Traversing the Tree

- Visiting each node in a specific order.
- Recursive methods are most commonly used to traverse a tree.
- Traversal orders
  - preorder
  - inorder
  - postorder

Inorder Traversal

- The aforementioned orders really refers to the order of the recursive call.
- Inorder:
  - call itself to traverse the left subtree
  - visit the current node
  - call itself to traverse the right subtree
  - Visiting means doing something to a node, simplest is to print it out.

inOrder

```
public void inOrder(Node current) {
   if (current == null)
      return;
   inOrder(current.left);
   current.printNode();
   inorder(current.right);
}
```

Preorder and Postorder

- Preorder
  - visit the node
  - call itself on left subtree
  - call itself on right subtree
- Postorder
  - call itself on left subtree
  - call itself on right subtree
  - visit the node

Deleting a Node

- Deletion is the most complicated operation on a BST.
- The tree may need to be reorganized.
- Cases:
  - Node to be deleted is a leaf
  - Node to be deleted has one child
  - Node to be deleted has two children
- First step, find the node to be deleted and its parent

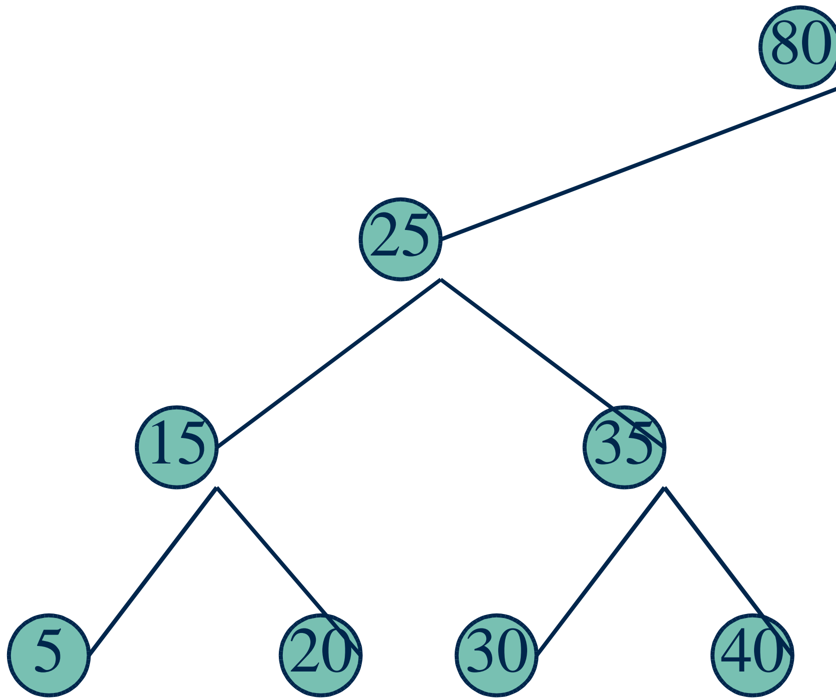
Leaf

```
if (current.getLeft()==null &&
    current.getRight() == null)
         {
          // deleting a leaf
          if (current==root)
              root = null;
          else {
              if (parent.getLeft()==current)
               parent.setLeft(null);
              else
               parent.setRight(null);
          }
         }
```

One Child

```
else if (current.getLeft() == null) {
    // only has a right left
    if (current==root)
     root = current.getRight();
    else if (current==parent.getLeft())
     parent.setLeft(current.getRight());
    else
     parent.setRight(current.getRight());
}
else if (current.getRight() == null) {
    // only has a left left
    if (current==root)
     root = current.getLeft();
    else if (current==parent.getLeft())
     parent.setLeft(current.getLeft());
    else
     parent.setRight(current.getLeft());
}
```

Two Children



- Can not simply replace with one child.
- Must replace with smallest of the right subtree

The Inorder Successor

- Replace the node with its inorder successor.
- To find inorder successor
  - Go to the right child
  - Keep going down to the last left child
  - If right child has no left child, then right child

getAndDelSuccessor

```
private TreeNode getAndDelSuccessor(TreeNode delNode) {
        TreeNode successorParent = delNodem successor = delNode;
        TreeNode current = delNode.getRight();
        while (current != null) {
            successorParent = successor;
            successor = current;
            current = current.getLeft();
        }
        if (successor.getRight()==null && successor.getLeft()==null){
                if (successorParent.getLeft()==successor)
                    successorParent.setLeft(null);
                else
                    successorParent.setRight(null);
                return successor;
            }
        if (successor != delNode.getRight()) {
                successorParent.setLeft(successor.getRight());
                successor.setRight(delNode.getRight());
            }
        return successor;
```

Successor is Right Child of `delNode`

- We can just move the right subtree up.
  1. Disconnect **`current`** from **`parent`** and plug in **`successor`**.
  2. Disconnect **`current`**'s left child and connect it as the left child of **`successor`**.

Delete a nodewith two children

```
 else {
    TreeNode successor = getAndDelSuccessor
                                    (current);
    if (current == root)
        root = successor;
    else if (current==parent.getLeft())
        parent.setLeft(successor);
    else
        parent.setRight(successor);
    if (current.getLeft()!=successor)
       successor.setLeft(current.getLeft());
    if (current.getRight()!=successor)
       successor.setRight(current.getRight());
}
```