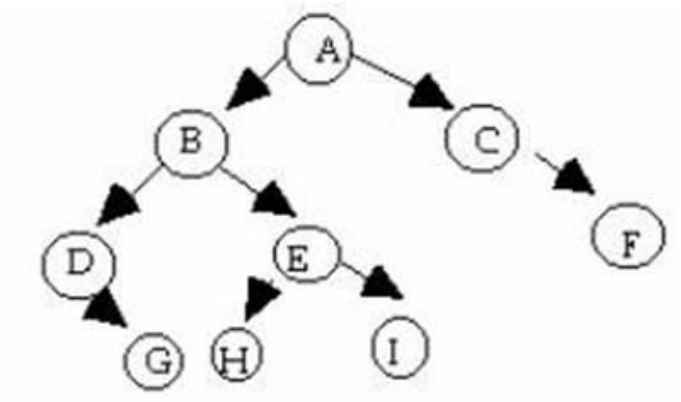
Lab #12 - Binary Search Trees

PART 1: Exercises

1. The following question refers to the tree shown here. The letters are labels, not values.

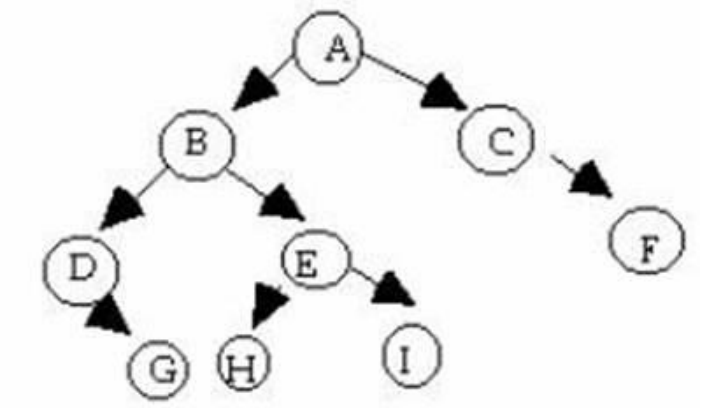


Which traversal generates the order: D G B H E I A C F?

A) postorder
B) inorder
C) preorder
D) no standard traversal

---

2. The following question is based on this tree. The letters are labels, not values.

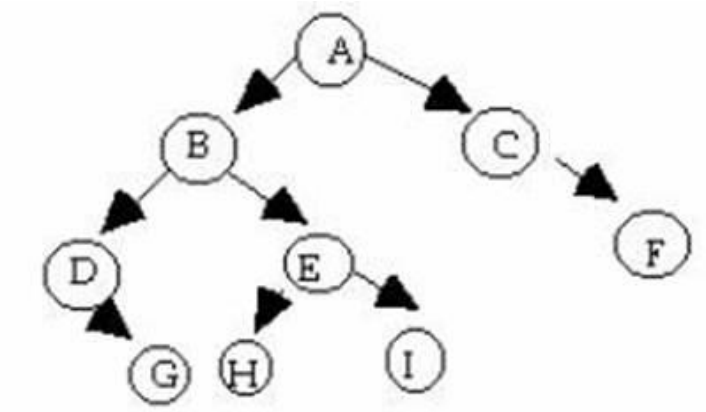


Which traversal generates the order: A B D G E H I C F?

A) preorder
B) postorder
C) inorder
D) none

---

3. The following question is based on this tree. The letters are labels, not values.

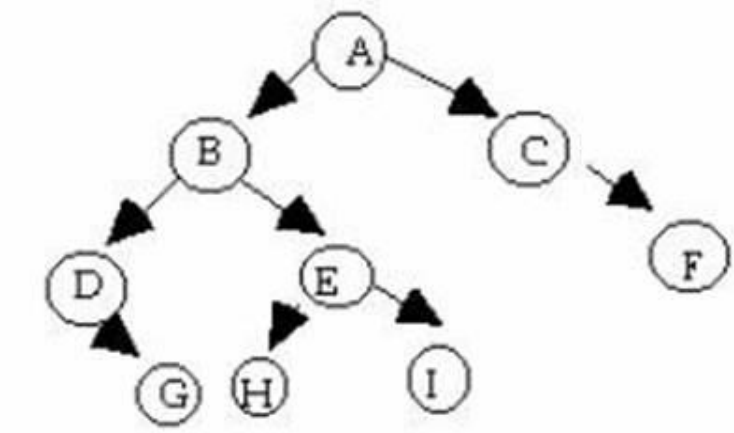


Which traversal generates the order: A B G D E C F H I?

A) preorder
B) postorder
C) inorder
D) none

---

4. The following question is based on this tree. The letters are labels, not values.

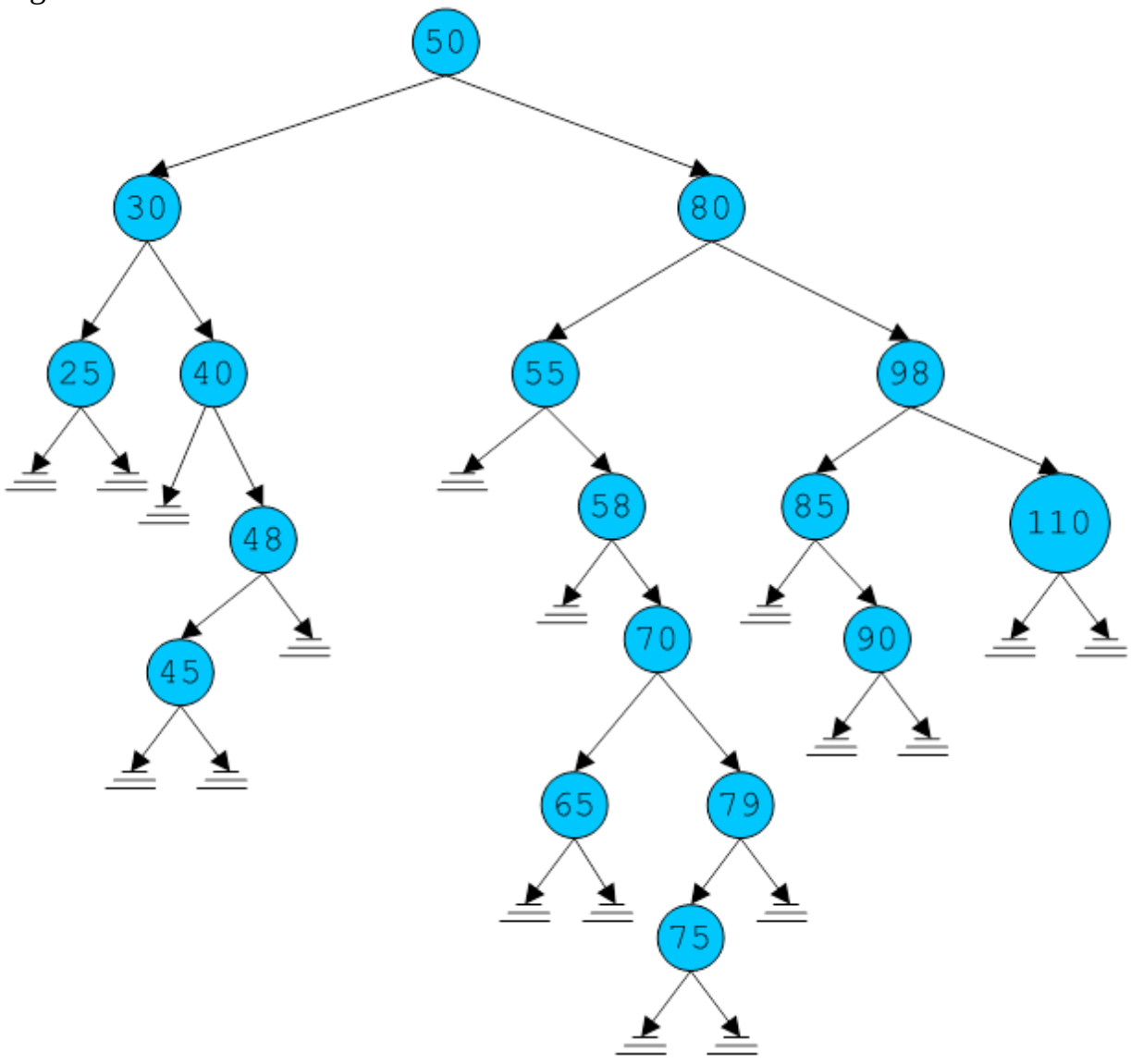


Which traversal generates the order: G D H I E B F C A

A) preorder
B) postorder
C) inorder
D) none

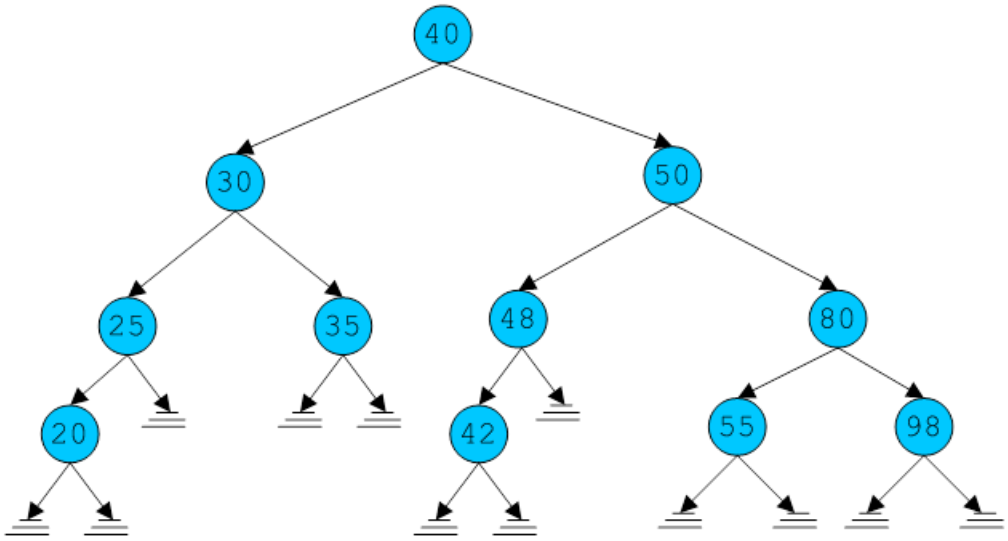
---

5. The following questions are based on this BST. All questions work with the original tree.



1. Delete node 40 and redraw the tree.
2. Delete node 65 and redraw the tree.
3. Delete node 50. Who could replace it?

6. The following questions are based on this BST.



1. Give the sequence of nodes for a **preorder** traversal:
2. Give the sequence of nodes for an **inorder** traversal:
3. Give the sequence of nodes for a **postorder** traversal:

PART 2: Programming

1. Extend the class `BinaryTree` to include a boolean method that determines whether a binary tree is a BST.

ANSWER:

```
public boolean isBinarySearchTree() {
    return isBST(root);
}

//helper method called by isBinarySearchTree
public boolean isBST(BinaryTreeNode<T> tree){
   ...

}
```

2. Extend the class `BinaryTree` to include a boolean method `similarTrees` that determines whether the shapes of two trees are the same (the nodes do not have to contain the same values, but each node must have the same number of children)

ANSWER:

```
public boolean similarTrees(BinaryTreeNode<T> otherTree) {
    return similar(root, otherTree);
}

//helper method called by similarTrees
public boolean similar(BinaryTreeNode<T> tree1, BinaryTreeNode<T> tree2) {
    ...
}
```

3-4. Extend the class `BinaryTree` to include 2 more methods: `nodeCount` (count the number of leaves in a binary tree) and `leavesCount` (count the number of nodes in a binary tree). Use recursion in both! Add statements to the client program (lecture notes) to test these new added methods.

ANSWER:

```
public int treeLeavesCount() {
        return leavesCount(root);
}

//helper method called by treeLeavesCount
private int leavesCount(BinaryTreeNode<T> t)  {
    ...
}
      public int treeNodeCount()  {
        return nodeCount(root);
      }

      //helper method called by treeNodeCount
      private int nodeCount(BinaryTreeNode<T> t) {
         ...
      }
```

References:
Java Programming: From Problem Analysis to Program Design, by D.S. Malik, Thomson Course Technology, 2008