binary Tree	A structure of nodes such that parent nodes
Root	The node in a tree that has no parent.
Leaf	Any node that has no children
Depth	the distance a node is from the root
Height	The maximum depth of any node in the tree
Subtree	The part of a tree whose root is a given node
Level	All the nodes at a particular depth

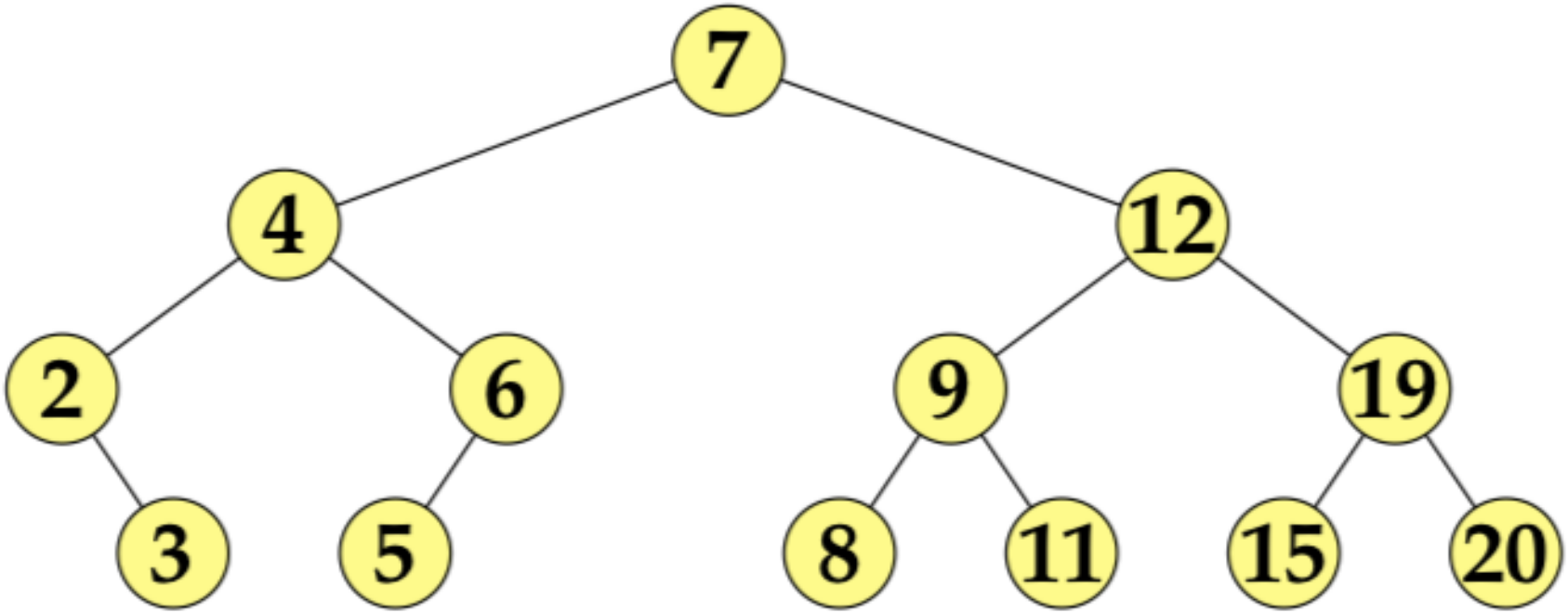
insert

- `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
- Handling Duplicates: Several possibilities: "Just say No", add in right subtree, do something in Node

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

Live write first pseudocode then java for insertUtil

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+"]");
}
```

Practice

- Given the data:
- 6, 19, 10, 5, 43, 31, 11, 8, 4, 17, 49, 36

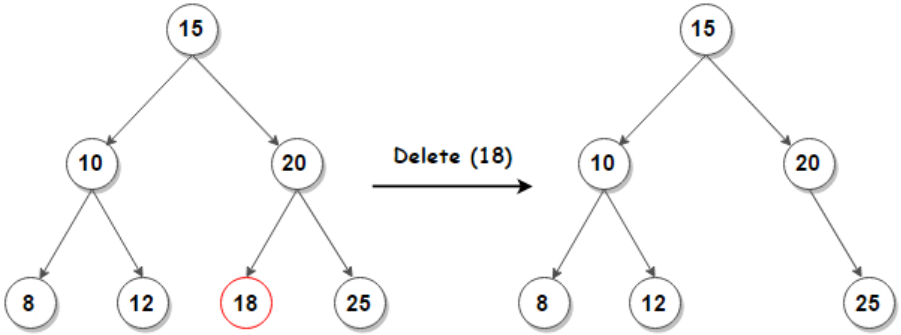
- Draw the binary tree
- Write the preorder traversal of your tree
- Write the postorder traversal of your tree
- What the height of the tree?
- If the data were re-arranged, what is the shortest possible tree?

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

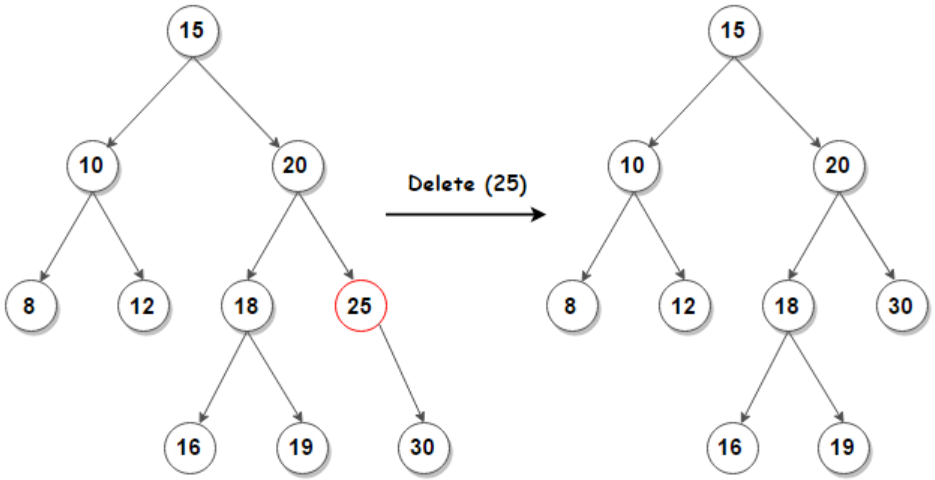
Leaf

- Just delete

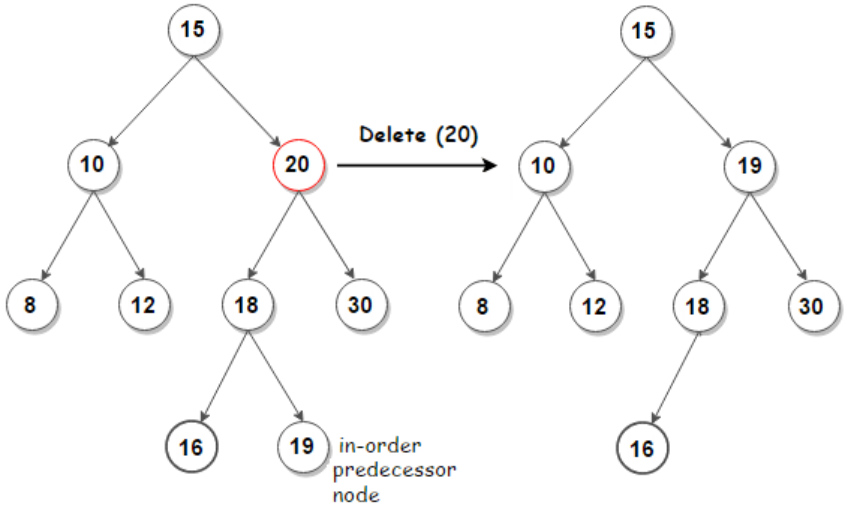


One child

- Replace with child – skip over like in linked list



2 Children
Replace with Predecessor



remove pseudocode

```
boolean remove(element)
  return removeUtil(element, root, null);

boolean removeUtil(element, node, parent)
  if (node==null) return false;
  if (node.payload>element)
    removeUtil(element, node.left, node);
  else if (node.payload<element)
    removeUtil(element, node.right, node);
  else
```

remove pseudocode 2

```
// found the node to delete
if (node.right==null && node.left==null)
  // at a leaf
  if node==parent.right
     parent.right <- null
  else
     parent.left <- node.left
  return true

if (node.right==null) // one child on left
  if node==parent.right
     parent.right <- node.left
  else
     parent.left <- node.left
  return true
if (node.left==null)
  // one descendent on right
  // see node.right above
  return true
```

remove pseudocode 3

```
// two children
successorNode = inorderSucessor(node.right)
node.payload <- successorNode.payload
removeUtil(successorNode.payload,
node.right, node);
return true;
```

Live Write inOrderSuccessor