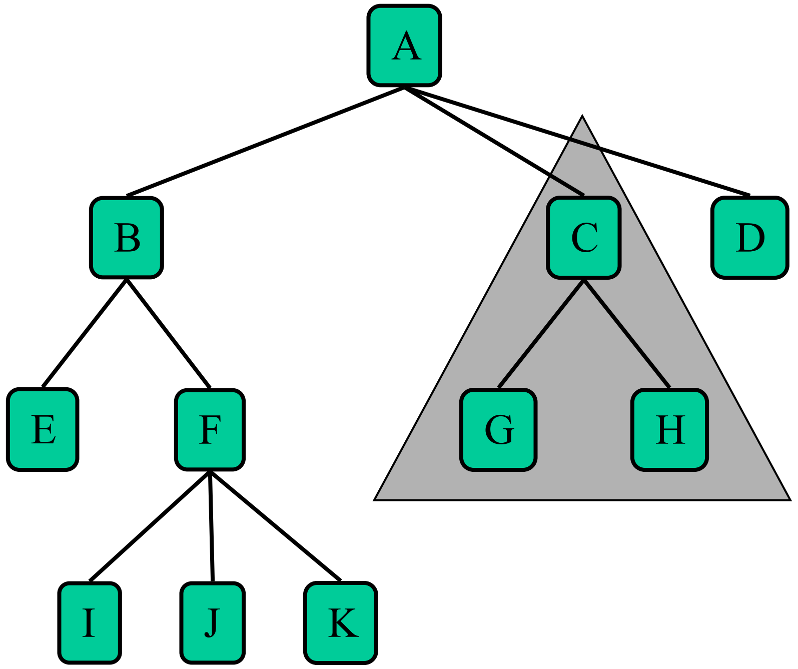
CS206

Trees
Part 2

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



Interface

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

Height / maxDepth

Again, using a recursive helper method.
Slightly different from Tuesday but equivalent.

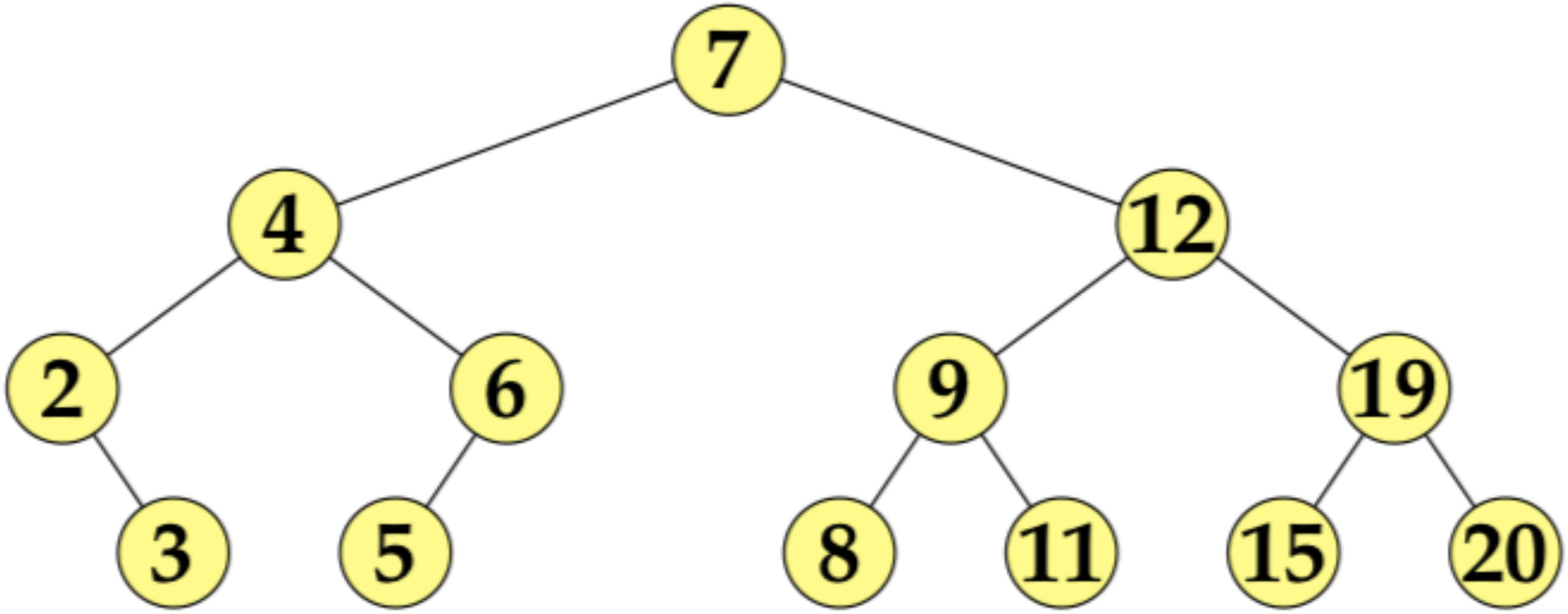
```
@Override
public int height() {
   int tmp = maxDepthUtil(root,0) - 1;
   return tmp>=0 ? tmp : 0;
}

int maxDepthUtil(Node n, int currDepth) {
   if (node == null)
         return currDepth;
     int rd = maxDepthUtil(node.right, currDepth+1);
     int ld = maxDepthUtil(node.left, currDepth+1);
     return rd>ld ? rd : ld;
}
```

size() without size

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

Traversals / Printing



Postorder traversal

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

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

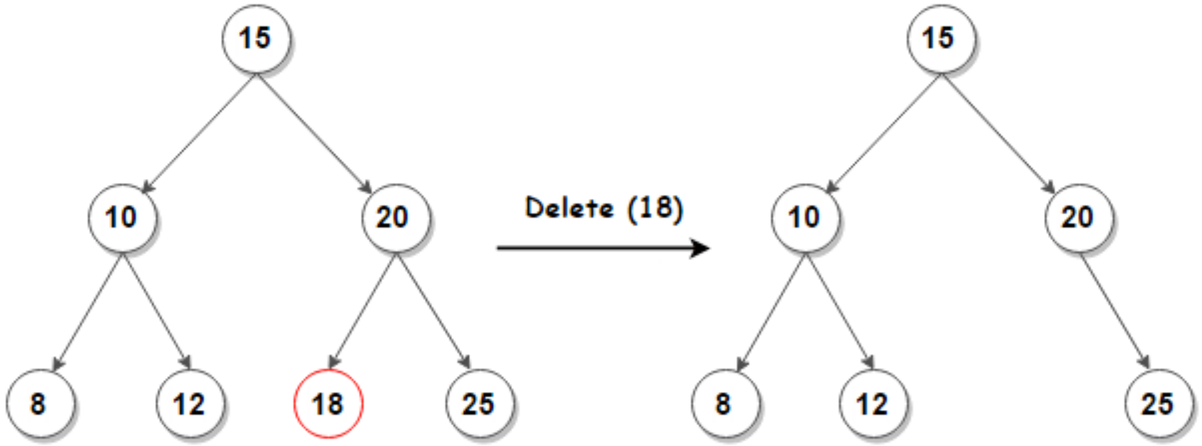
What change to get a pre-order traversal?

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

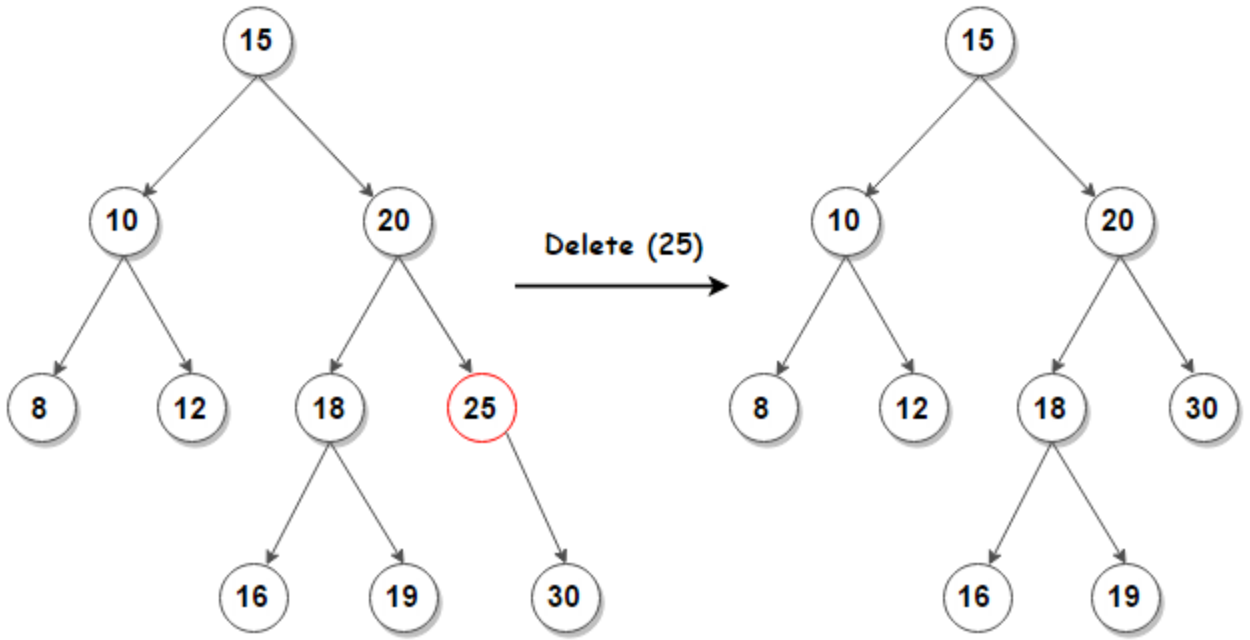
Leaf

- Just delete

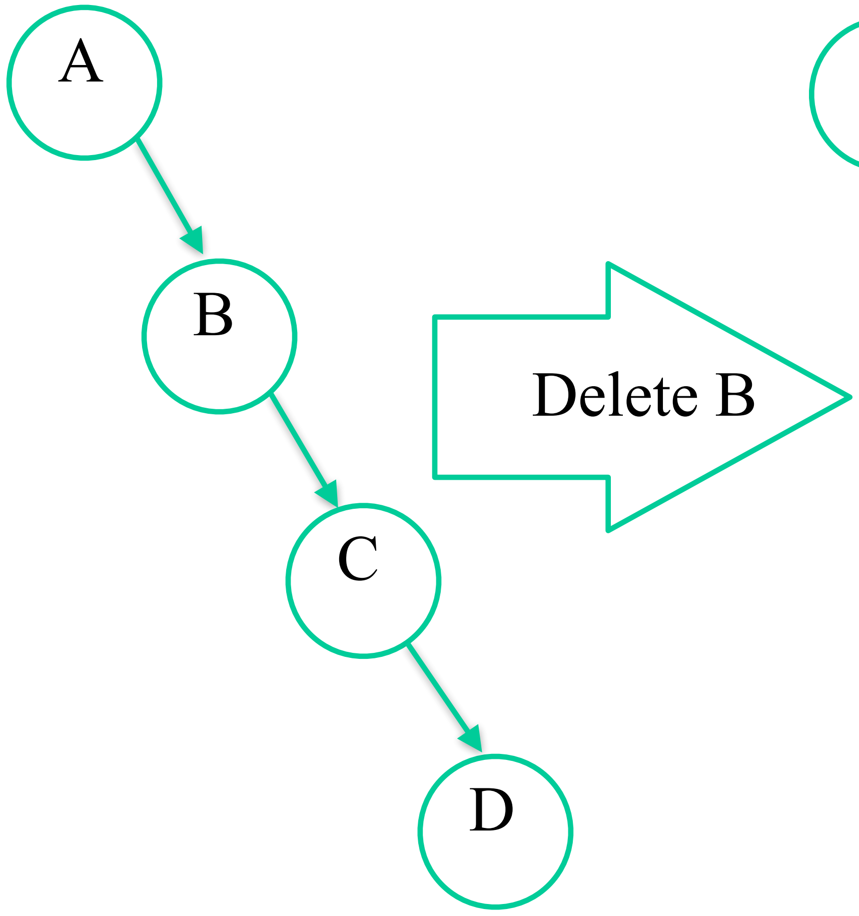


One child

- Replace with child – skip over like in linked list



One-child: not just for leaves

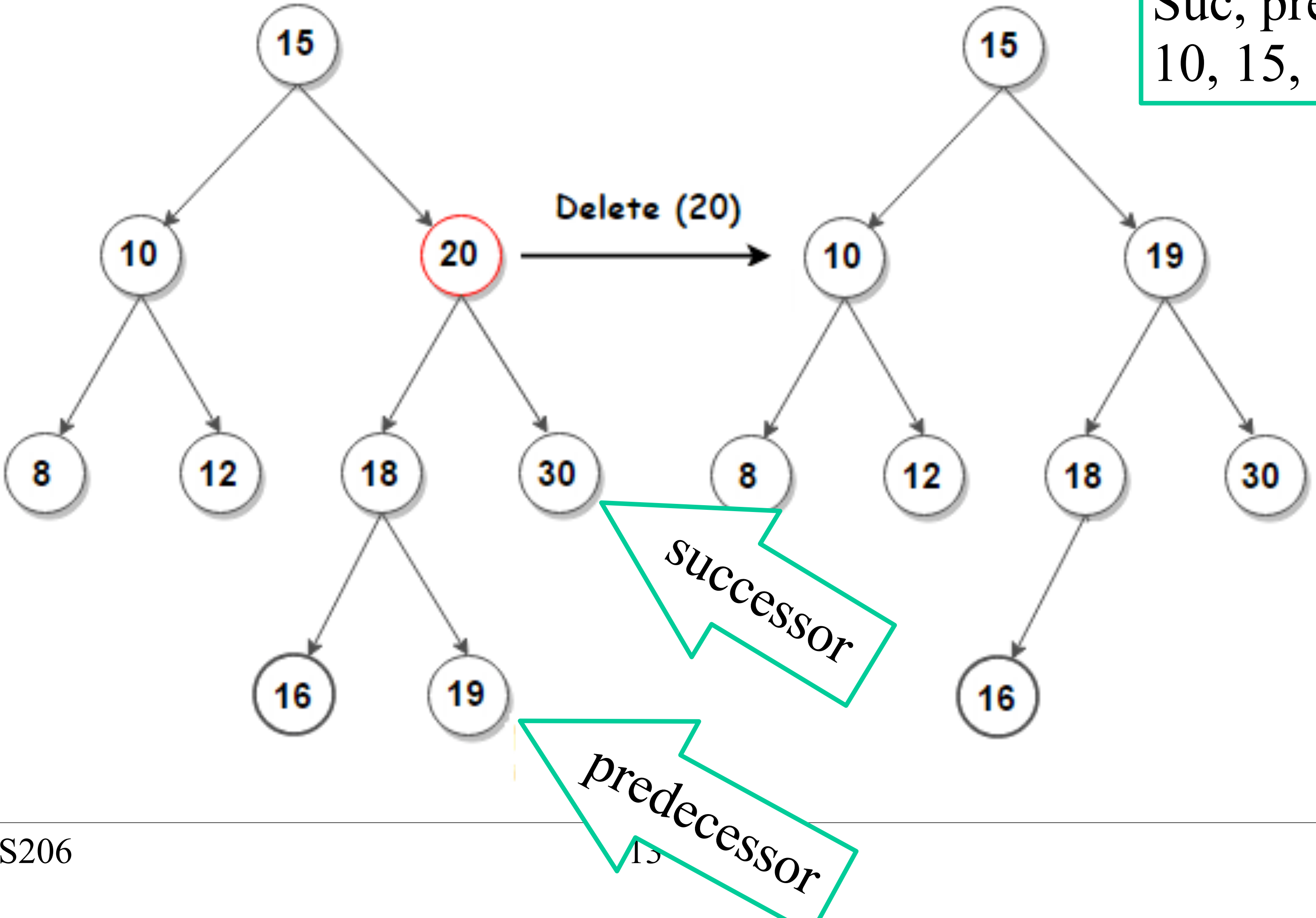


Two Children

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

2 child replacement

Suc, pred for 10, 15, 19?



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
```

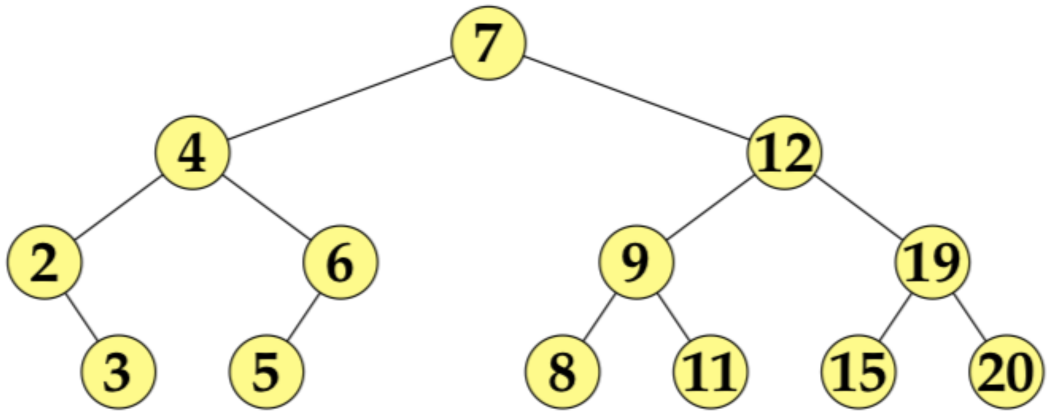
remove pseudocode 2

```
// found the node to delete
if (node.right==null && node.left==null)
  // at a leaf
  parent.remove(node)
  return true
if (node.right==null)
  // one descendent on left
  attach node.left to parent
  return true;
if (node.left==null)
  // one descendent on right
  attach node.right to parent
  return true;
```

remove pseudocode 3

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

Breadth First traversal



0 [7]
1 [4 12]
2 [2 6 9 19]
3 [3 5 8 11 15 20]

mini-lab exercise

- Complete the implementation of insertUtil using pencil and paper is OK.
- Strive to be correct
- Think.
  - Draw pictures of trees and what you want your code to do.
- Take a photo of your code and send it to gtowell206@cs.brynmawr.edu