Trees

Part 4
March 31

Insert

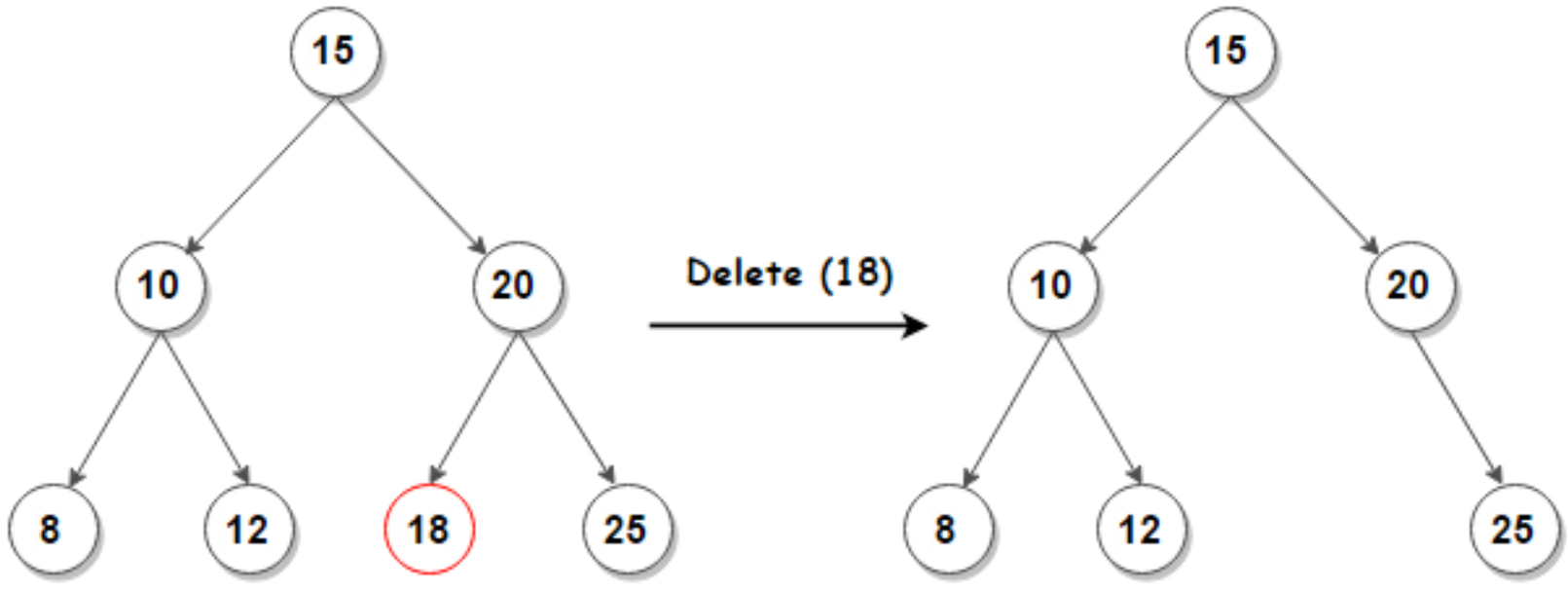
```
public void insert(E element) {
    root = iInsert(root, element);
}

private Node iInsert(Node treepart, E element) {
    if (treepart == null) {
        size++;
        return new Node(element);
    }
    int cmp = treepart.payload.compareTo(element);
    if (cmp==0) return treepart;
    if (cmp>0) {
        treepart.left = iInsert(treepart.left, element);
        return treepart;
    }
    else {
        treepart.right = iInsert(treepart.right, element);
        return treepart;
    }}
```

Remove

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

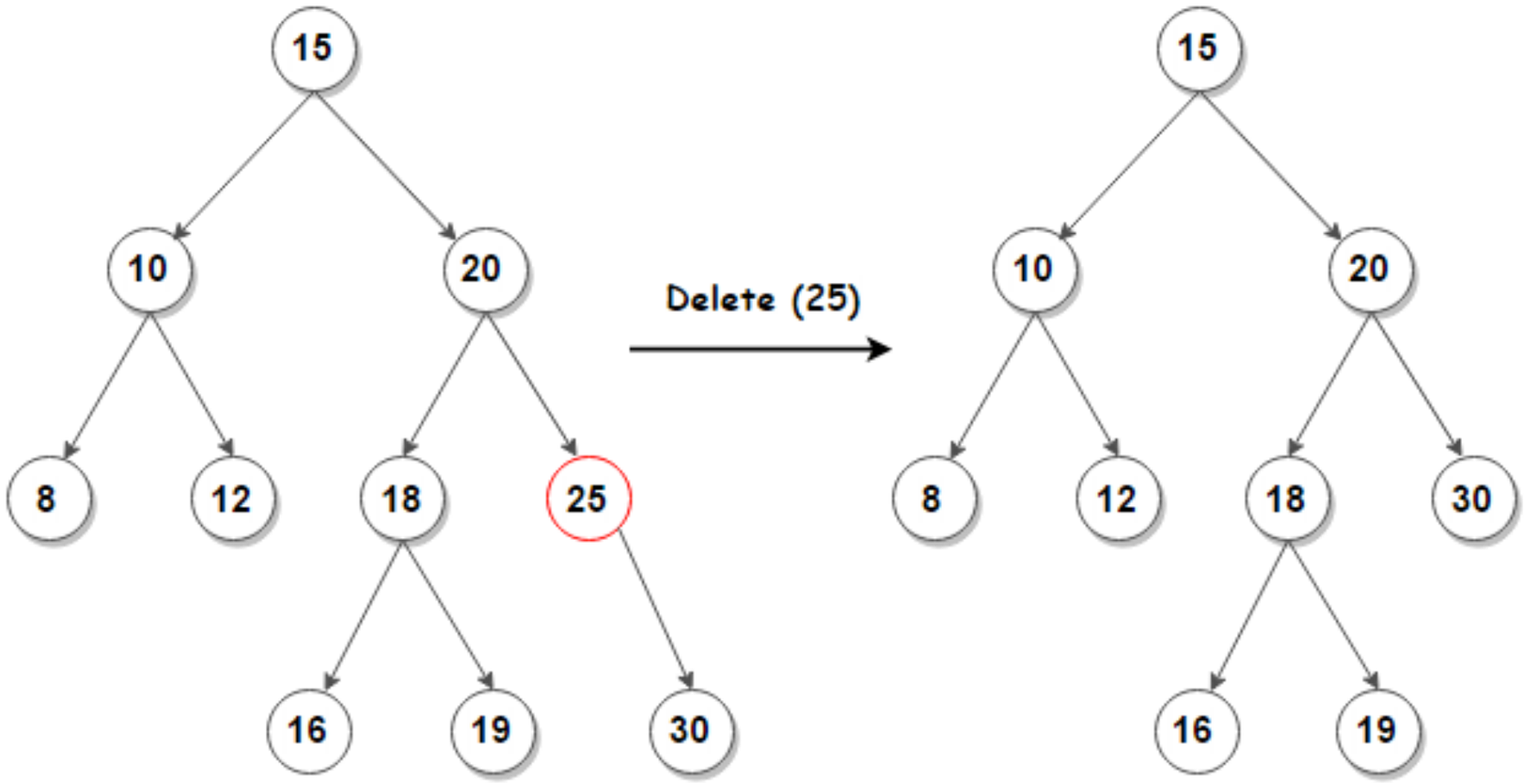
No children (Leaf)

- Just delete

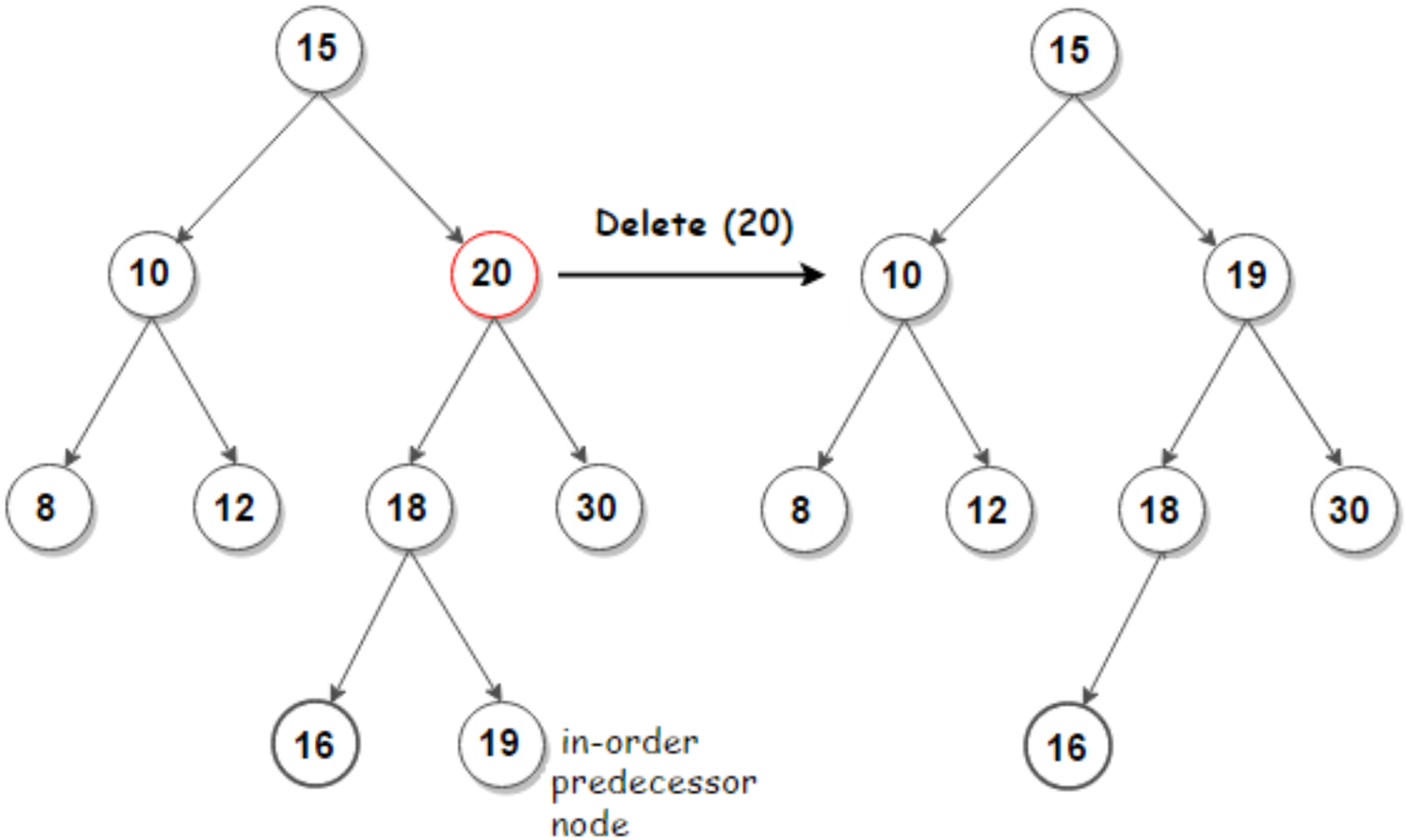


One child

- Replace with child – skip over like in linked list



2 children
Replace with Predecessor



Getting the predecessor

- Pseudocode:

- Go left
  - repeat
    - go right
    - until right child is null

Getting the Max

```
// E pred = iMaxPayload(treepart.left);

private E iMaxPayload(Node treepart) {
     if (treepart.right==null)
          return treepart.payload;
     else
          return iMaxKey(treepart.right);
}
```

Recursive

```
private E iMaxPayloadNR(Node treepart) {
     Node rightChild = treepart.right;
     while (rightChild !=null) {
          treepart = rightChild;
          rightChild = treepart.right;
     }
     return treepart.payload;
}
```

Non-Recursive

Remove
Stepping down the tree

```
public boolean removeAlt(E element) {
    if (root==null)
        return false;
    return iRemoveAlt(root, null, element);
}

private boolean iRemoveAlt(Node treepart, Node parent,
                           E toBeRemoved) {
int cmp = treepart.payload.compareTo(toBeRemoved);
if (cmp>0) {
   if (treepart.left==null) return false;  // case 1
   return iRemoveAlt(treepart.left, treepart, toBeRemoved);
} else if (cmp>0) {
   if (treepart.right==null) return false; // case 1
   return iRemoveAlt(treepart.right, treepart, toBeRemoved);
}
…
```

Remove
Case 2: no children

```
} else {  // cmp==0
    // this is the thing I want to get rid of!!!!
    if (treepart.left==null && treepart.right==null) {
        // Case 1: no children
        if (parent==null) {
            root=null;
        } else {
            if (parent.right==treepart)
                parent.right=null;
            else
                parent.left=null;
        }
        size--;
        return true;
    }
```

Remove
Case 3: 1 child

```
if (treepart.left==null) {
    // the right branch is NOT null
    // Case 2: Only a right child
    if (parent==null) {
        root=treepart.right;
    } else {
        if (parent.right==treepart)
            parent.right = treepart.right;
        else
            parent.left = treepart.right;
    }
    size--;
    return true;
}
```

Code for only left child is essentially identical

Remove
Case 4: 2 children

```
// case 4: Two children
E pred = iMinKey(treepart.right);
iRemoveAlt(treepart.right, treepart, pred);
treepart.payload = pred;
return true;
```

Mini-homework

- On class website open LinkedBinaryTree code for today's lecture.
- Find the remove method.
  - Not removeAlt which I just discussed, remove
- Build a tree with the following data:
  - 154, 181, 85, 99, 118, 57, 116, 190, 135, 174, 80, 43, 86, 42, 70, 183, 50, 149, 82, 130
- Write a detailed trace through all method calls and the call stack for the deletion of 99 followed by the deletion of 154.
  - Along the lines of insert at the beginning of class today.
  - For another example, see lecture notes from March 3 https://cs.brynmawr.edu/Courses/cs206/spring2020/lec12/lec12.pdf
  - Feel free to use VSC breakpoints to help you, or just do it by hand