Trees

mostly chapter 26
## Binary Tree — terms

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

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

```
    7
   / \
  4   12
 / \ / \  /
2  6 9 11 15
|   |   |   |
3   5 8 19 20
```
Postorder traversal

```java
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
• Just delete
One child

- Replace with child – skip over like in linked list
2 Children
Replace with Predecessor

Delete (20)

in-order predecessor node
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

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