Lab 10: Binary Trees

CS 0445: Data Structures

TAs: Jon Rutkauskas
Brian Nixon
http://db.cs.pitt.edu/courses/cs0445/current.term/

November 18, 2019
University of Pittsburgh, Pittsburgh, PA

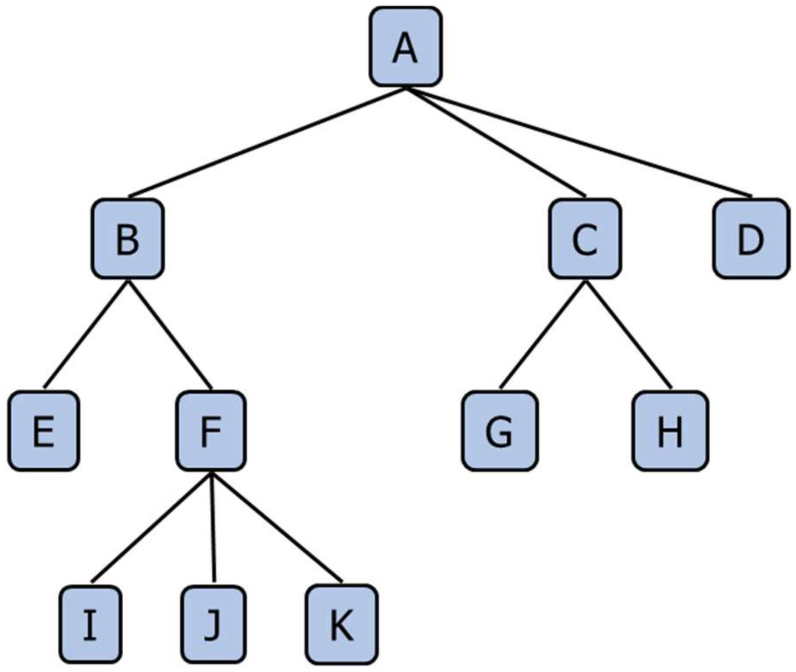
Trees

Efficient search and insert

Flexible

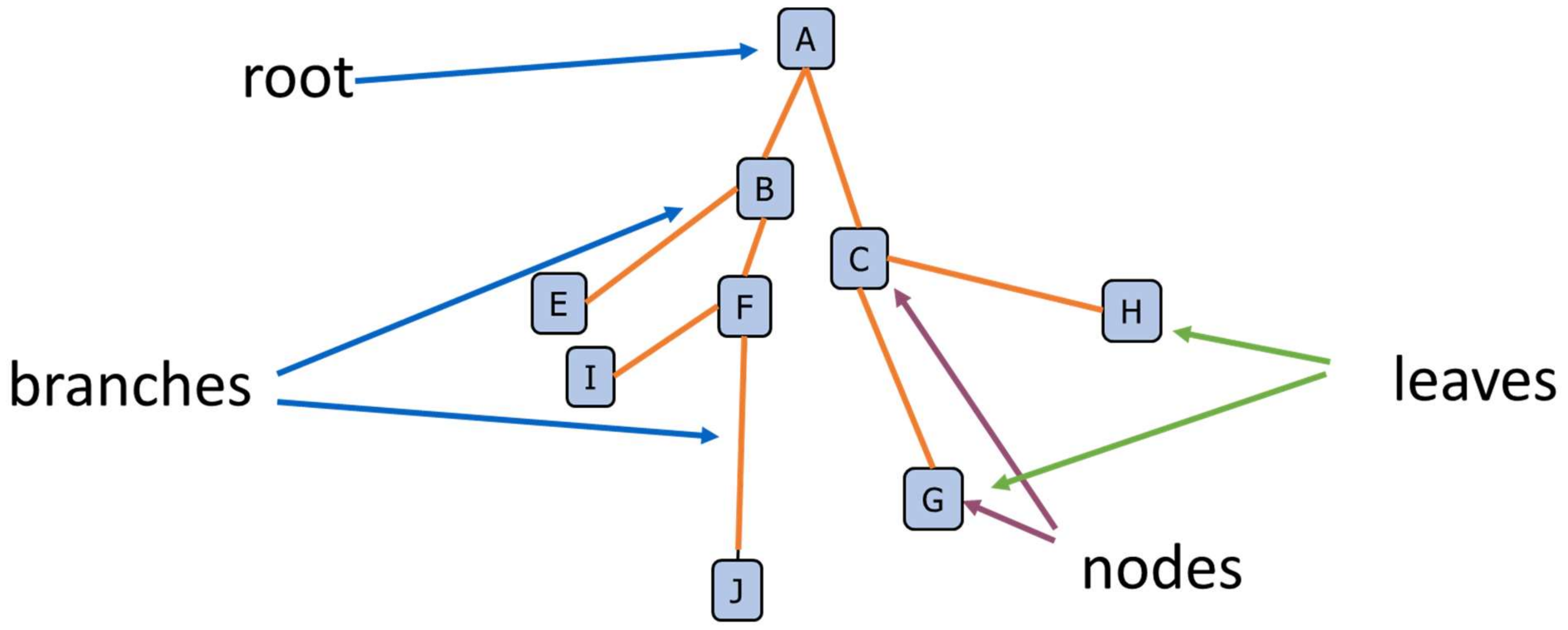
Used in file systems and databases

Good for organization



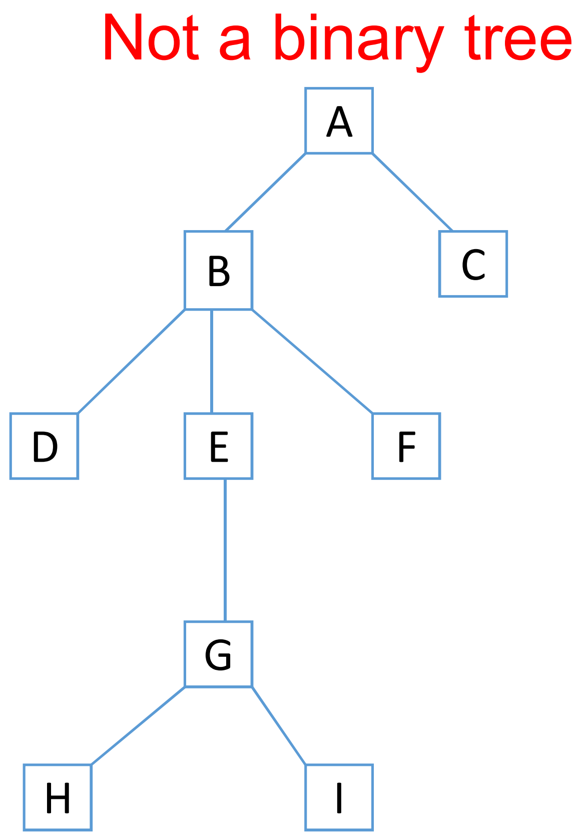
Trees

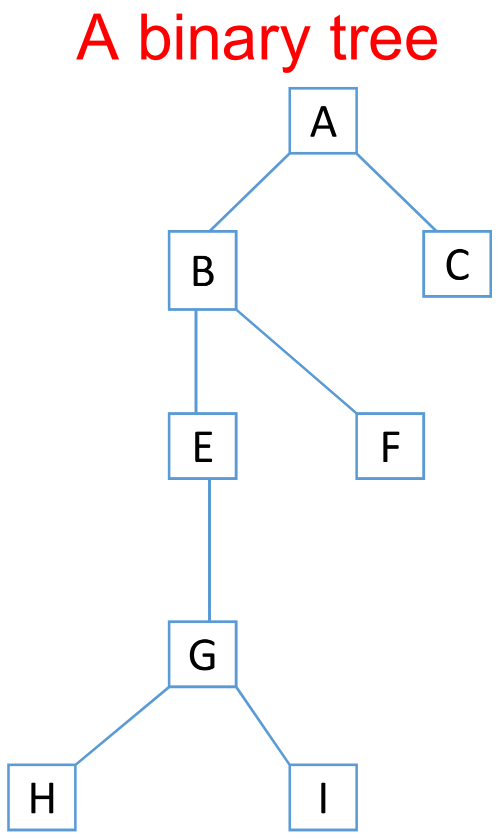
Unlike bags, stacks, or queues, trees are hierarchical



Binary Trees

Each node has at most 2 children



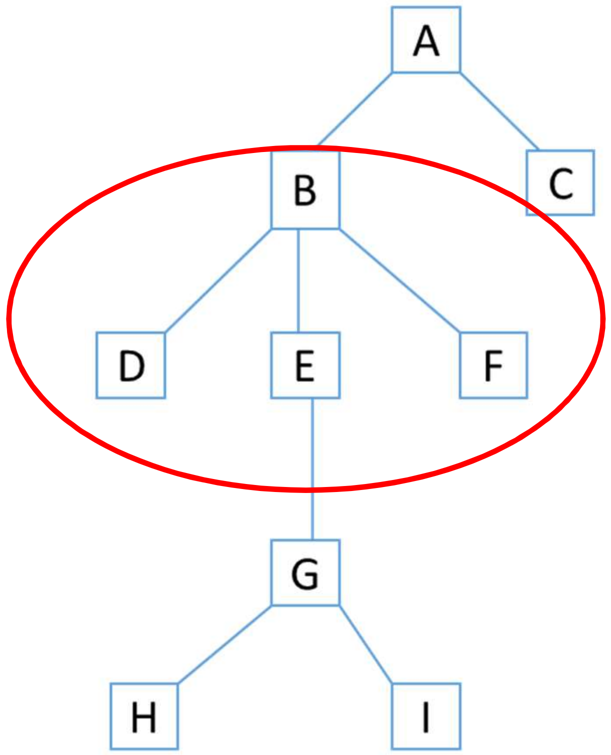


A Recursive Data Structure

Trees are structured in such a way that using recursive methods makes sense

A tree can almost be defined in terms of itself

- A node can have children which are themselves trees

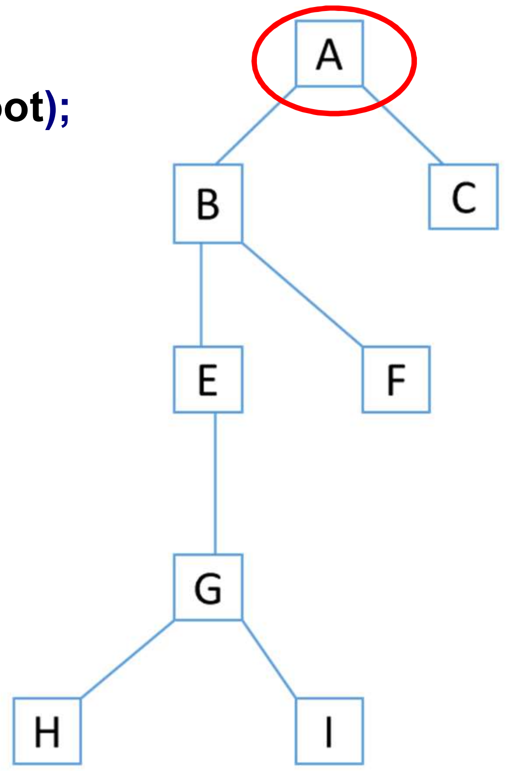


Recursive Method: GetNumberOfNodes

```
public int getNumNodes(BinaryTree tree) {
        //Call a recursive helper method
        int count = 0;
         if(tree.root != null)
                count = recursiveGetNumNodes(tree.root);
        return count;
}
public int recursiveGetNumNodes(Node rood) {
        int count = 1;
        if(root.leftChild != null)
                count += recursiveGetNumNodes(root.leftChild);
        if(root.rightChild != null)
                count += recursiveGetNumNodes(root.rightChild);
        return count;
 }
```

GetNumberOfNodes

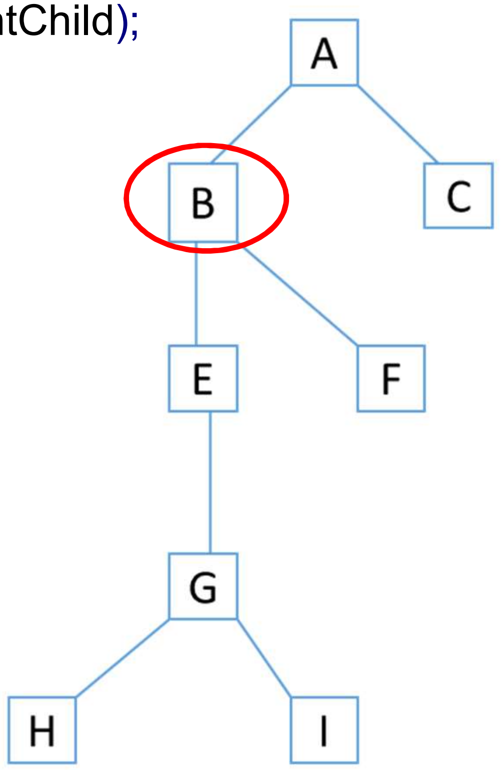
```
public int getNumNodes(BinaryTree tree) {
        //Call a recursive helper method
        int count = 0;
        if(tree.root != null)
                count = recursiveGetNumNodes(tree.root);
        return count;
}
```



Count = 1 + left + right

GetNumberOfNodes

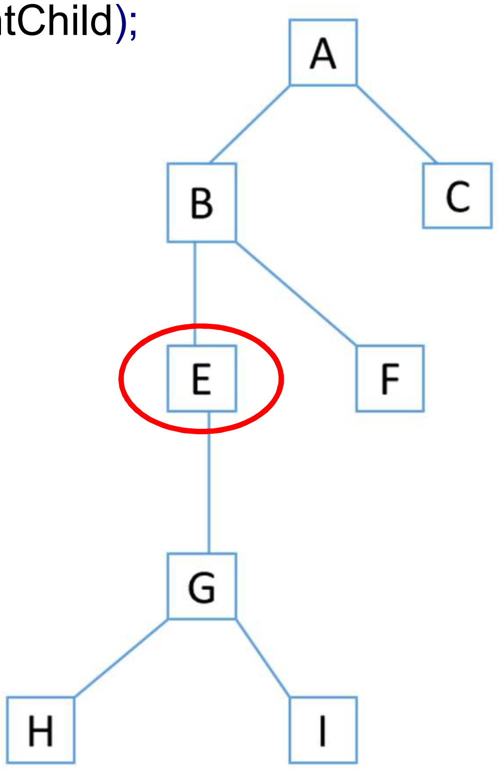
```
public int recursiveGetNumNodes(Node rood) {
        int count = 1;
        if(root.leftChild != null)
                count += recursiveGetNumNodes(root.leftChild);
        if(root.rightChild != null)
                count += recursiveGetNumNodes(root.rightChild);
        return count;
}
```



Count = 1 + left + right

GetNumberOfNodes

```
public int recursiveGetNumNodes(Node rood) {
        int count = 1;
        if(root.leftChild != null)
                count += recursiveGetNumNodes(root.leftChild);
        if(root.rightChild != null)
                count += recursiveGetNumNodes(root.rightChild);
        return count;
}
```

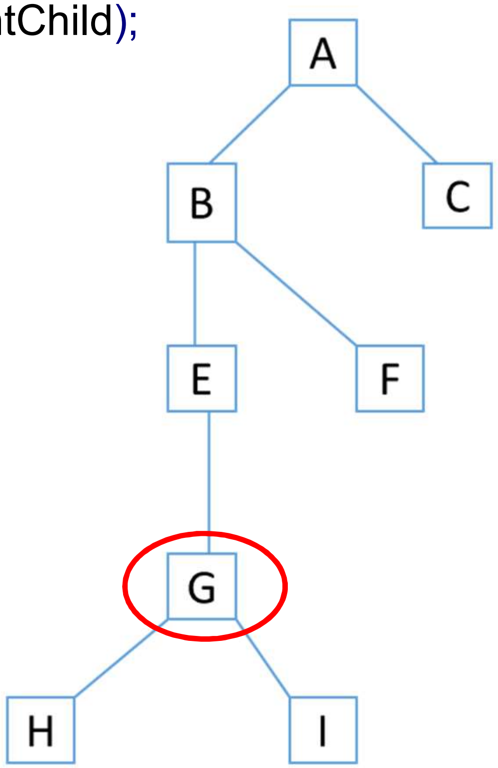


Count = 1 + left + 0

GetNumberOfNodes

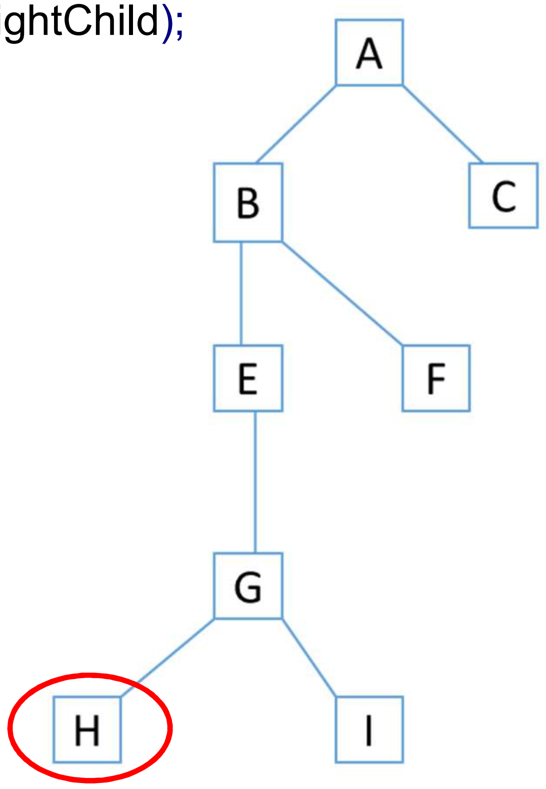
```
public int recursiveGetNumNodes(Node rood) {
        int count = 1;
        if(root.leftChild != null)
                count += recursiveGetNumNodes(root.leftChild);
        if(root.rightChild != null)
                count += recursiveGetNumNodes(root.rightChild);
        return count;
}
```

Count = 1 + left + right



GetNumberOfNodes

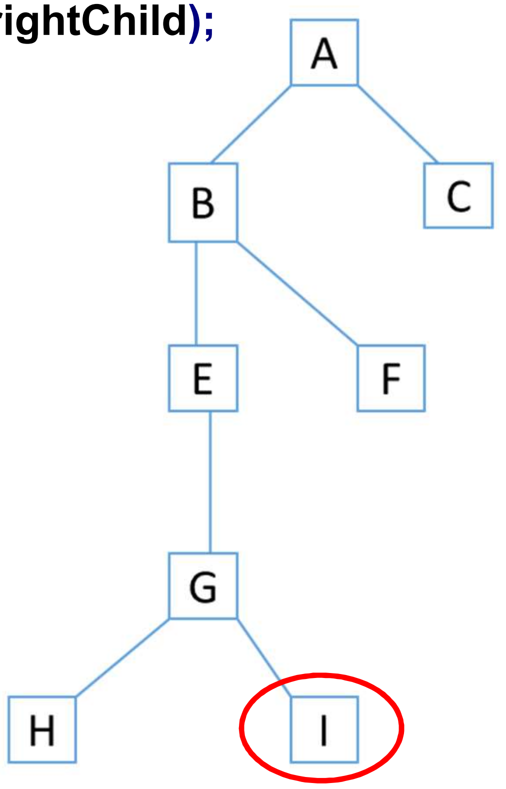
```
public int recursiveGetNumNodes(Node rood) {
        int count = 1;
        if(root.leftChild != null)
                count += recursiveGetNumNodes(root.leftChild);
        if(root.rightChild != null)
                count += recursiveGetNumNodes(root.rightChild);
        return count;
}
```



Count = 1 + 0 + 0

GetNumberOfNodes

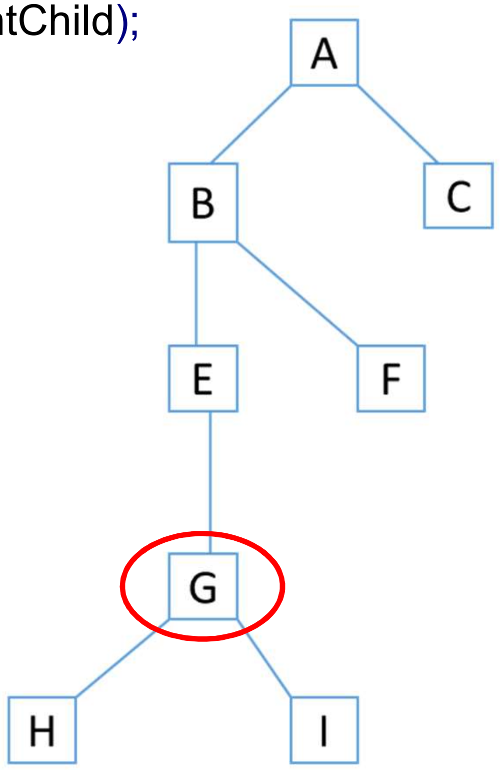
```
public int recursiveGetNumNodes(Node rood) {
        int count = 1;
        if(root.leftChild != null)
                count += recursiveGetNumNodes(root.leftChild);
        if(root.rightChild != null)
                count += recursiveGetNumNodes(root.rightChild);
        return count;
}
```



Count = 1 + 0 + 0

GetNumberOfNodes

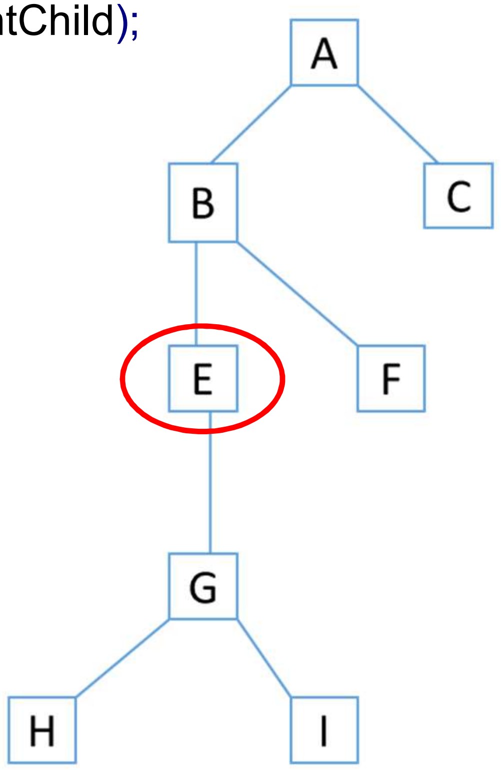
```
public int recursiveGetNumNodes(Node rood) {
        int count = 1;
        if(root.leftChild != null)
                count += recursiveGetNumNodes(root.leftChild);
        if(root.rightChild != null)
                count += recursiveGetNumNodes(root.rightChild);
        return count;
}
```



Count = 1 + 1 + 1 = 3

GetNumberOfNodes

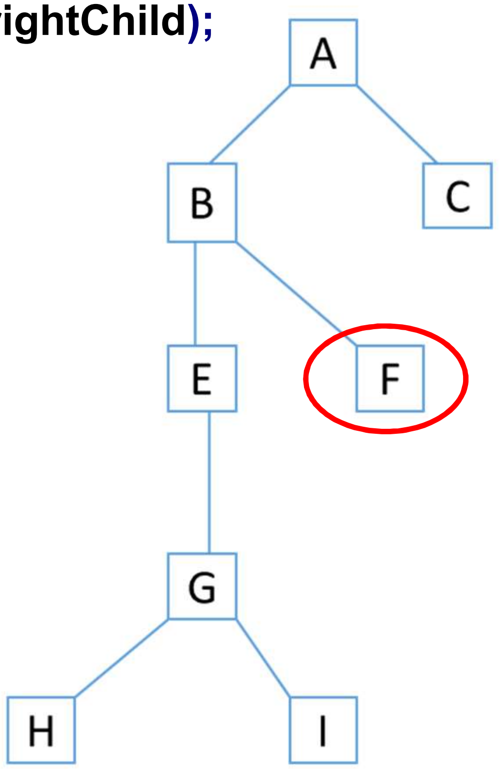
```
public int recursiveGetNumNodes(Node rood) {
        int count = 1;
        if(root.leftChild != null)
                count += recursiveGetNumNodes(root.leftChild);
        if(root.rightChild != null)
                count += recursiveGetNumNodes(root.rightChild);
        return count;
}
```



Count = 1 + 3 + 0 = 4

GetNumberOfNodes

```
public int recursiveGetNumNodes(Node rood) {
        int count = 1;
        if(root.leftChild != null)
                count += recursiveGetNumNodes(root.leftChild);
        if(root.rightChild != null)
                count += recursiveGetNumNodes(root.rightChild);
        return count;
}
```

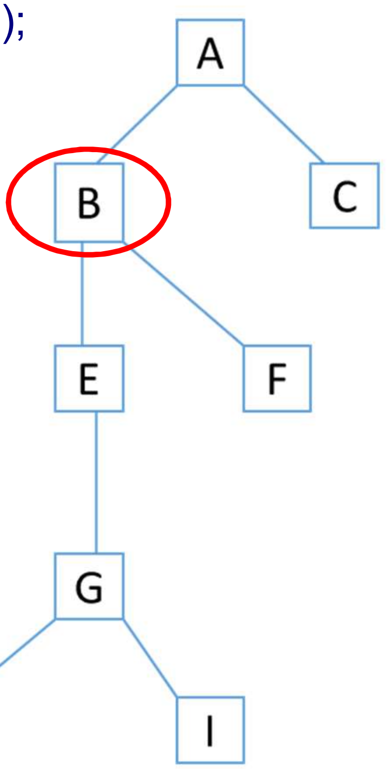


Count = 1 + 0 + 0 = 1

GetNumberOfNodes

```
public int recursiveGetNumNodes(Node rood) {
        int count = 1;
        if(root.leftChild != null)
                count += recursiveGetNumNodes(root.leftChild);
        if(root.rightChild != null)
                count += recursiveGetNumNodes(root.rightChild);
        return count;
}
```

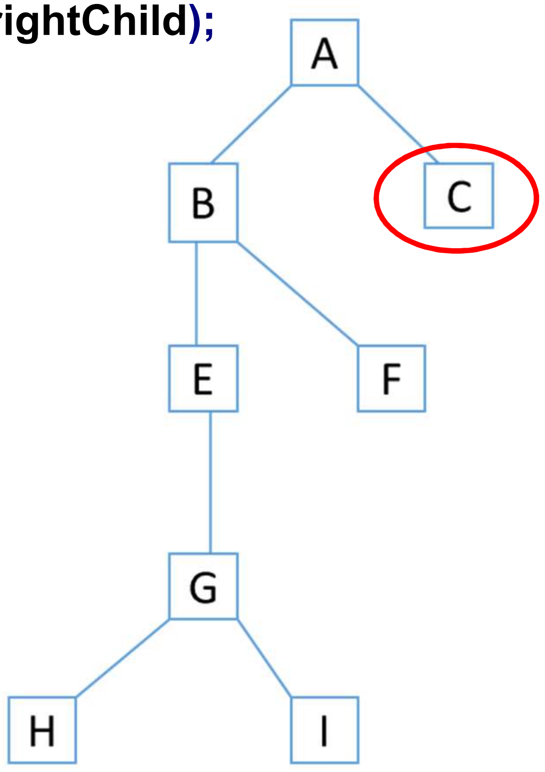
Count = 1 + 4 + 1 = 6



GetNumberOfNodes

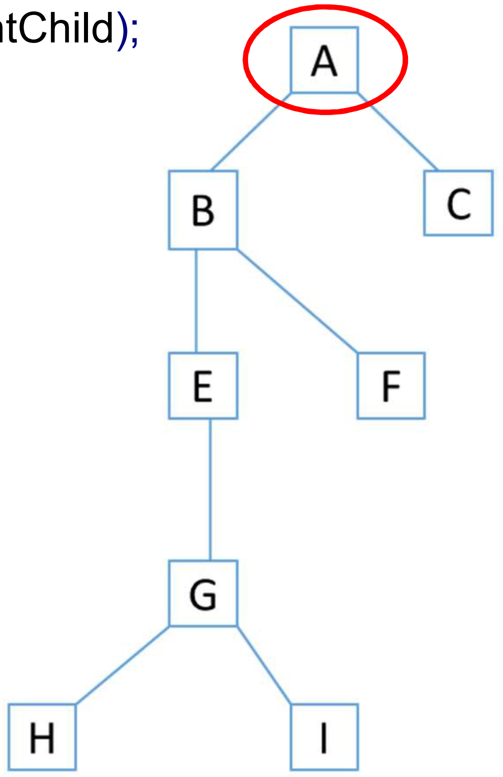
```
public int recursiveGetNumNodes(Node rood) {
            int count = 1;
            if(root.leftChild != null)
                        count += recursiveGetNumNodes(root.leftChild);
            if(root.rightChild != null)
                        count += recursiveGetNumNodes(root.rightChild);
            return count;
}
```

Count = 1 + 0 + 0 = 1



GetNumberOfNodes

```
public int recursiveGetNumNodes(Node rood) {
            int count = 1;
            if(root.leftChild != null)
                        count += recursiveGetNumNodes(root.leftChild);
            if(root.rightChild != null)
                        count += recursiveGetNumNodes(root.rightChild);
            return count;
 }
```

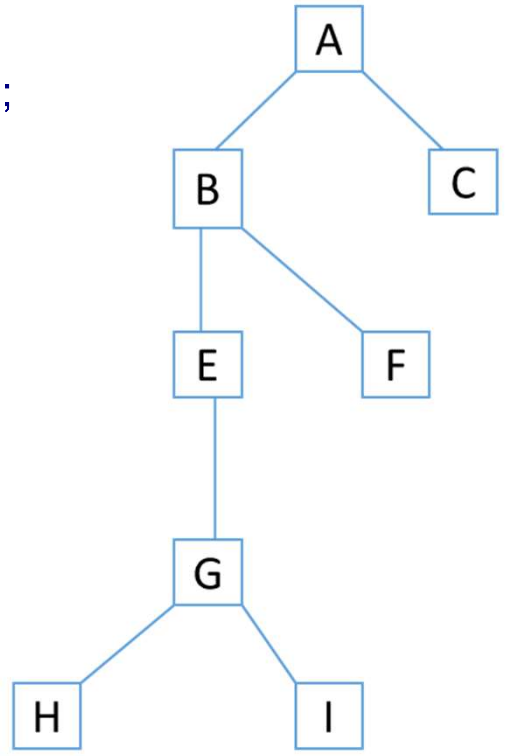


Count = 1 + 6 + 1 = 8

GetNumberOfNodes

```
public int getNumNodes(BinaryTree tree) {
        //Call a recursive helper method
        int count = 0;
        if(tree.root != null)
                count = recursiveGetNumNodes(tree.root);
        return count;
}
```

Count = 8



Traversal

Preorder:

1. Visit the parent node
2. Visit the left child (subtree)
3. Visit the right child (subtree)

Postorder:

1. Visit the left child (subtree)
2. Visit the right child (subtree)
3. Visit the parent node

Inorder:

1. Visit the left child (subtree)
2. Visit the parent node
3. Visit the right child (subtree)

Note: The following animations are demonstrations of determining tree traversals by hand.

In a real implementation, traversals are conducted using stacks and queues.

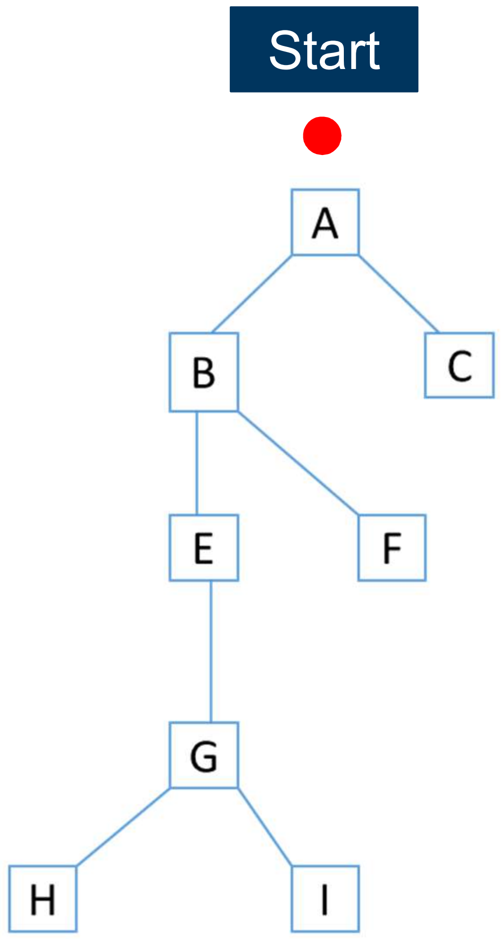
Would an Inorder traversal be possible on a ternary tree?

Preorder Traversal

```
preOrder(Node v)
    visit(v)
    for each child w of v
        preorder(w)
```

Add a node to the traversal when touching its left side

Traversal:

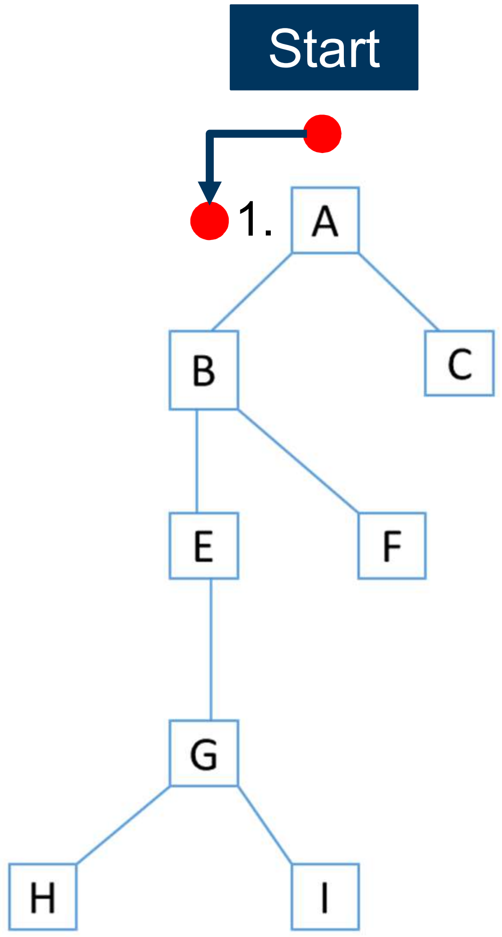


Preorder Traversal

```
preOrder(Node v)
    visit(v)
    for each child w of v
        preorder(w)
```

Add a node to the traversal when touching its left side

Traversal: A

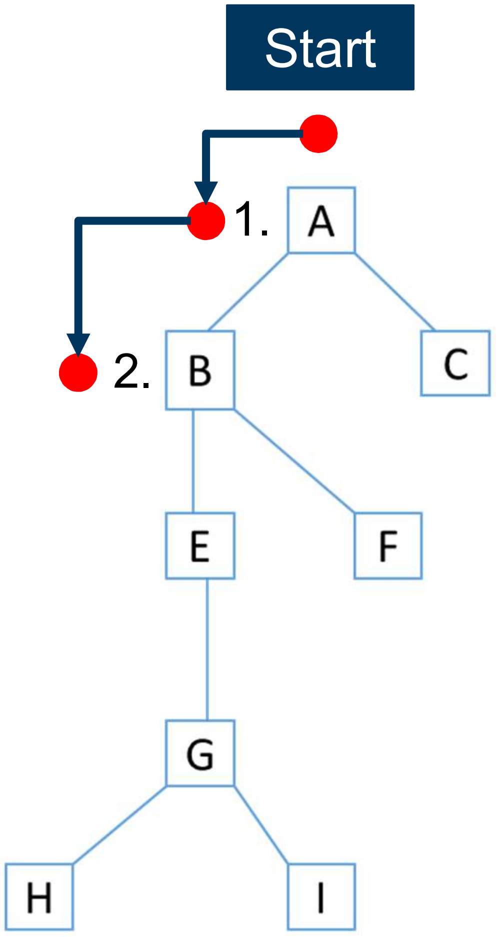


Preorder Traversal

```
preOrder(Node v)
    visit(v)
    for each child w of v
        preorder(w)
```

Add a node to the traversal when touching its left side

Traversal: AB

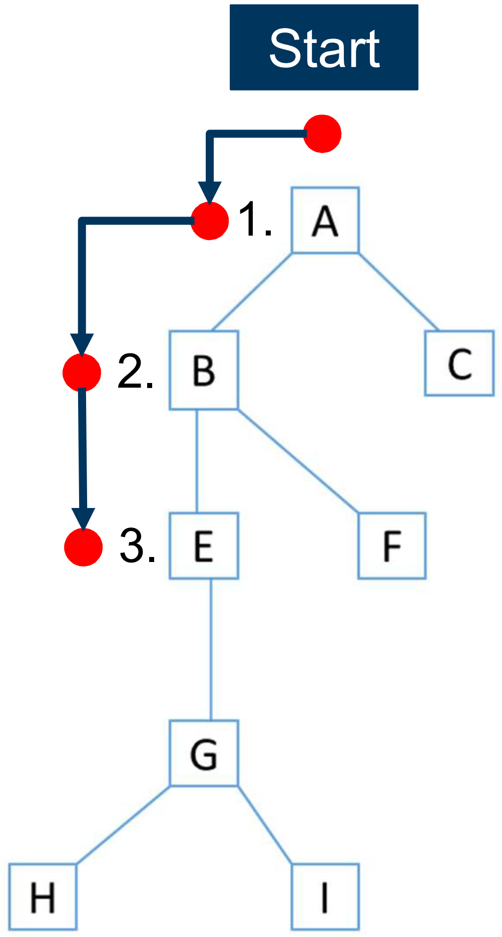


Preorder Traversal

```
preOrder(Node v)
     visit(v)
     for each child w of v
          preorder(w)
```

Add a node to the traversal when touching its left side

Traversal: ABE

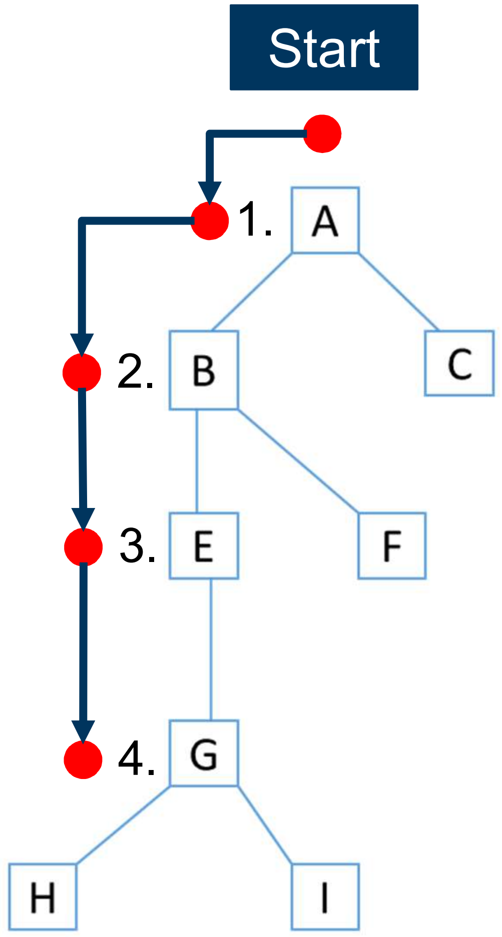


Preorder Traversal

```
preOrder(Node v)
        visit(v)
        for each child w of v
                preorder(w)
```

Add a node to the traversal when touching its left side

Traversal: ABEG

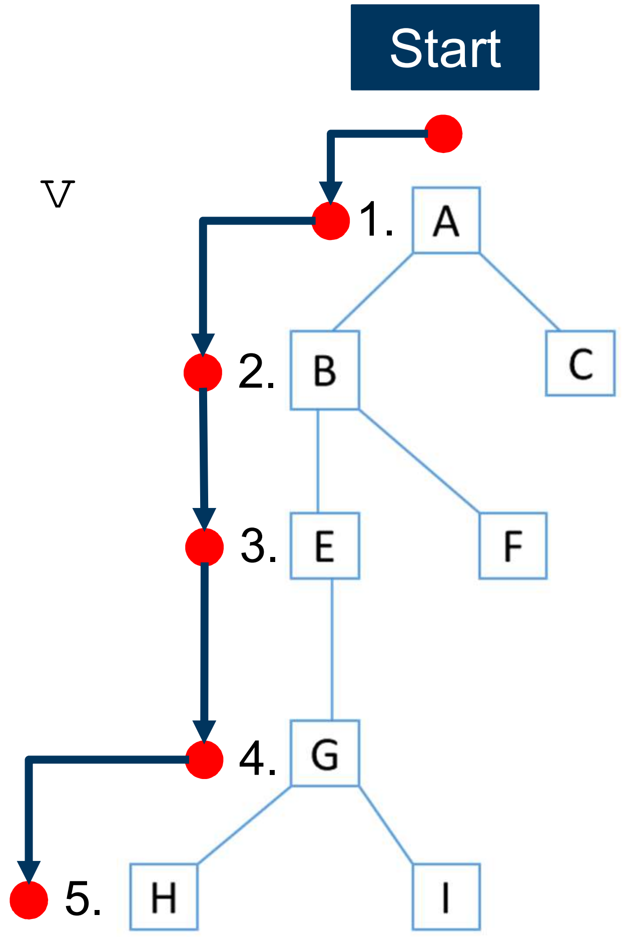


Preorder Traversal

```
preOrder(Node v)
     visit(v)
     for each child w of v
          preorder(w)
```

Add a node to the traversal when touching its left side

Traversal: ABEGH

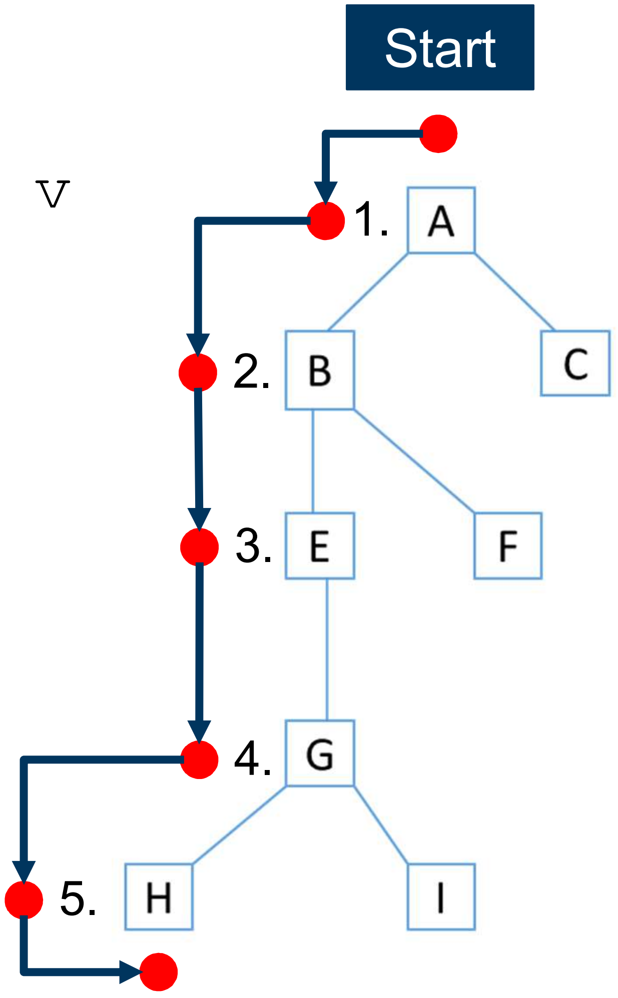


Preorder Traversal

```
preOrder(Node v)
     visit(v)
     for each child w of v
          preorder(w)
```

Add a node to the traversal when touching its left side

Traversal: ABEGH

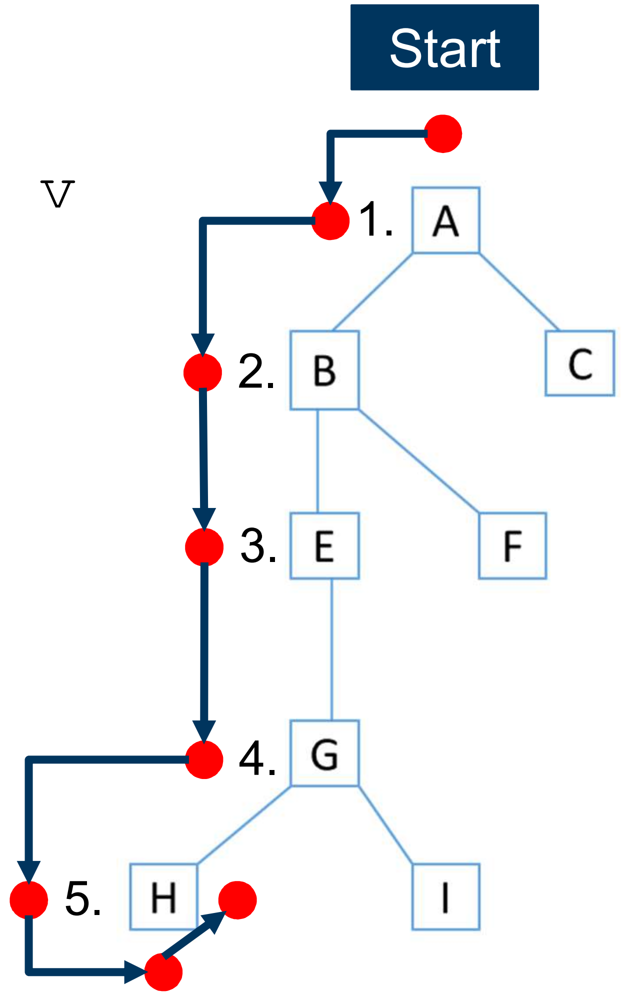


Preorder Traversal

```
preOrder(Node v)
     visit(v)
     for each child w of v
          preorder(w)
```

Add a node to the traversal when touching its left side

Traversal: ABEGH

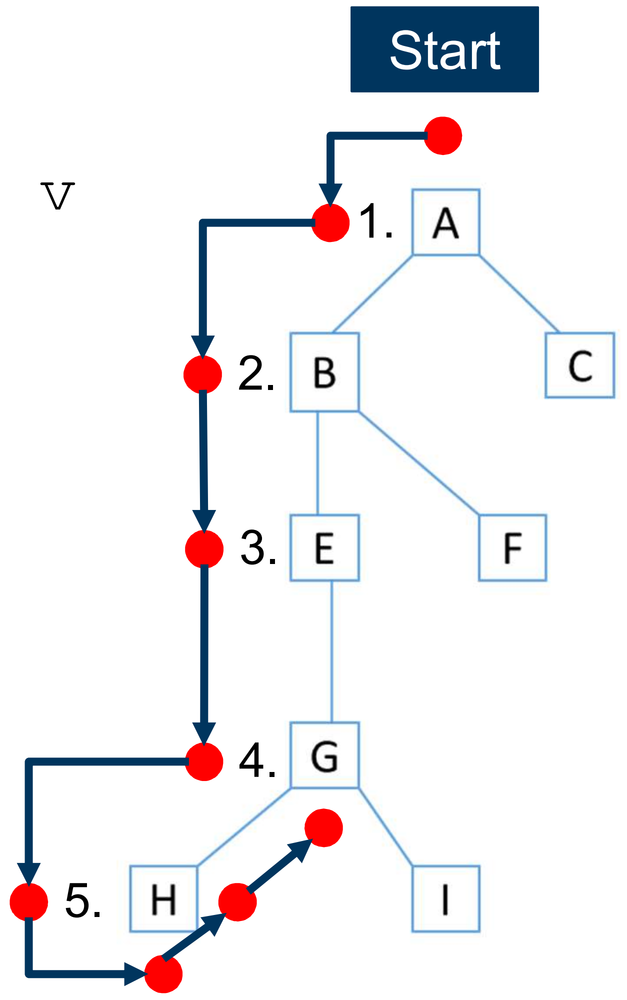


Preorder Traversal

```
preOrder(Node v)
    visit(v)
    for each child w of v
        preorder(w)
```

Add a node to the traversal when touching its left side

Traversal: ABEGH

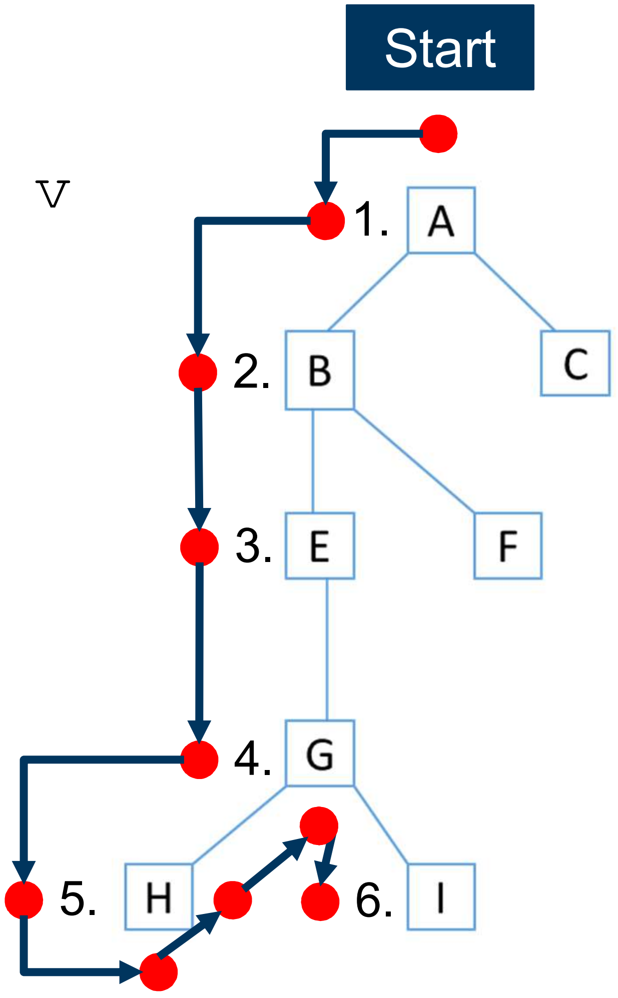


Preorder Traversal

```
preOrder(Node v)
    visit(v)
    for each child w of v
        preorder(w)
```

Add a node to the traversal when touching its left side

Traversal: ABEGHI

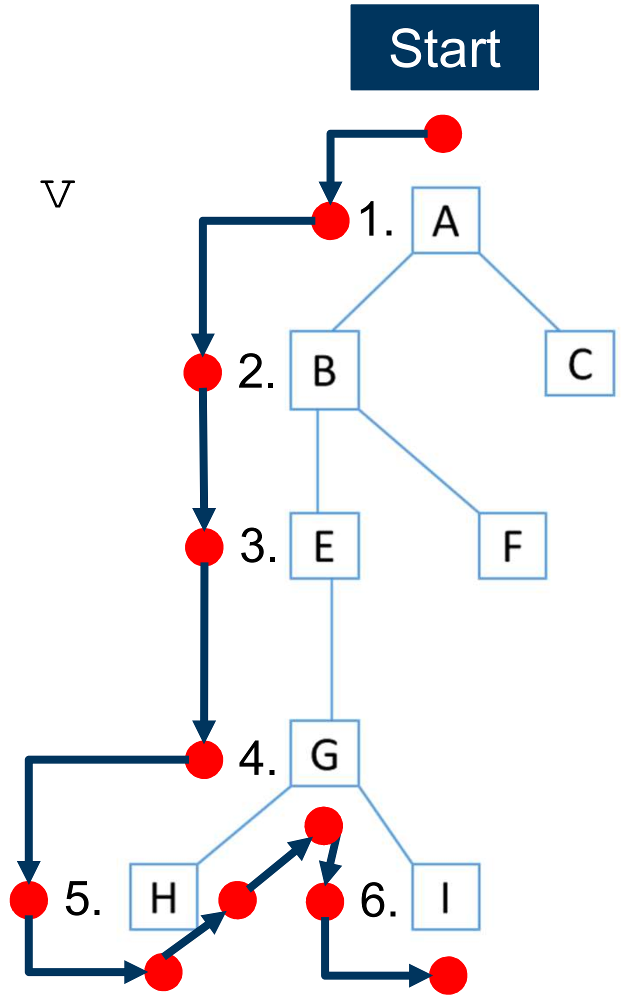


Preorder Traversal

```
preOrder(Node v)
    visit(v)
    for each child w of v
        preorder(w)
```

Add a node to the traversal when touching its left side

Traversal: ABEGHI

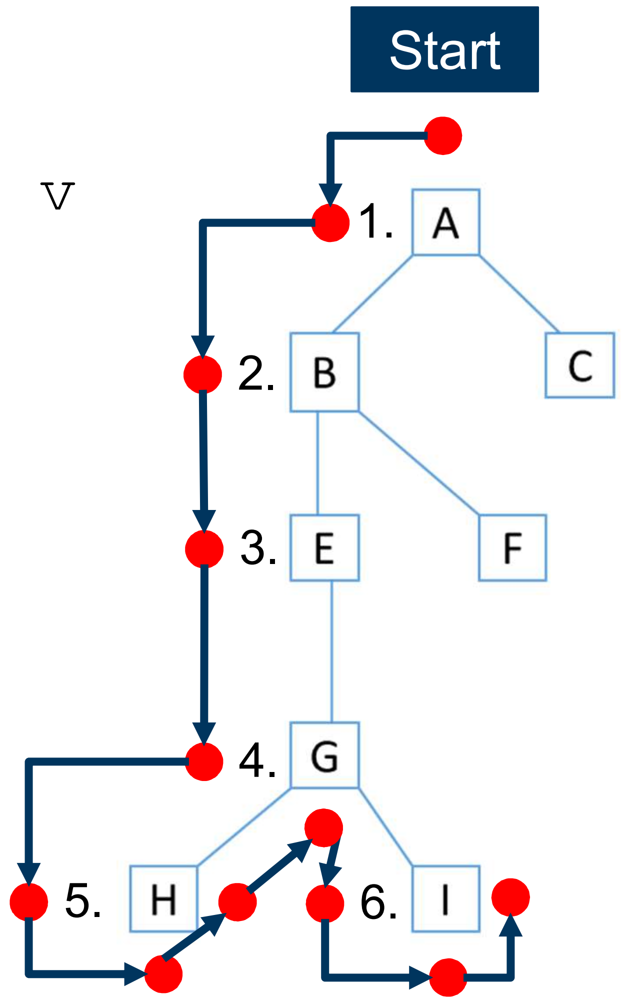


Preorder Traversal

```
preOrder(Node v)
    visit(v)
    for each child w of v
        preorder(w)
```

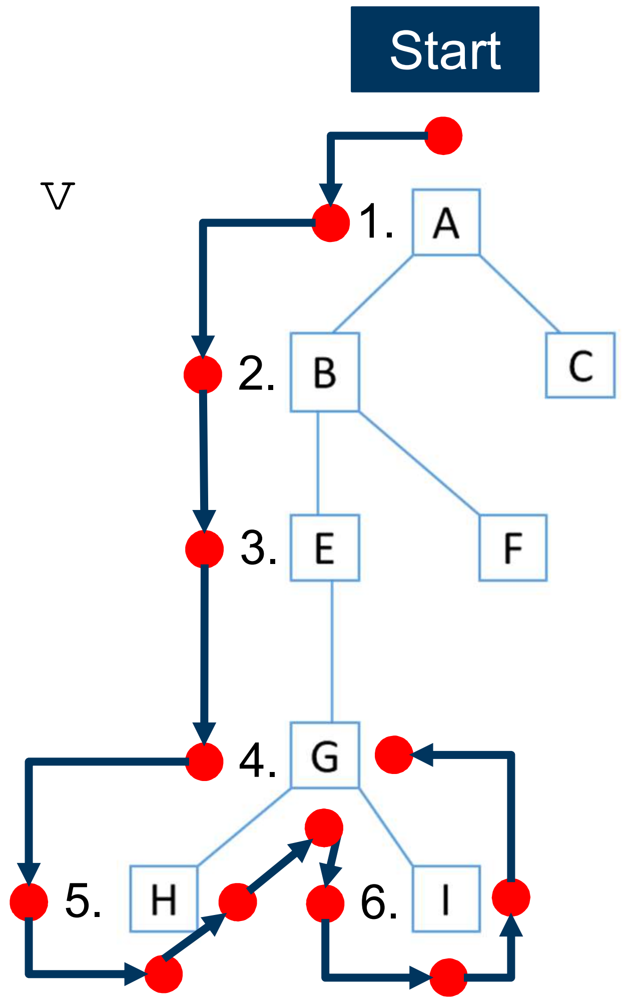
Add a node to the traversal when touching its left side

Traversal: ABEGHI



Preorder Traversal

```
preOrder(Node v)
    visit(v)
    for each child w of v
        preorder(w)
```



Add a node to the traversal when touching its left side

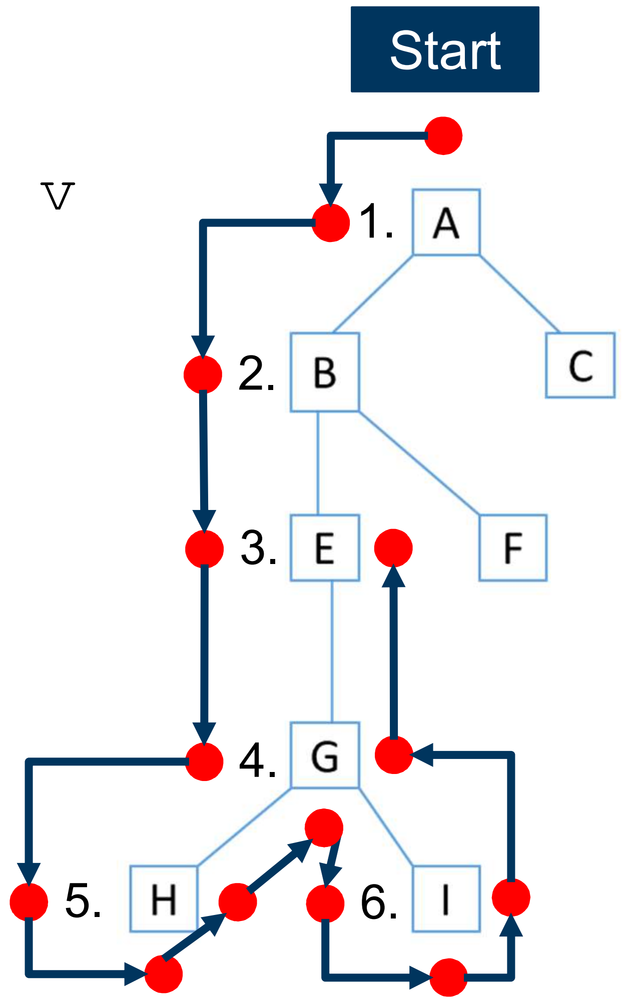
Traversal: ABEGHI

Preorder Traversal

```
preOrder(Node v)
    visit(v)
    for each child w of v
        preorder(w)
```

Add a node to the traversal when touching its left side

Traversal: ABEGHI

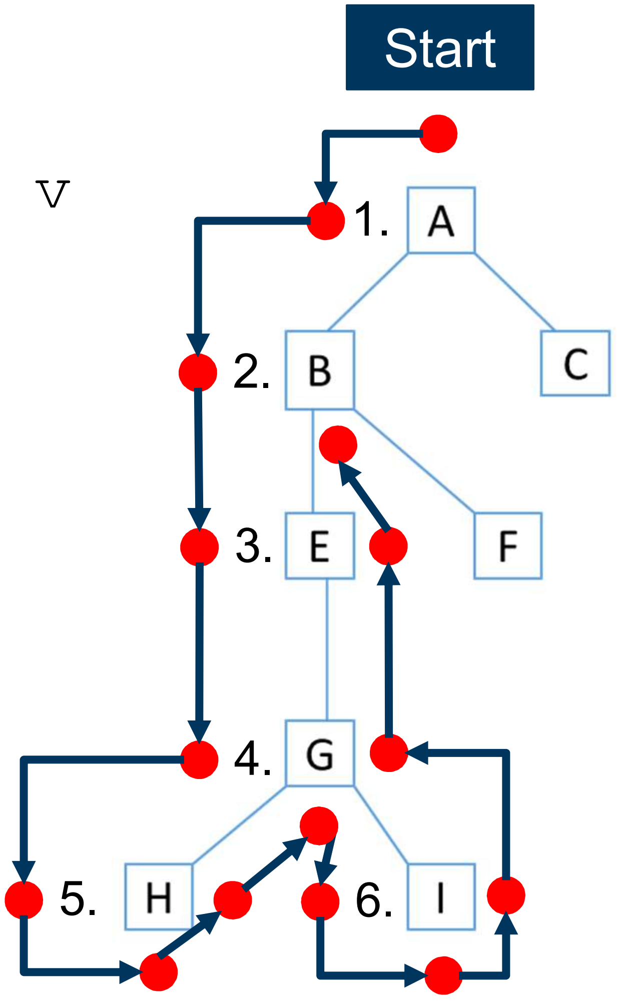


Preorder Traversal

```
preOrder(Node v)
     visit(v)
     for each child w of v
          preorder(w)
```

Add a node to the traversal when touching its left side

Traversal: ABEGHI

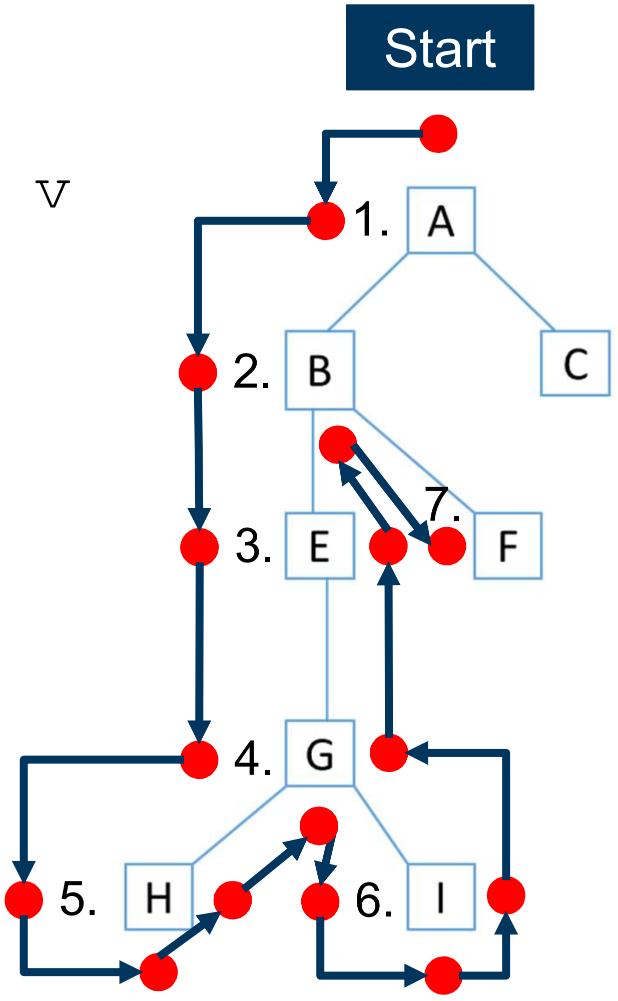


Preorder Traversal

```
preOrder(Node v)
          visit(v)
          for each child w of v
                    preorder(w)
```

Add a node to the traversal when touching its left side

Traversal: ABEGHIF

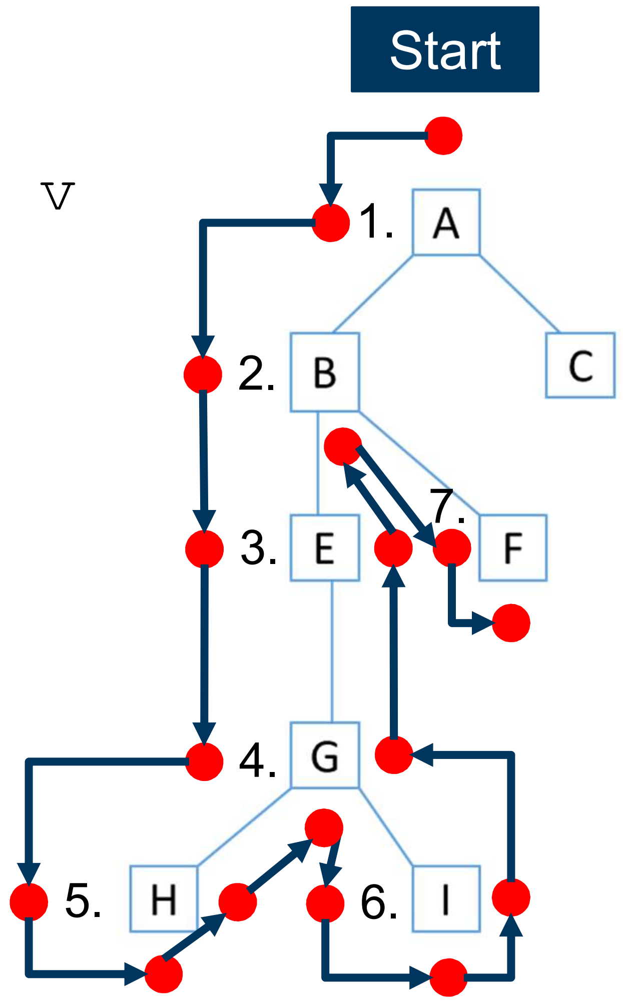


Preorder Traversal

```
preOrder(Node v)
    visit(v)
    for each child w of v
        preorder(w)
```

Add a node to the traversal when touching its left side

Traversal: ABEGHIF

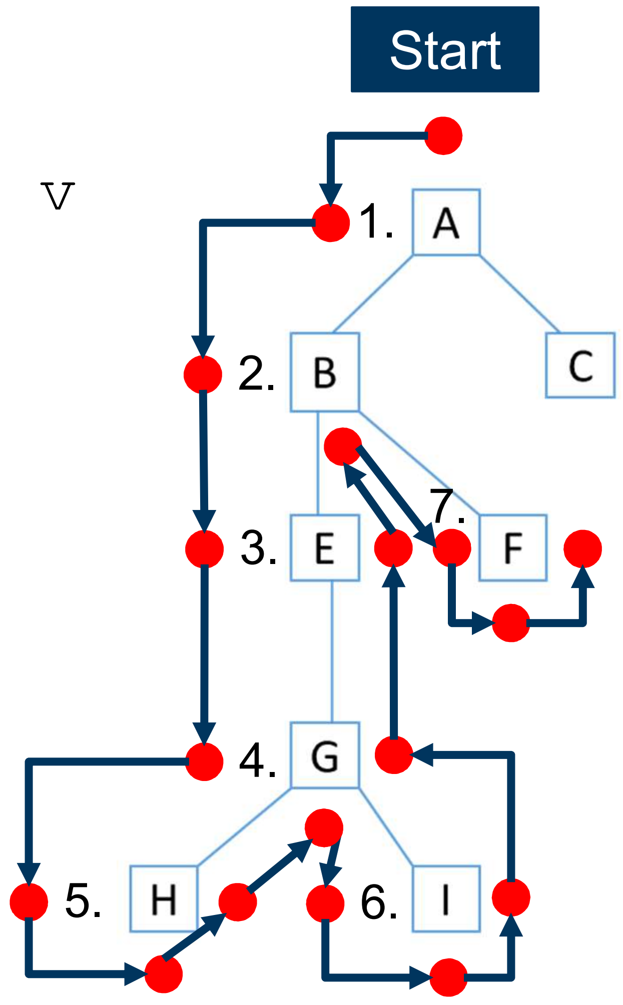


Preorder Traversal

```
preOrder(Node v)
    visit(v)
    for each child w of v
        preorder(w)
```

Add a node to the traversal when touching its left side

Traversal: ABEGHIF

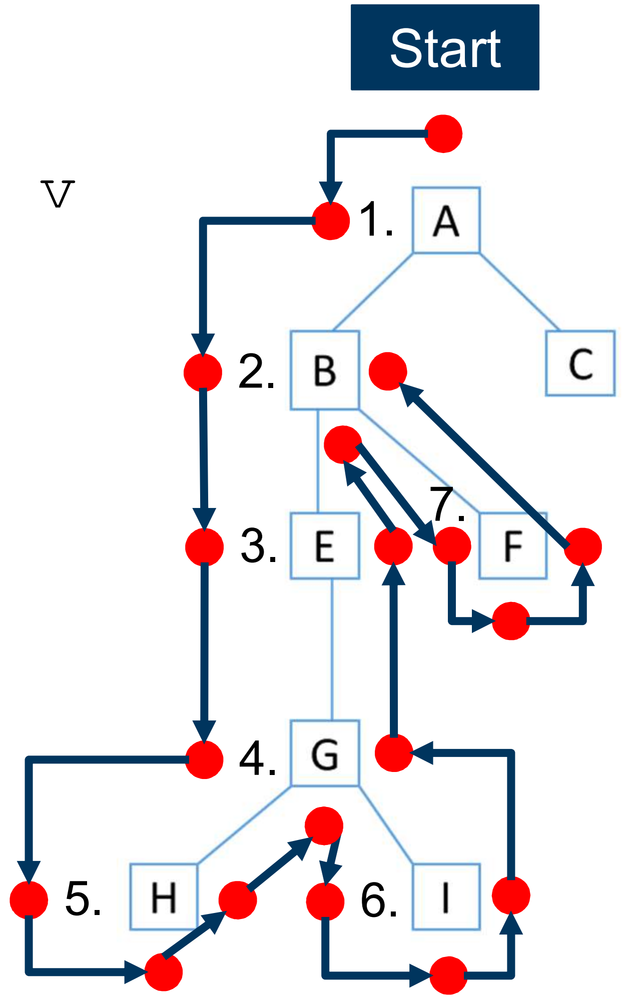


Preorder Traversal

```
preOrder(Node v)
    visit(v)
    for each child w of v
        preorder(w)
```

Add a node to the traversal when touching its left side

Traversal: ABEGHIF

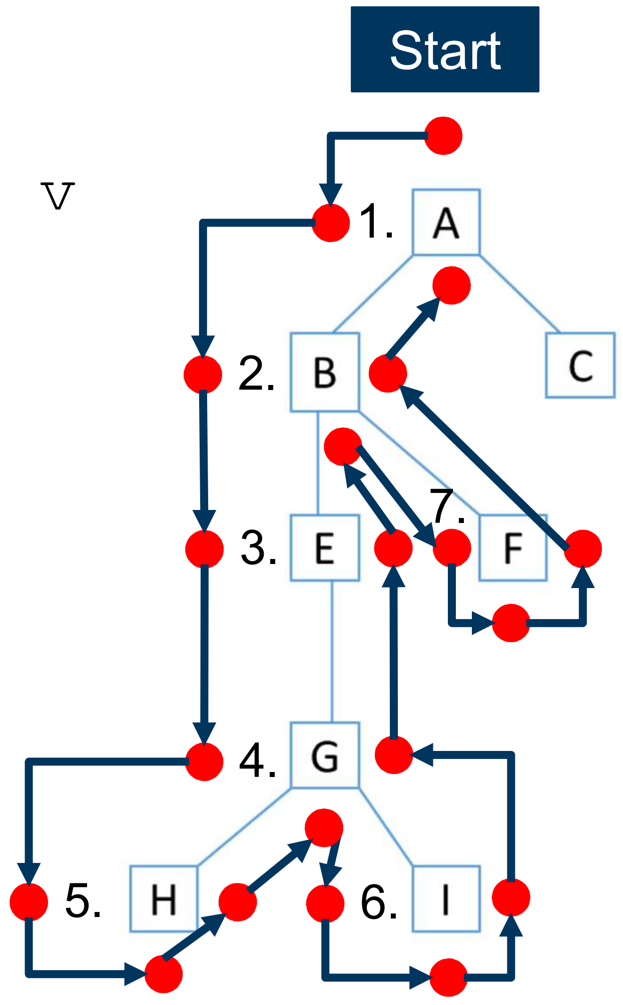


Preorder Traversal

```
preOrder(Node v)
    visit(v)
    for each child w of v
        preorder(w)
```

Add a node to the traversal when touching its left side

Traversal: ABEGHIF

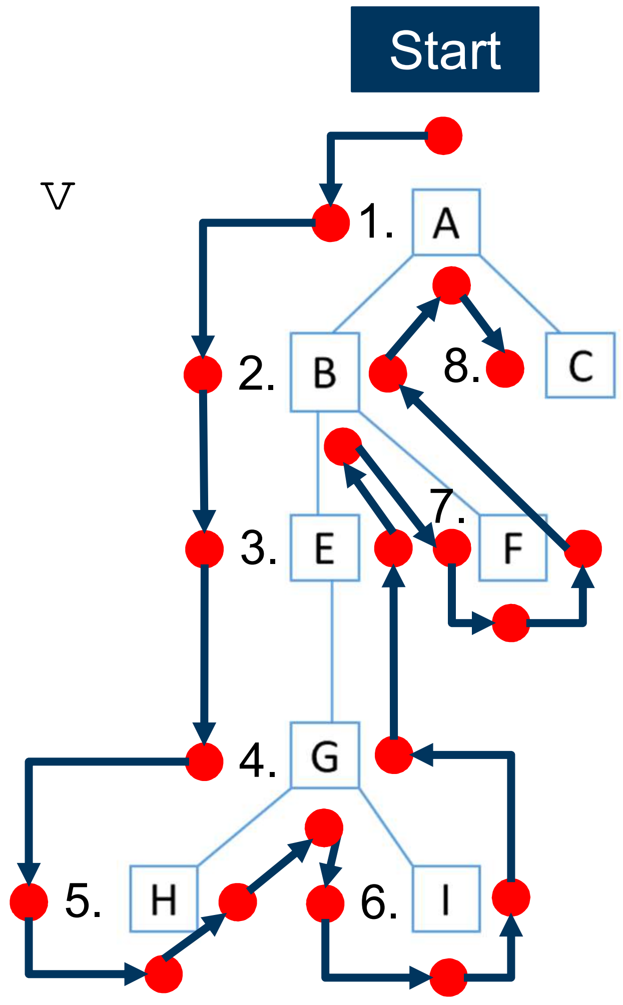


Preorder Traversal

```
preOrder(Node v)
     visit(v)
     for each child w of v
          preorder(w)
```

Add a node to the traversal when touching its left side

Traversal: ABEGHIFC

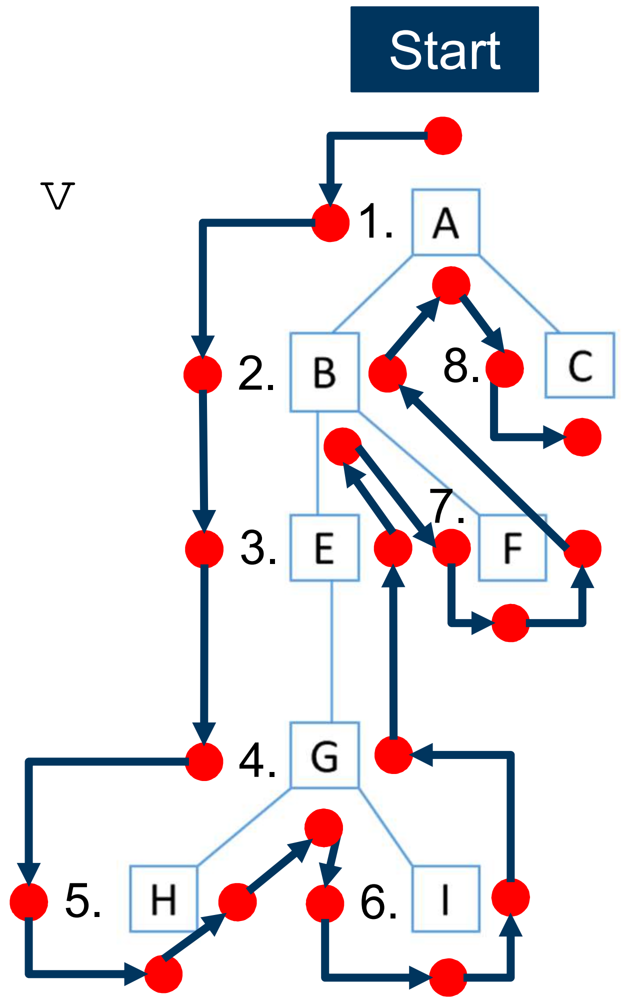


Preorder Traversal

```
preOrder(Node v)
    visit(v)
    for each child w of v
        preorder(w)
```

Add a node to the traversal when touching its left side

Traversal: ABEGHIFC

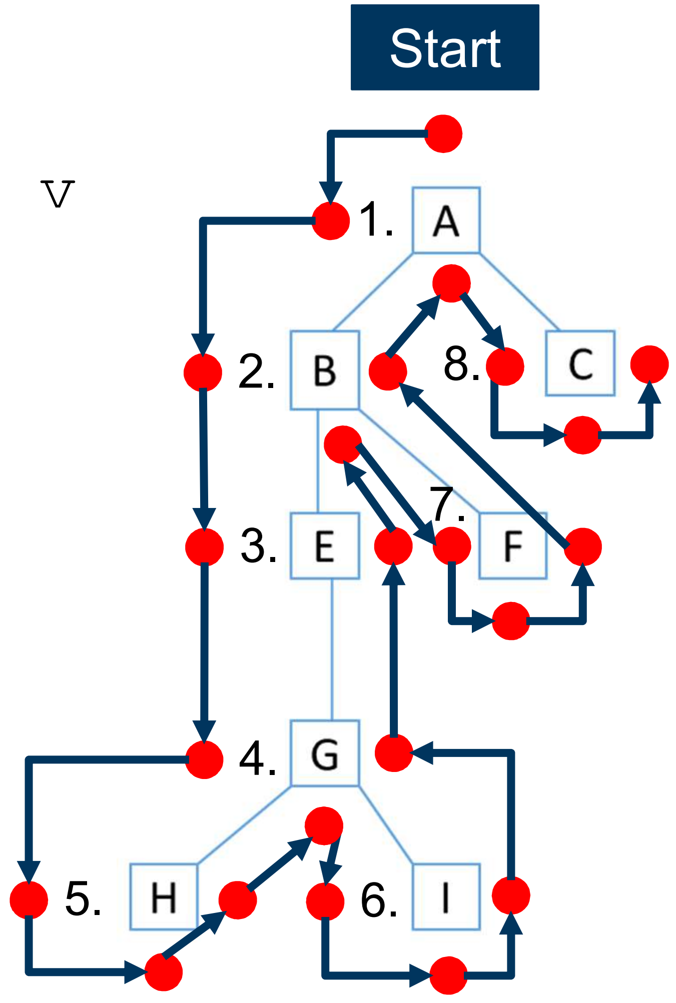


Preorder Traversal

```
preOrder(Node v)
    visit(v)
    for each child w of v
        preorder(w)
```

Add a node to the traversal when touching its left side

Traversal: ABEGHIFC

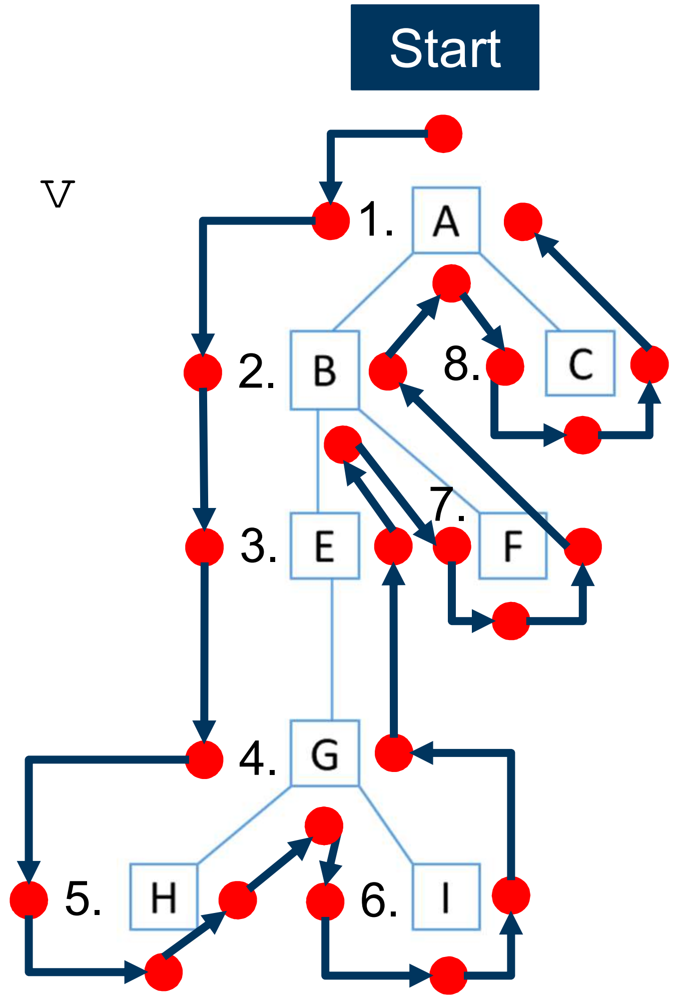


Preorder Traversal

```
preOrder(Node v)
    visit(v)
    for each child w of v
        preorder(w)
```

Add a node to the traversal when touching its left side

Traversal: ABEGHIFC

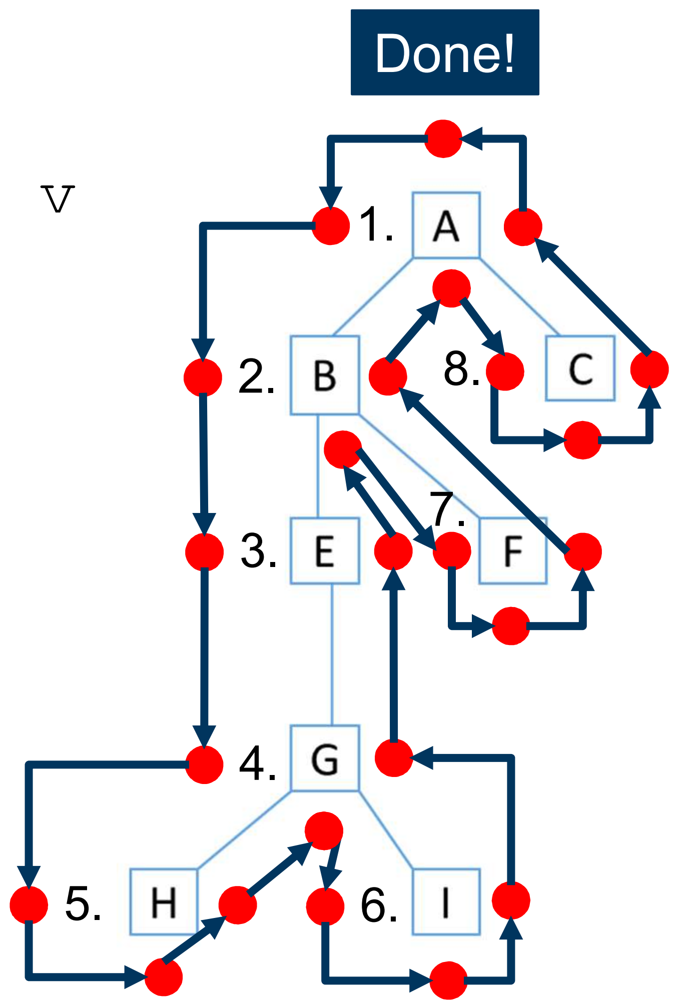


Preorder Traversal

```
preOrder(Node v)
     visit(v)
     for each child w of v
          preorder(w)
```

Add a node to the traversal when touching its left side

Traversal: ABEGHIFC

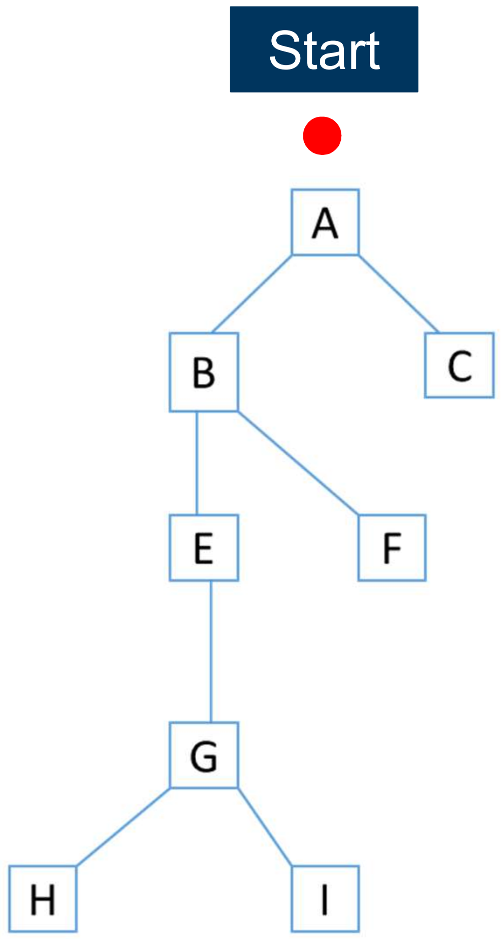


Postorder Traversal

```
postOrder(Node v)
    for each child w of v
        postOrder(w)
    visit(v)
```

Add a node to the traversal when touching its right side

Traversal:

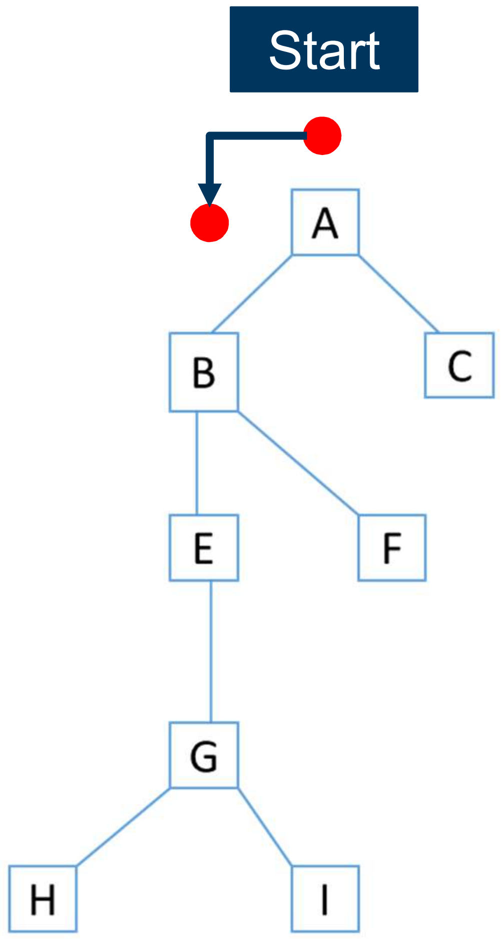


Postorder Traversal

```
postOrder(Node v)
      for each child w of v
            postOrder(w)
      visit(v)
```

Add a node to the traversal when touching its right side

Traversal:

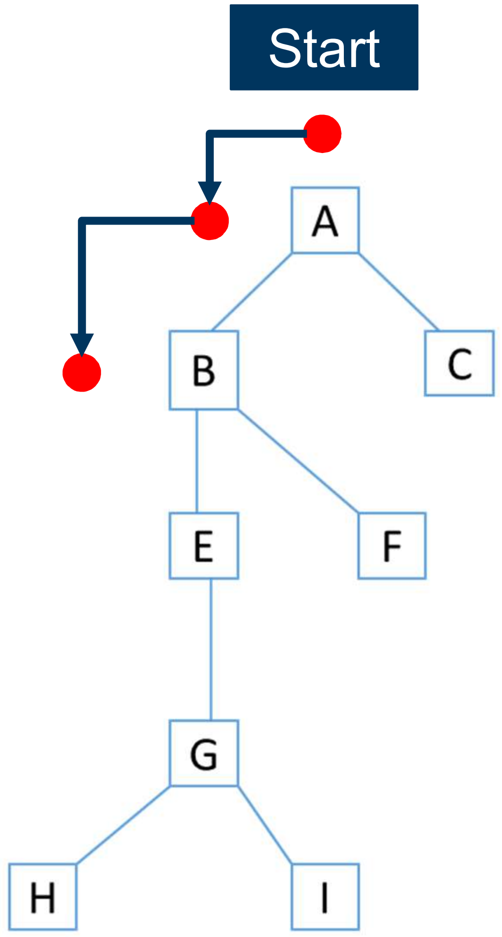


Postorder Traversal

```
postOrder(Node v)
        for each child w of v
                postOrder(w)
        visit(v)
```

Add a node to the traversal when touching its right side

Traversal:

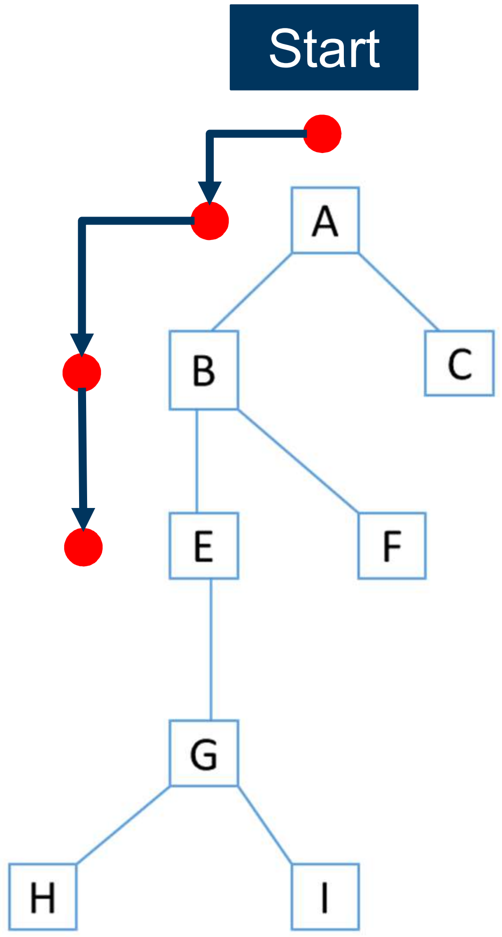


Postorder Traversal

```
postOrder(Node v)
    for each child w of v
        postOrder(w)
    visit(v)
```

Add a node to the traversal when touching its right side

Traversal:

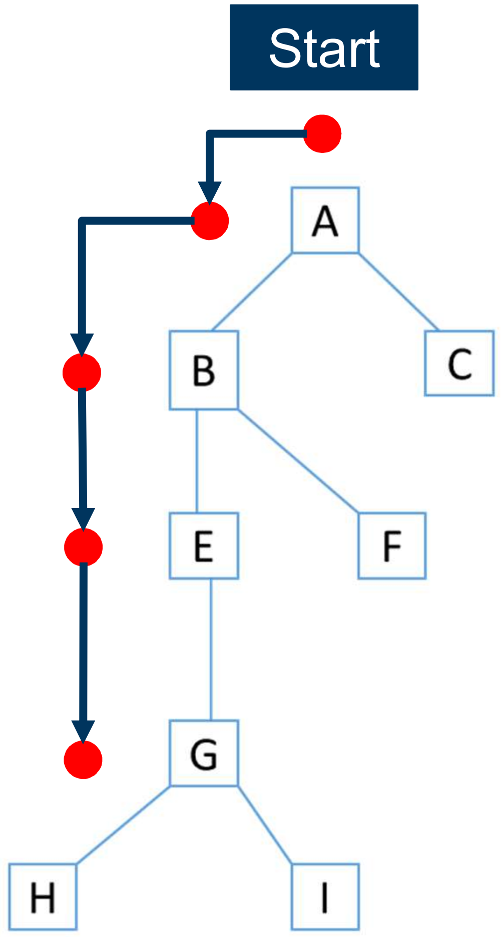


Postorder Traversal

```
postOrder(Node v)
    for each child w of v
        postOrder(w)
    visit(v)
```

Add a node to the traversal when touching its right side

Traversal:

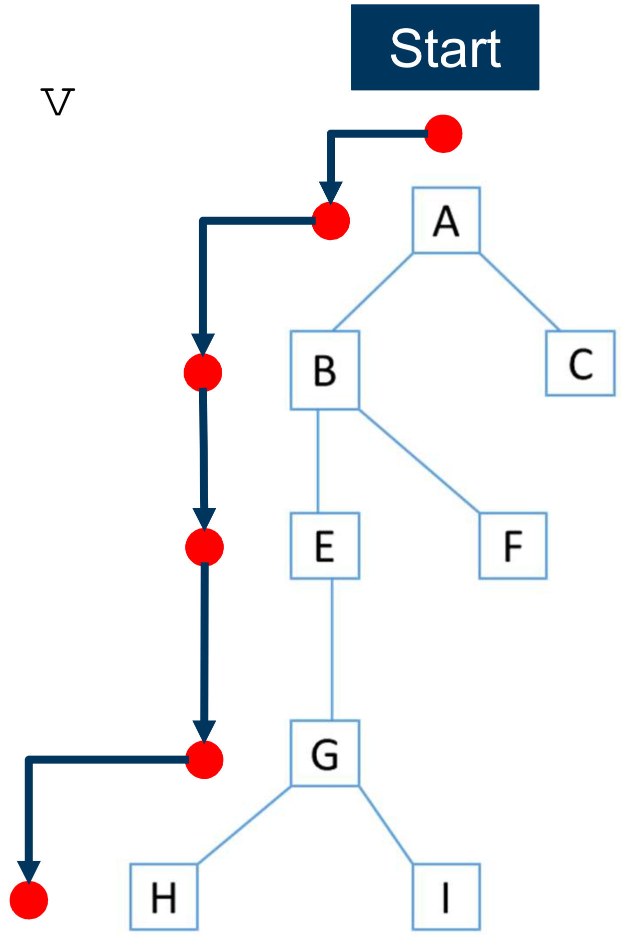


Postorder Traversal

```
postOrder(Node v)
        for each child w of v
                postOrder(w)
        visit(v)
```

Add a node to the traversal when touching its right side

Traversal:

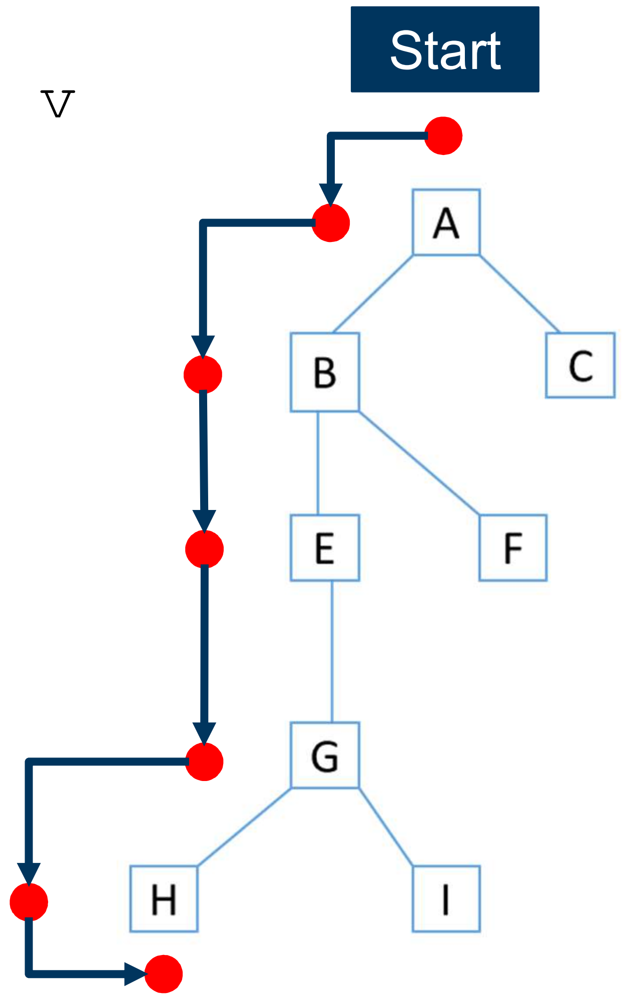


Postorder Traversal

```
postOrder(Node v)
        for each child w of v
                postOrder(w)
        visit(v)
```

Add a node to the traversal when touching its right side

Traversal:

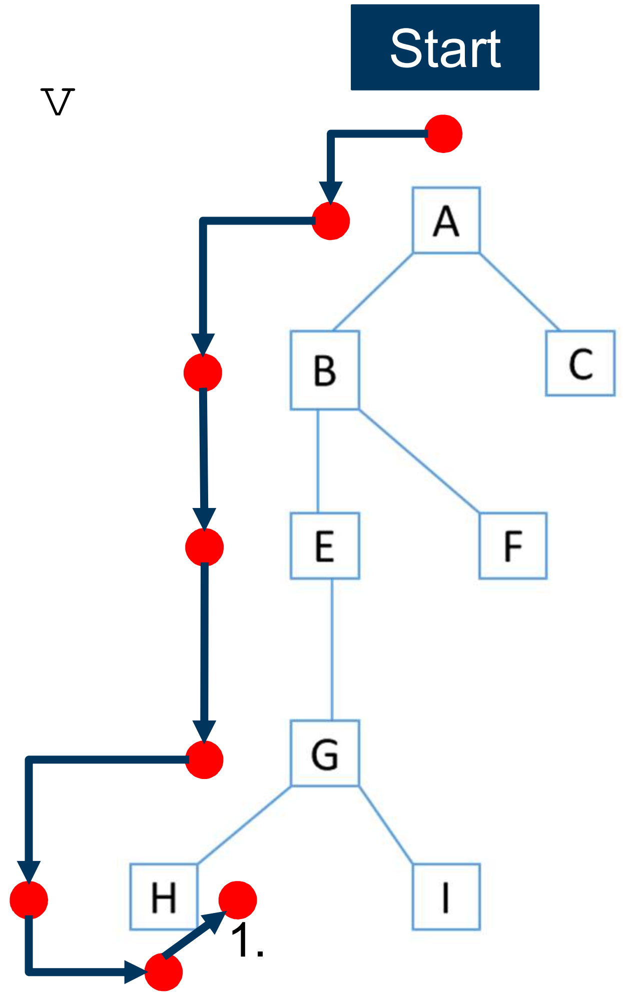


Postorder Traversal

```
postOrder(Node v)
    for each child w of v
        postOrder(w)
    visit(v)
```

Add a node to the traversal when touching its right side

Traversal: H

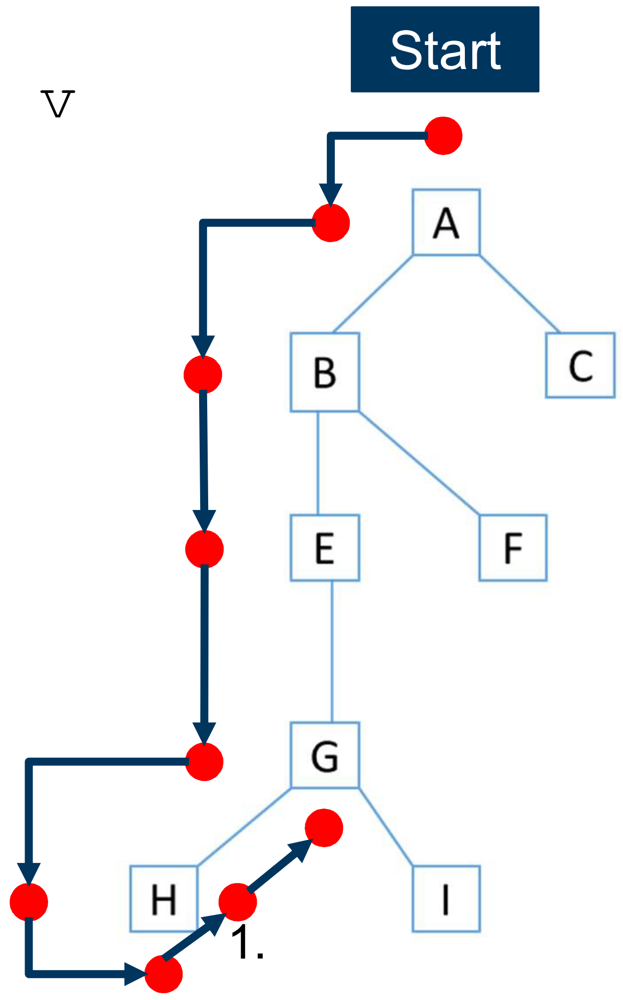


Postorder Traversal

```
postOrder(Node v)
    for each child w of v
        postOrder(w)
    visit(v)
```

Add a node to the traversal when touching its right side

Traversal: H

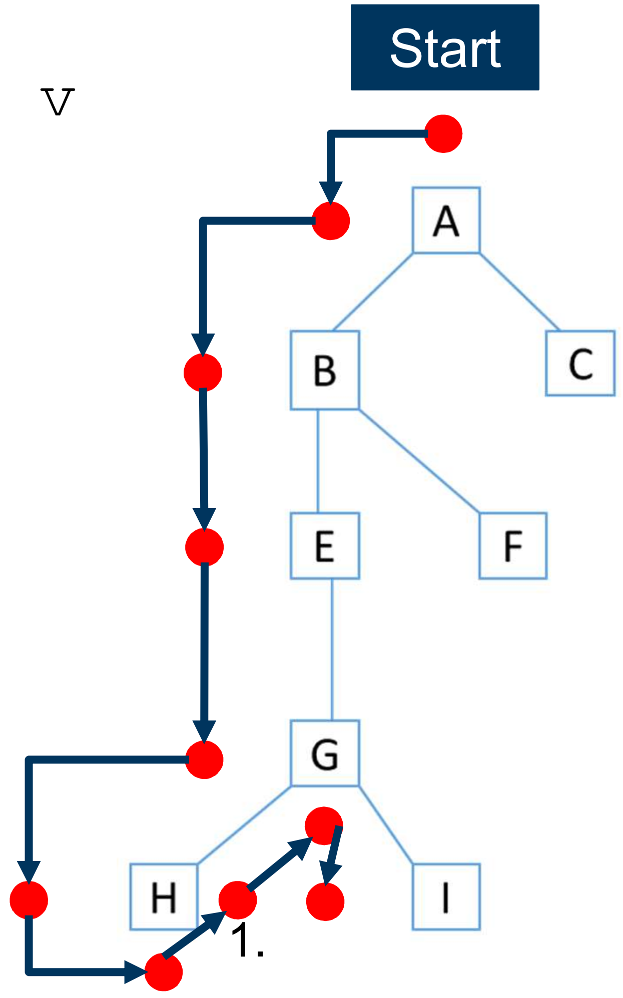


Postorder Traversal

```
postOrder(Node v)
      for each child w of v
            postOrder(w)
      visit(v)
```

Add a node to the traversal when touching its right side

Traversal: H

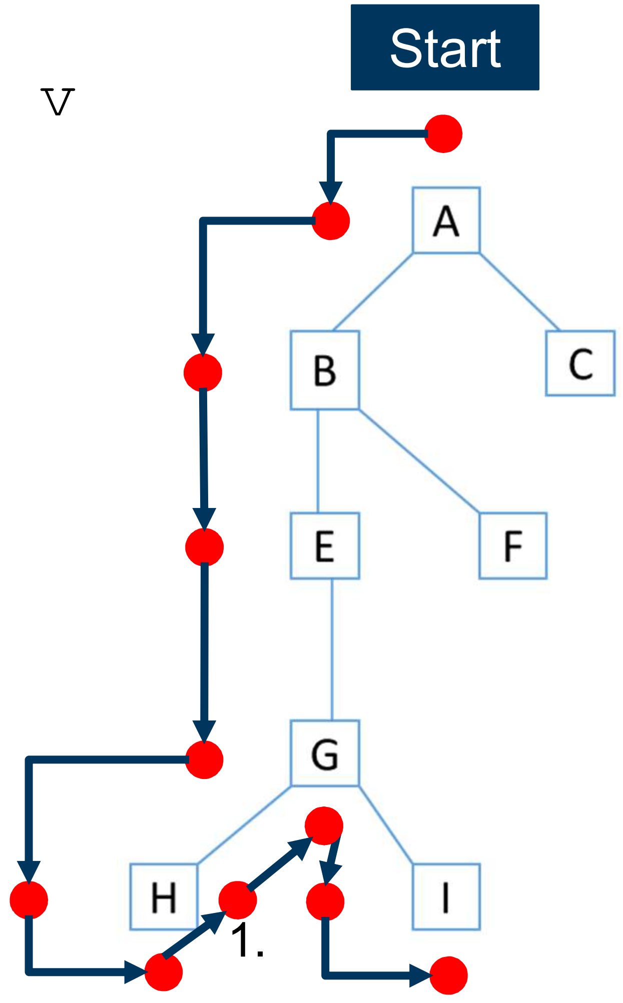


Postorder Traversal

```
postOrder(Node v)
        for each child w of v
                postOrder(w)
        visit(v)
```

Add a node to the traversal when touching its right side

Traversal: H

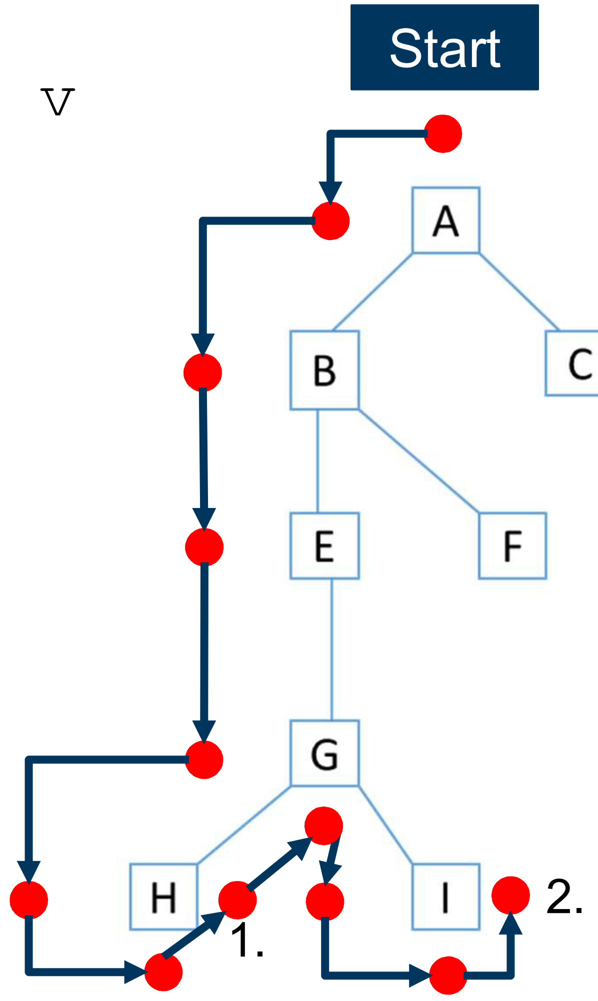


Postorder Traversal

```
postOrder(Node v)
        for each child w of v
                postOrder(w)
        visit(v)
```

Add a node to the traversal when touching its right side

Traversal: HI

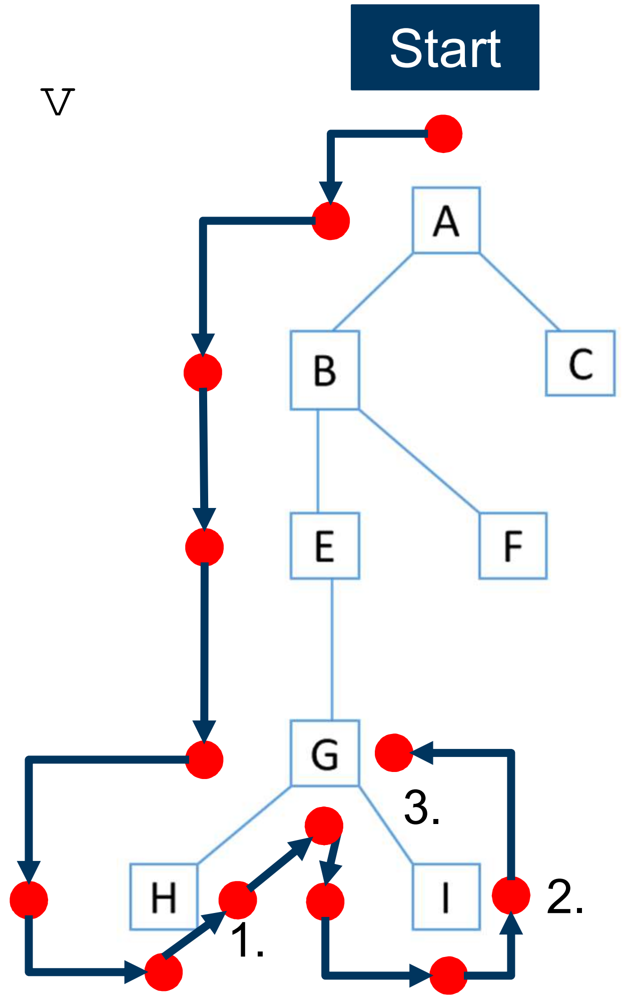


Postorder Traversal

```
postOrder(Node v)
        for each child w of v
                postOrder(w)
        visit(v)
```

Add a node to the traversal when touching its right side

Traversal: HIG

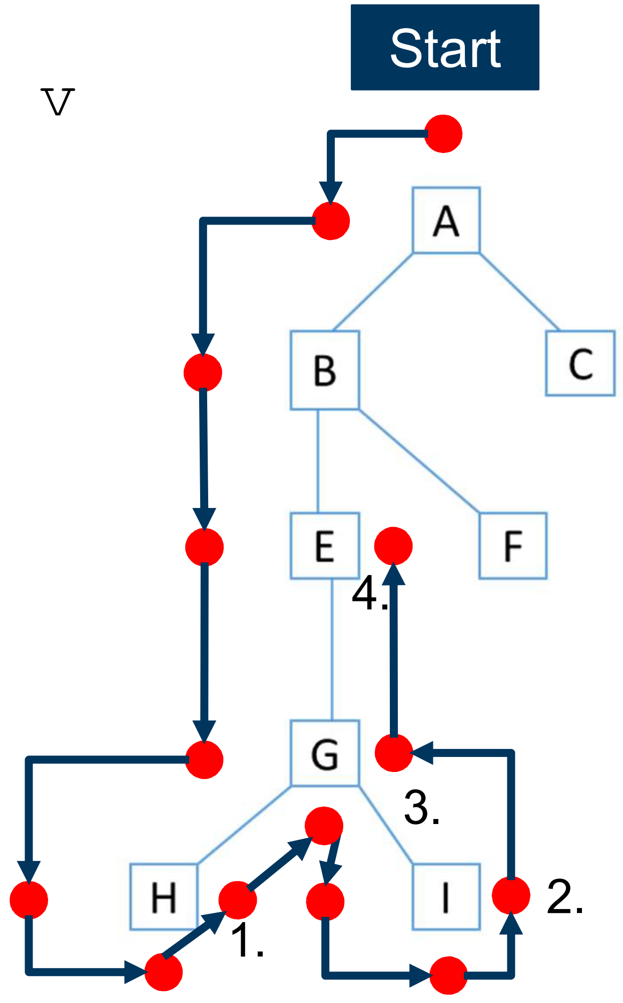


Postorder Traversal

```
postOrder(Node v)
        for each child w of v
                postOrder(w)
        visit(v)
```

Add a node to the traversal when touching its right side

Traversal: HIGE

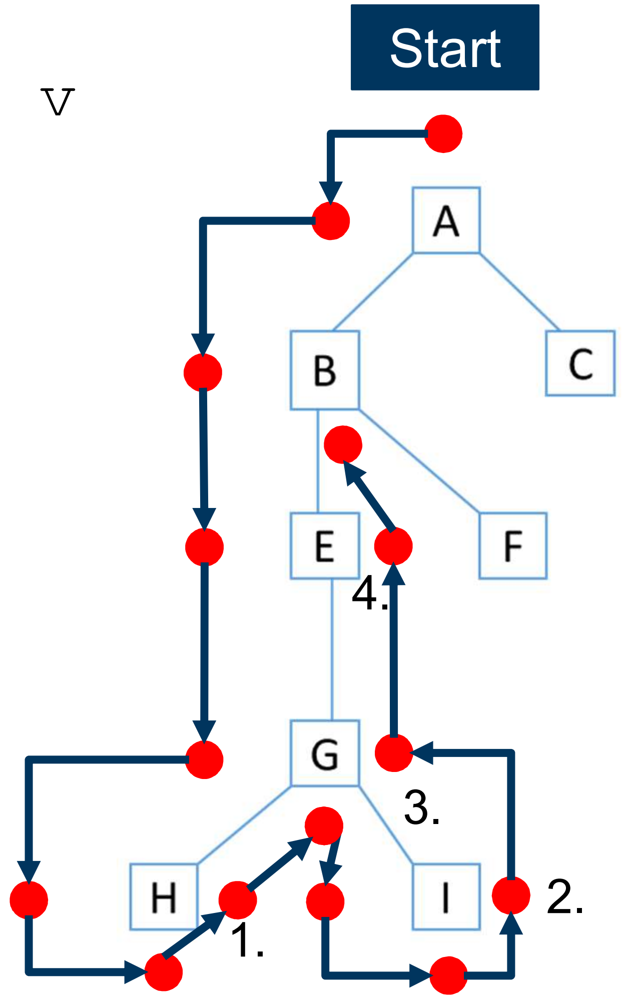


Postorder Traversal

```
postOrder(Node v)
    for each child w of v
        postOrder(w)
    visit(v)
```

Add a node to the traversal when touching its right side

Traversal: HIGE

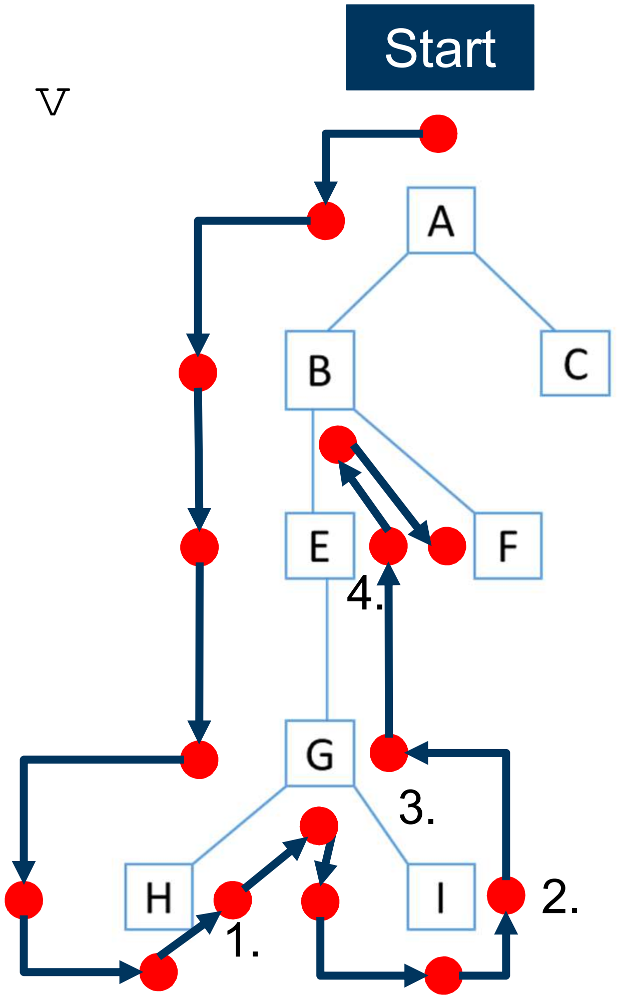


Postorder Traversal

```
postOrder(Node v)
    for each child w of v
        postOrder(w)
    visit(v)
```

Add a node to the traversal when touching its right side

Traversal: HIGE

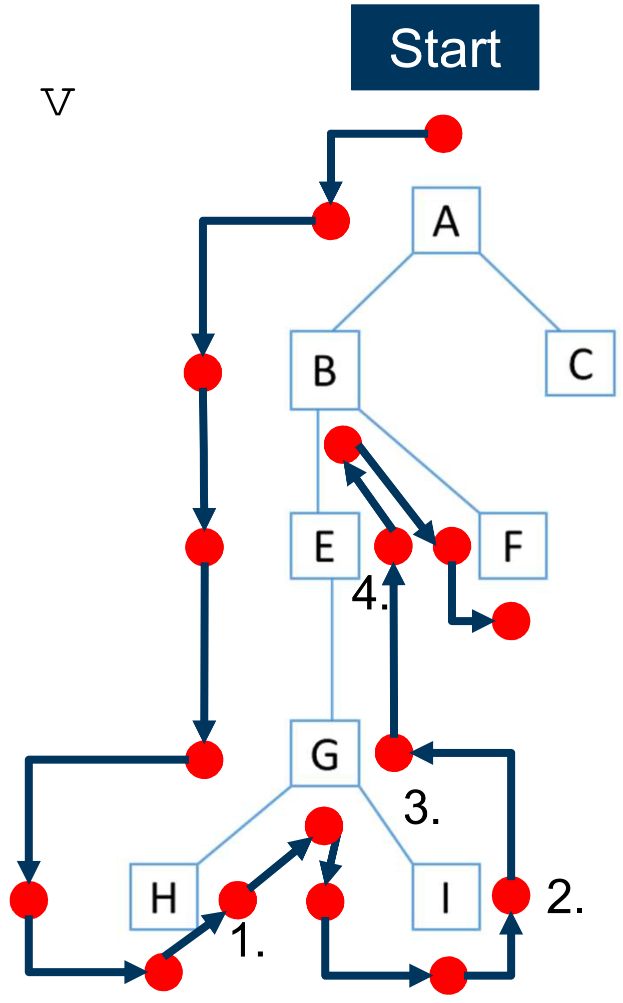


Postorder Traversal

```
postOrder(Node v)
        for each child w of v
                postOrder(w)
        visit(v)
```

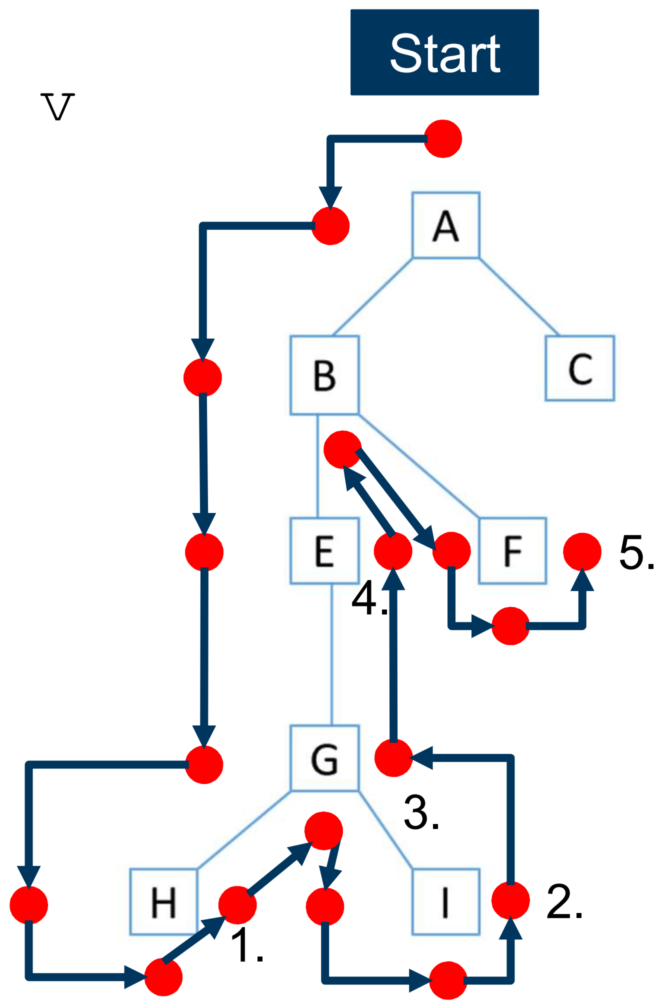
Add a node to the traversal when touching its right side

Traversal: HIGE



Postorder Traversal

```
postOrder(Node v)
        for each child w of v
                postOrder(w)
        visit(v)
```



Add a node to the traversal when touching its right side

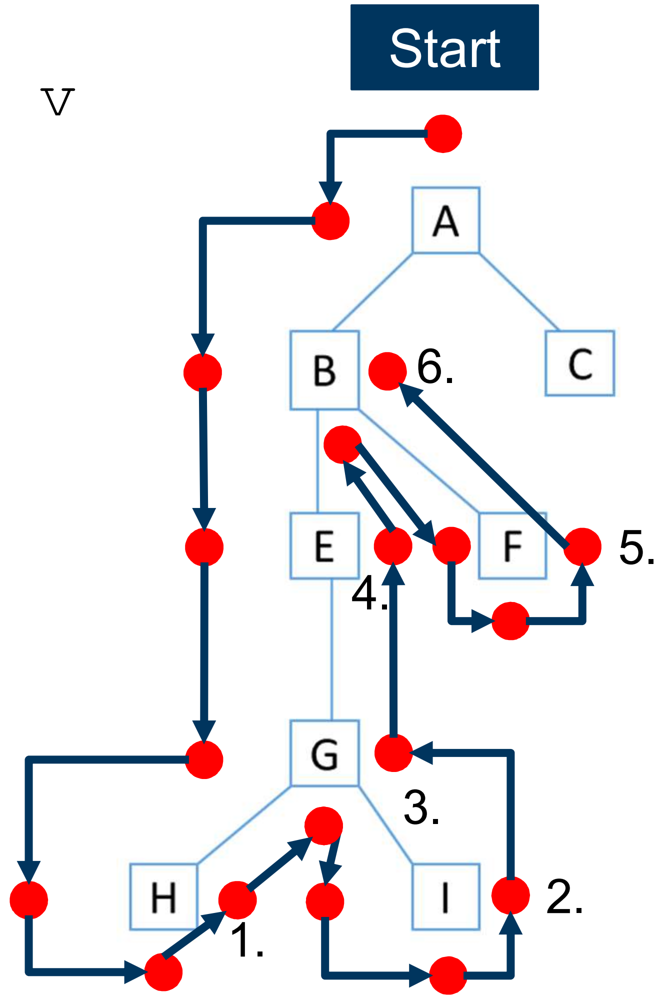
Traversal: HIGEF

Postorder Traversal

```
postOrder(Node v)
        for each child w of v
                postOrder(w)
        visit(v)
```

Add a node to the traversal when touching its right side

Traversal: HIGEFB

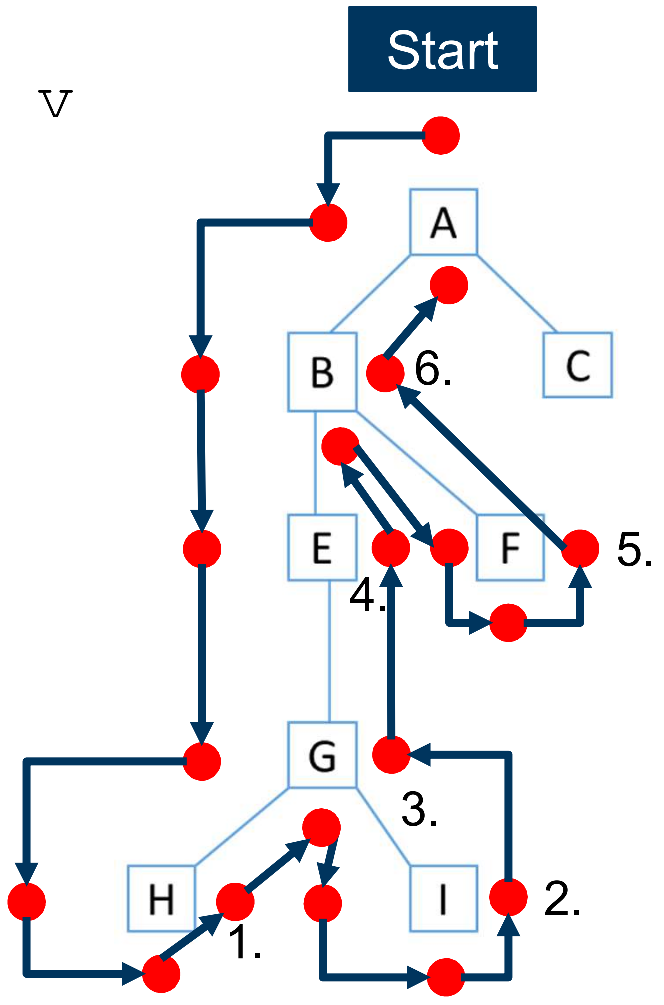


Postorder Traversal

```
postOrder(Node v)
    for each child w of v
        postOrder(w)
    visit(v)
```

Add a node to the traversal when touching its right side

Traversal: HIGEFB

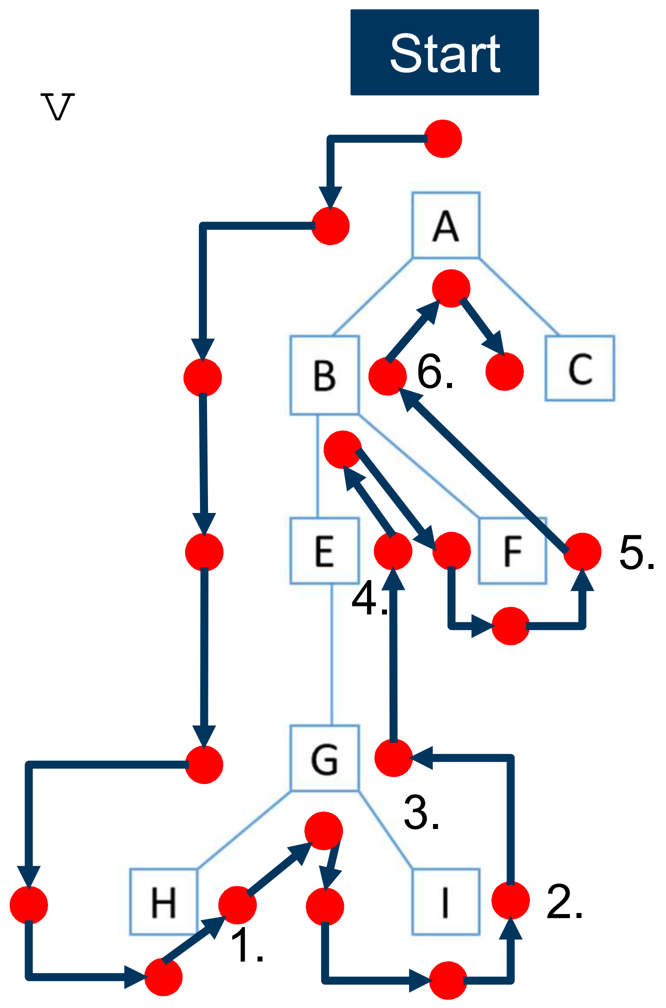


Postorder Traversal

```
postOrder(Node v)
        for each child w of v
                postOrder(w)
        visit(v)
```

Add a node to the traversal when touching its right side

Traversal: HIGEFB

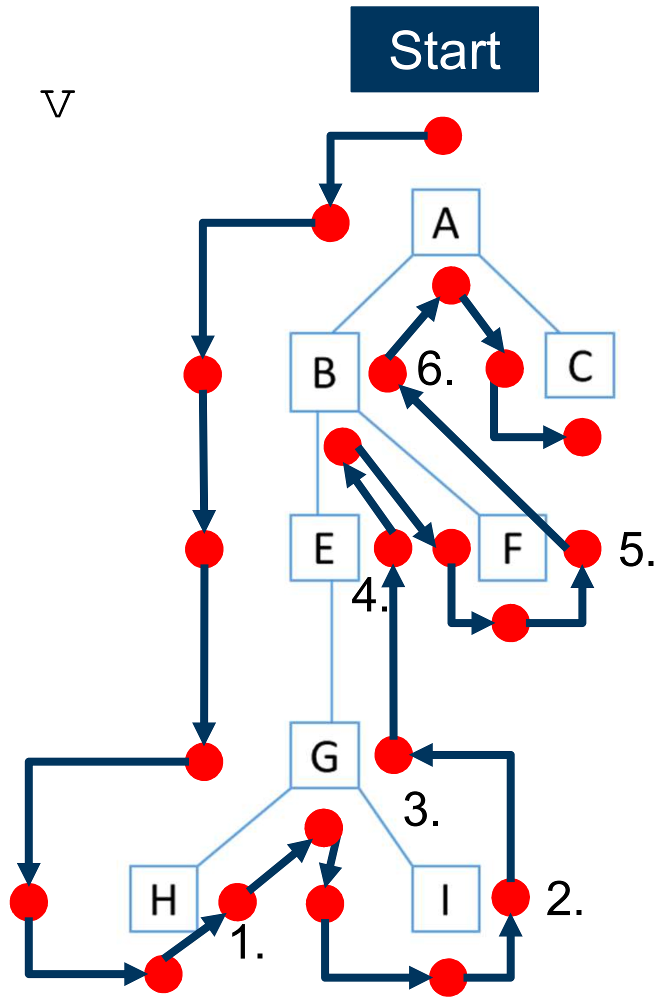


Postorder Traversal

```
postOrder(Node v)
        for each child w of v
                postOrder(w)
        visit(v)
```

Add a node to the traversal when touching its right side

Traversal: HIGEFB

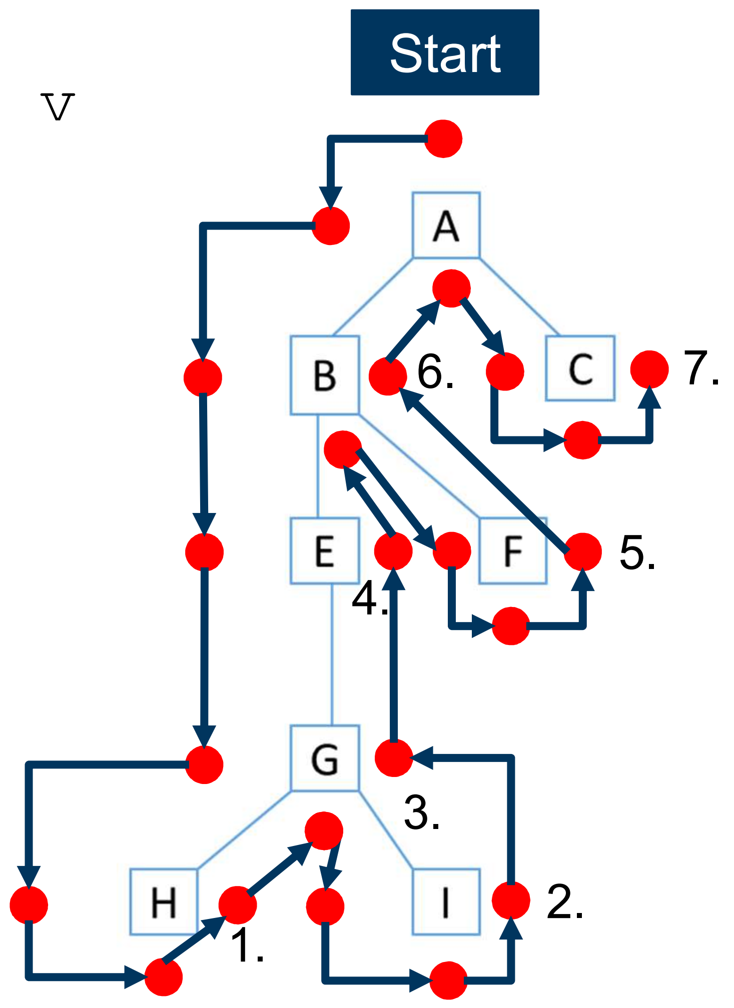


Postorder Traversal

```
postOrder(Node v)
    for each child w of v
        postOrder(w)
    visit(v)
```

Add a node to the traversal when touching its right side

Traversal: HIGEFBC

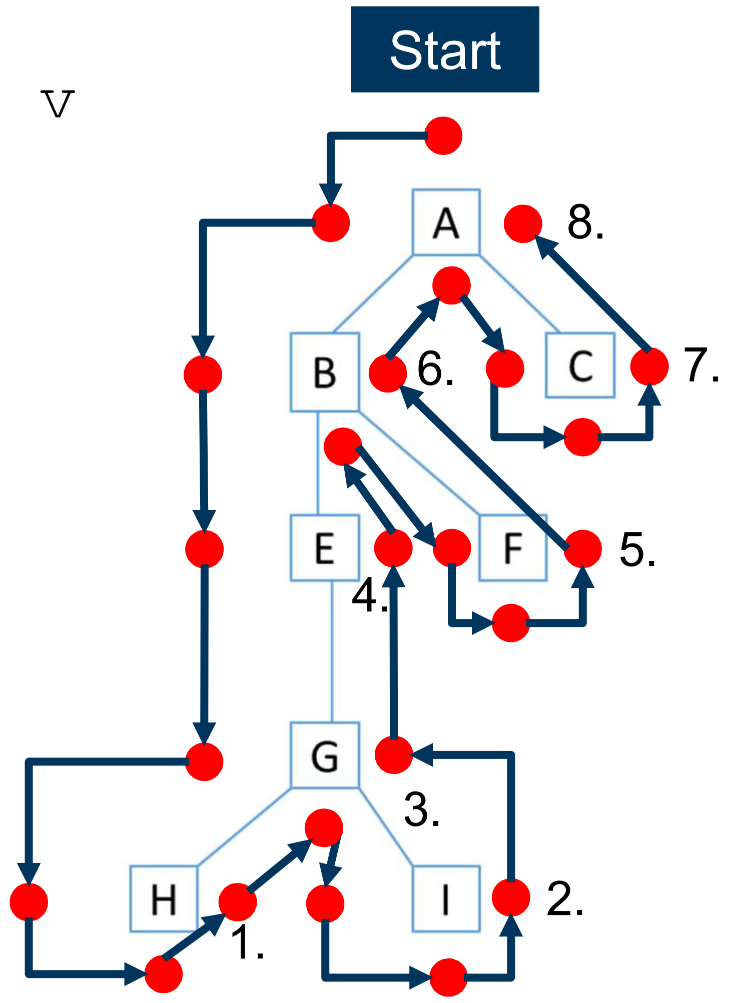


Postorder Traversal

```
postOrder(Node v)
    for each child w of v
        postOrder(w)
    visit(v)
```

Add a node to the traversal when touching its right side

Traversal: HIGEFBCA

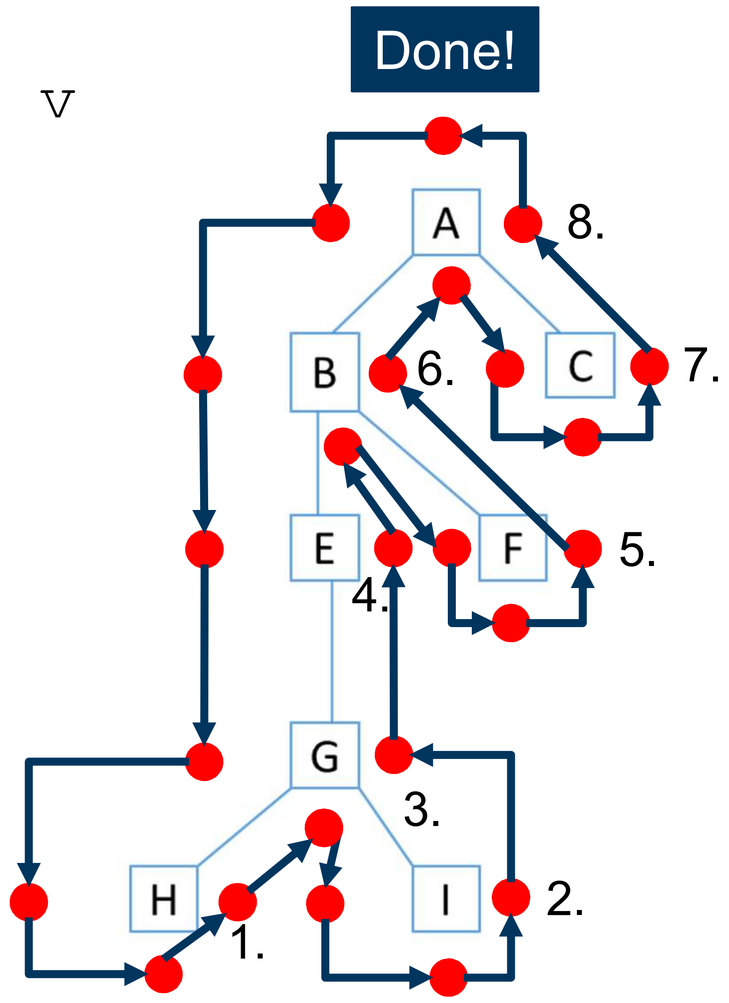


Postorder Traversal

```
postOrder(Node v)
    for each child w of v
        postOrder(w)
    visit(v)
```

Add a node to the traversal when touching its right side

Traversal: HIGEFBCA

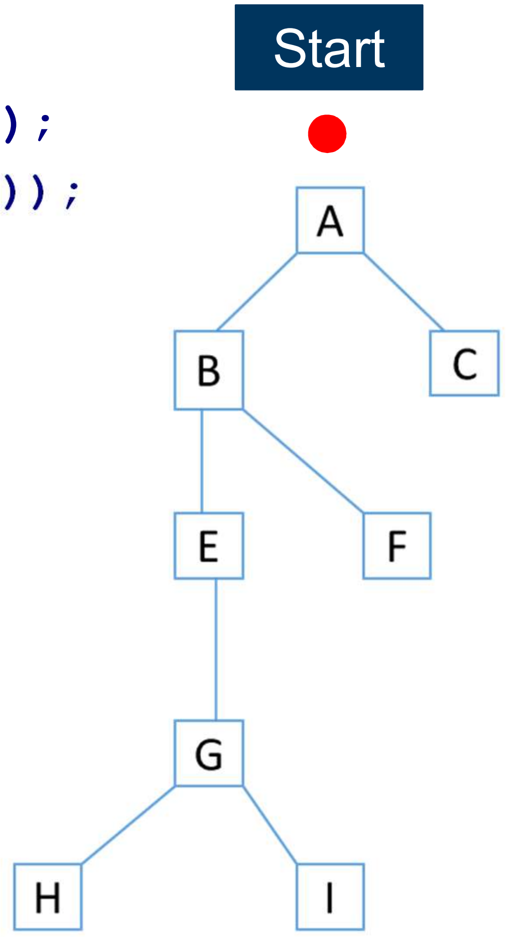


Inorder Traversal

```
InOrder(Node root)
      if (root != null)
            inOrder(root.leftChild());
            inOrder(root.rightChild());
```

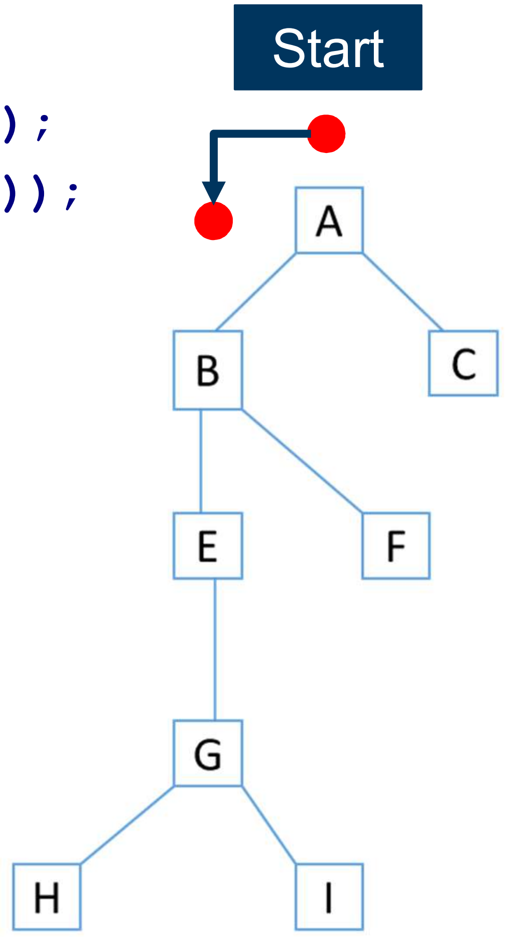
Add a node to the traversal when touching its bottom side

Traversal:



Inorder Traversal

```
InOrder(Node root)
      if (root != null)
            inOrder(root.leftChild());
            inOrder(root.rightChild());
```



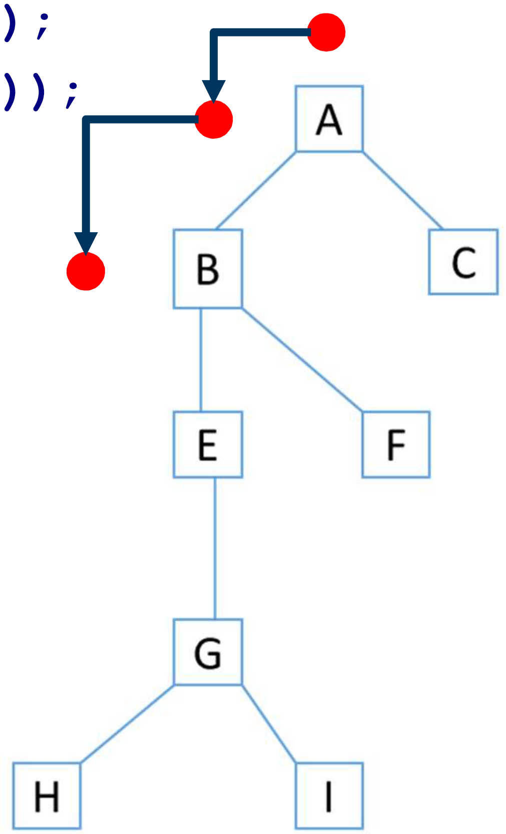
Add a node to the traversal when touching its bottom side

Traversal:

Inorder Traversal

```
InOrder(Node root)
      if (root != null)
            inOrder(root.leftChild());
            inOrder(root.rightChild());
```

Start



Add a node to the traversal when touching its bottom side

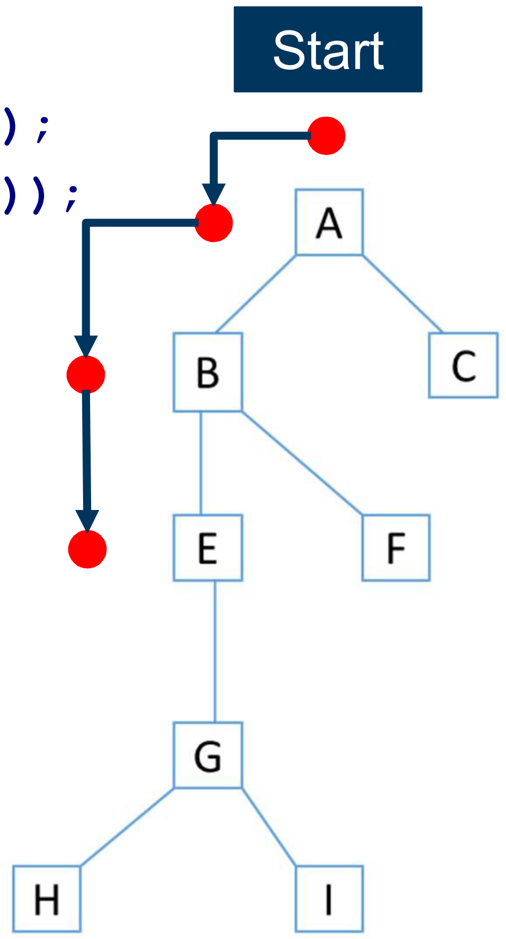
Traversal:

Inorder Traversal

```
InOrder(Node root)
      if (root != null)
            inOrder(root.leftChild());
            inOrder(root.rightChild());
```

Add a node to the traversal when touching its bottom side

Traversal:

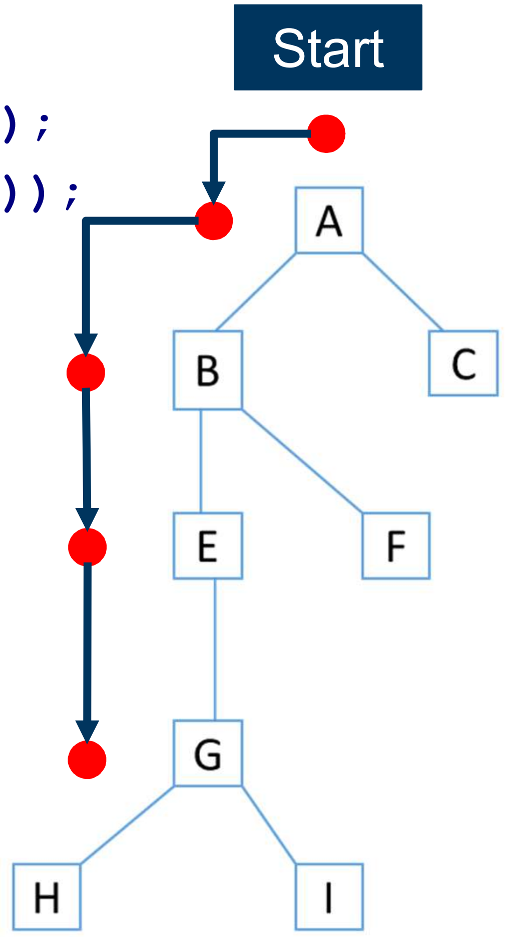


Inorder Traversal

```
InOrder(Node root)
      if (root != null)
            inOrder(root.leftChild());
            inOrder(root.rightChild());
```

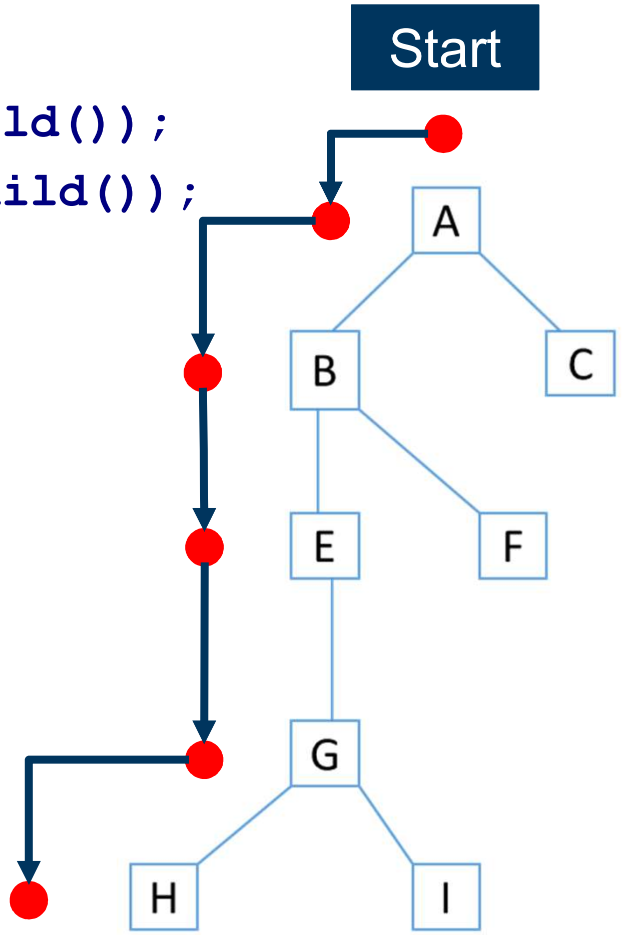
Add a node to the traversal when touching its bottom side

Traversal:



Inorder Traversal

```
InOrder(Node root)
      if (root != null)
            inOrder(root.leftChild());
            inOrder(root.rightChild());
```

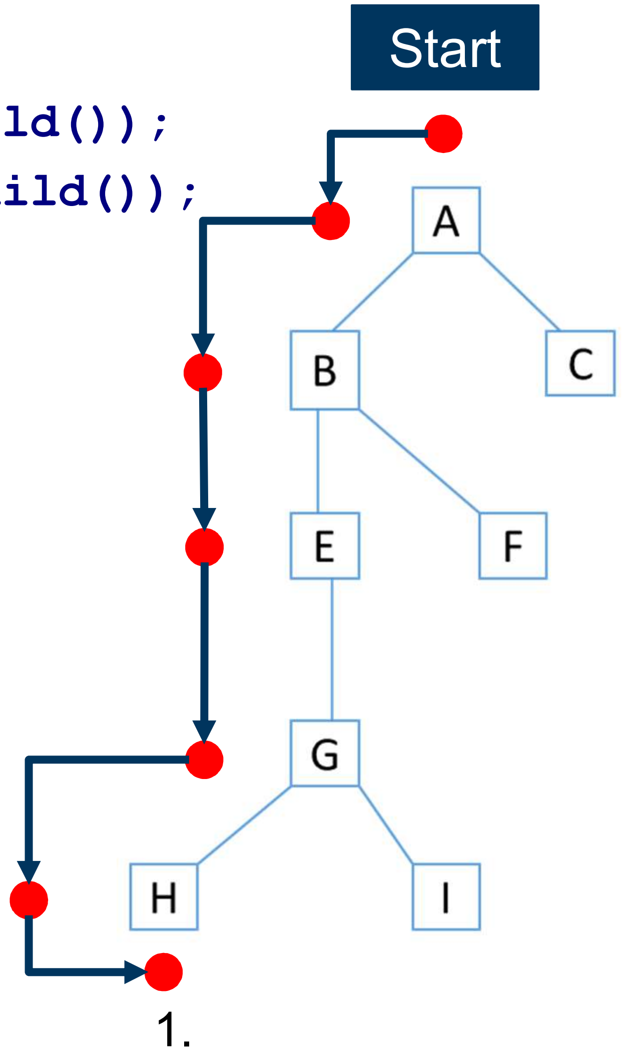


Add a node to the traversal when touching its bottom side

Traversal:

Inorder Traversal

```
InOrder(Node root)
      if (root != null)
            inOrder(root.leftChild());
            inOrder(root.rightChild());
```

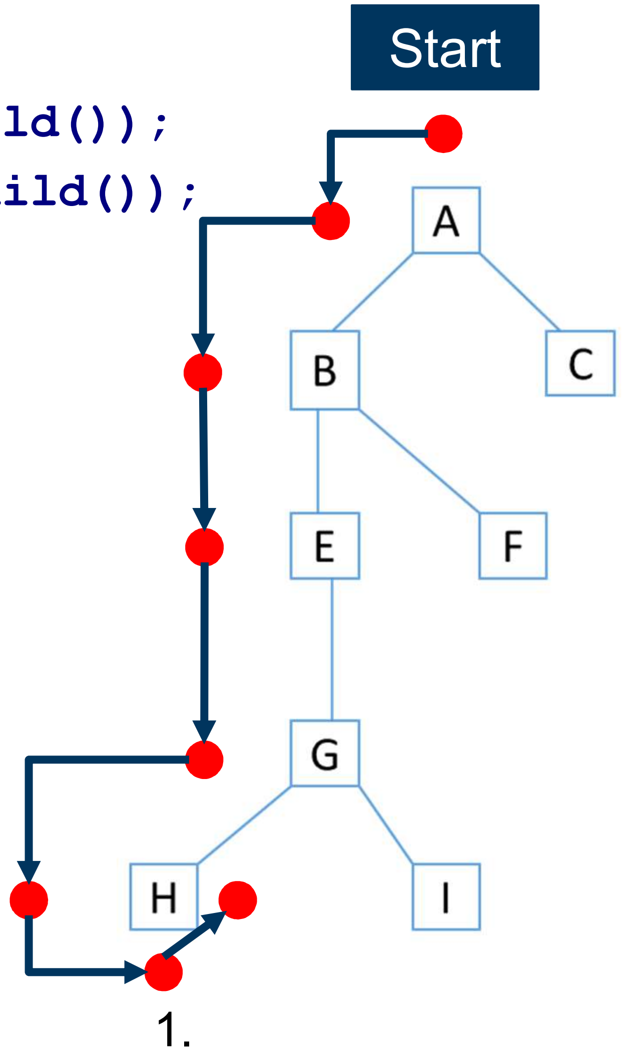


Add a node to the traversal when touching its bottom side

Traversal: H

Inorder Traversal

```
InOrder(Node root)
        if (root != null)
                inOrder(root.leftChild());
                inOrder(root.rightChild());
```



Add a node to the traversal when touching its bottom side

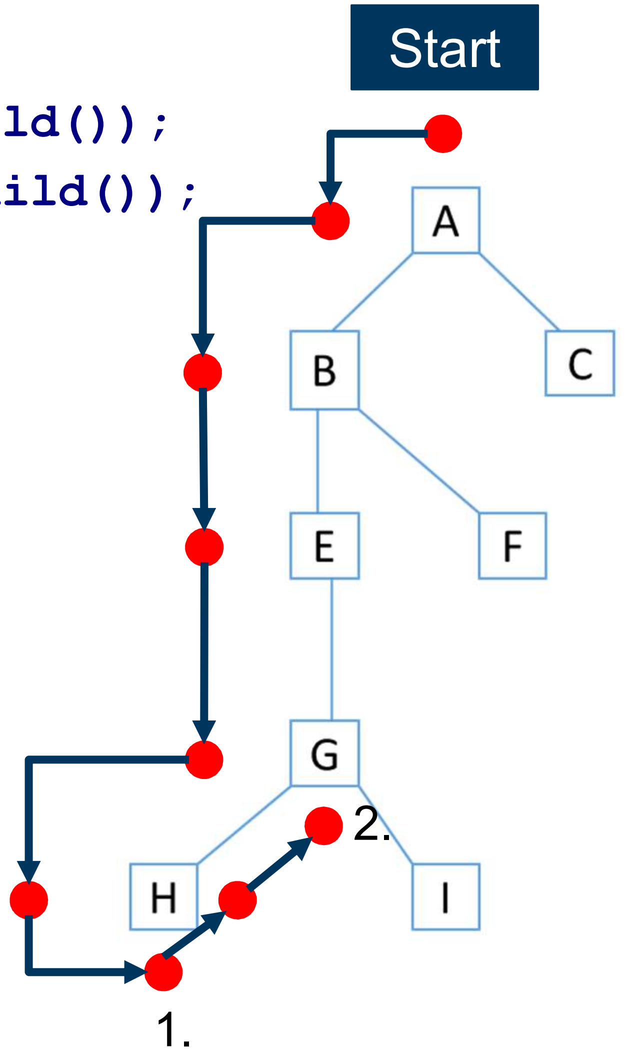
Traversal: H

Inorder Traversal

```
InOrder(Node root)
      if (root != null)
            inOrder(root.leftChild());
            inOrder(root.rightChild());
```

Add a node to the traversal when touching its bottom side

Traversal: HG

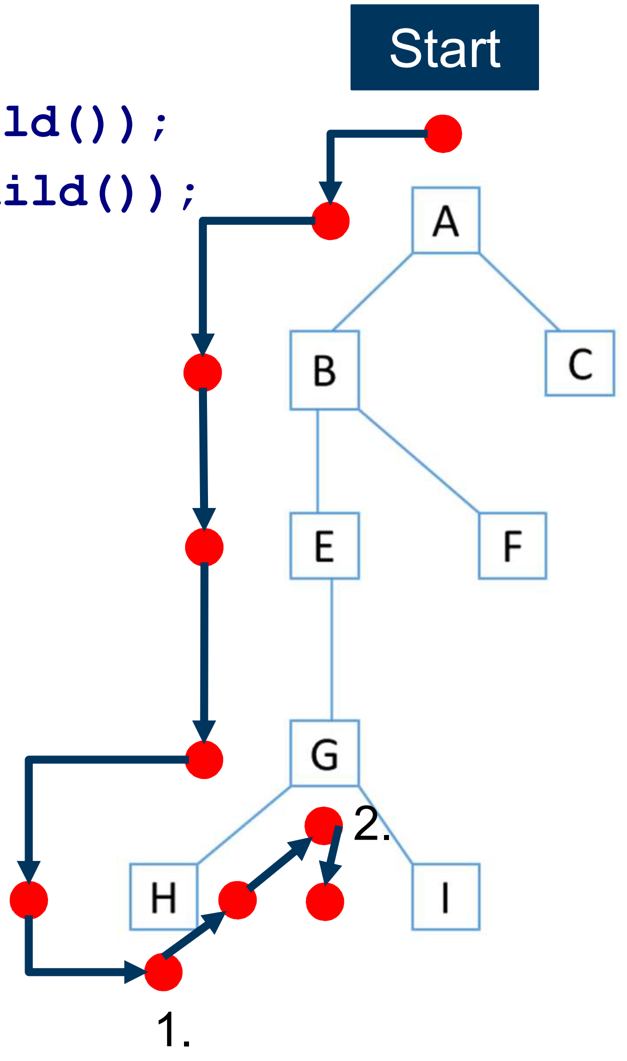


Inorder Traversal

```
InOrder(Node root)
      if (root != null)
            inOrder(root.leftChild());
            inOrder(root.rightChild());
```

Add a node to the traversal when touching its bottom side

Traversal: HG

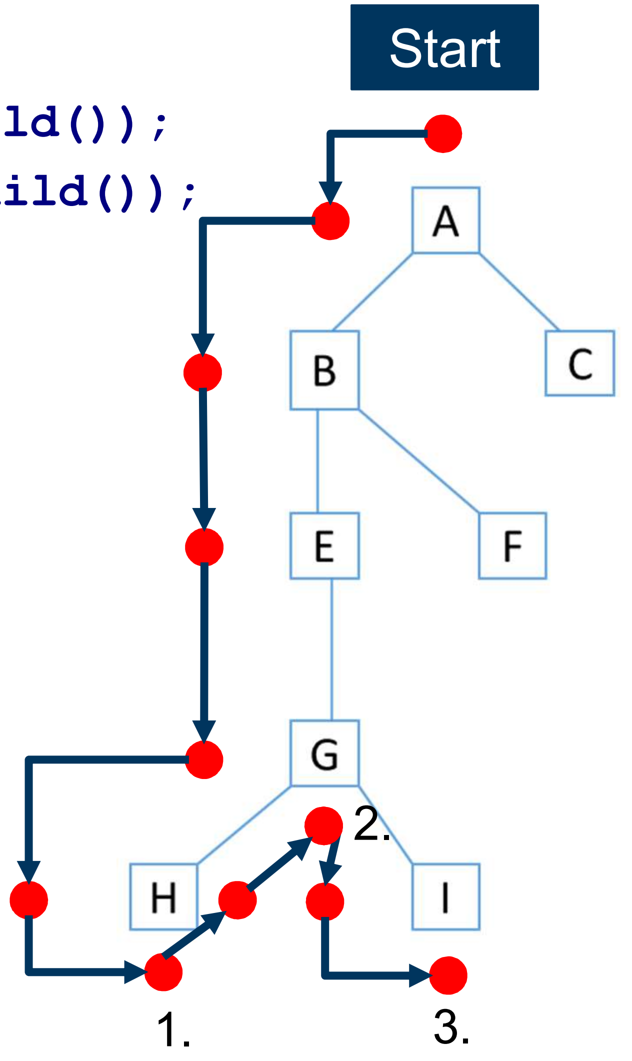


Inorder Traversal

```
InOrder(Node root)
      if (root != null)
            inOrder(root.leftChild());
            inOrder(root.rightChild());
```

Add a node to the traversal when touching its bottom side

Traversal: HGI

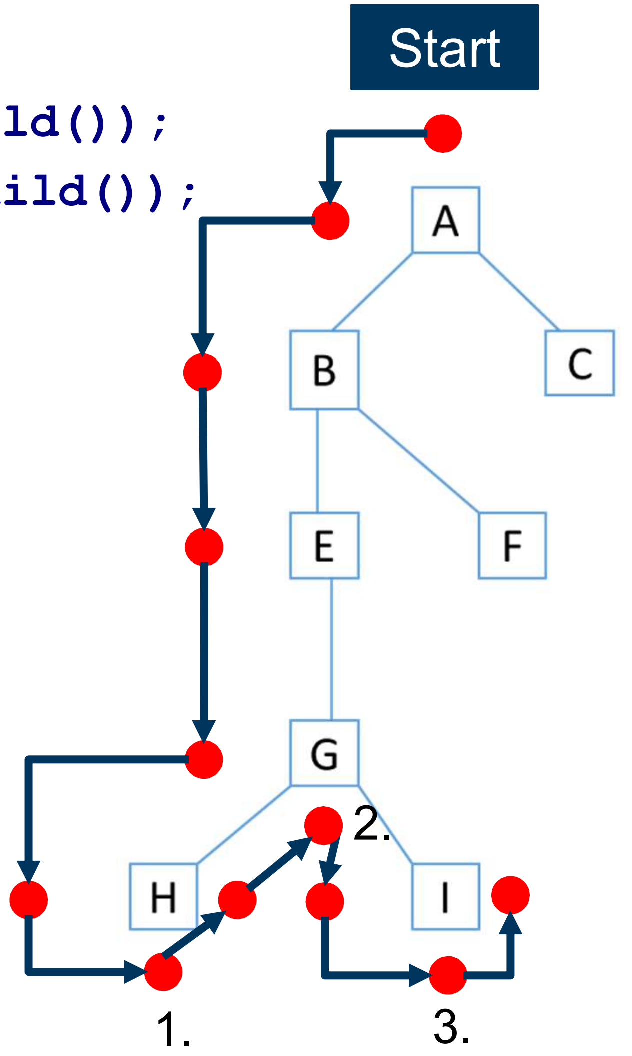


Inorder Traversal

```
InOrder(Node root)
      if (root != null)
            inOrder(root.leftChild());
            inOrder(root.rightChild());
```

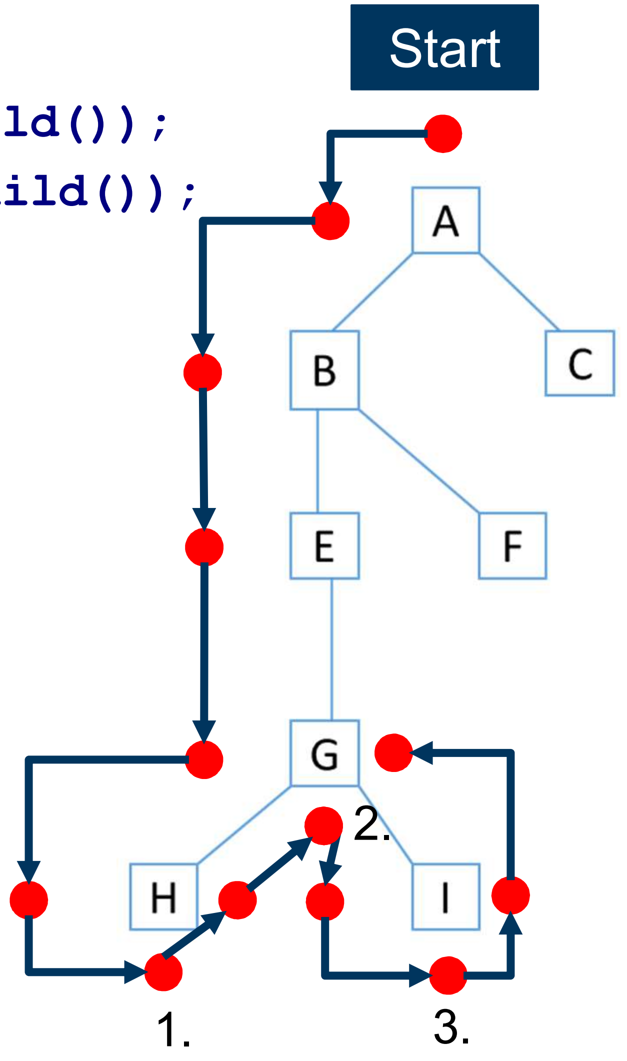
Add a node to the traversal when touching its bottom side

Traversal: HGI



Inorder Traversal

```
InOrder(Node root)
      if (root != null)
            inOrder(root.leftChild());
            inOrder(root.rightChild());
```



Add a node to the traversal when touching its bottom side

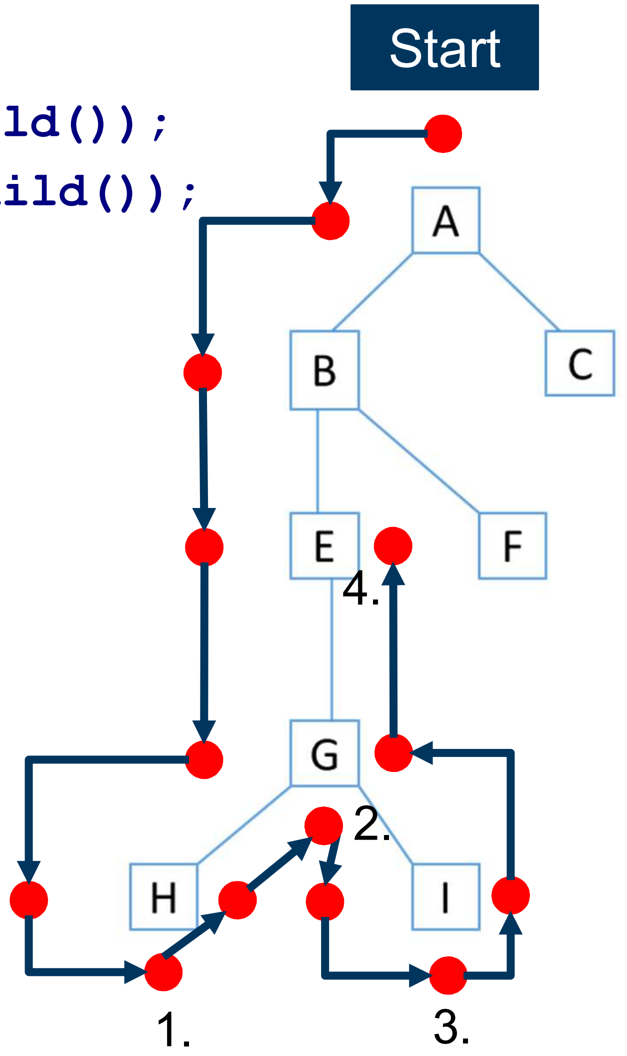
Traversal: HGI

Inorder Traversal

```
InOrder(Node root)
      if (root != null)
            inOrder(root.leftChild());
            inOrder(root.rightChild());
```

Add a node to the traversal when touching its bottom side

Traversal: HGIE

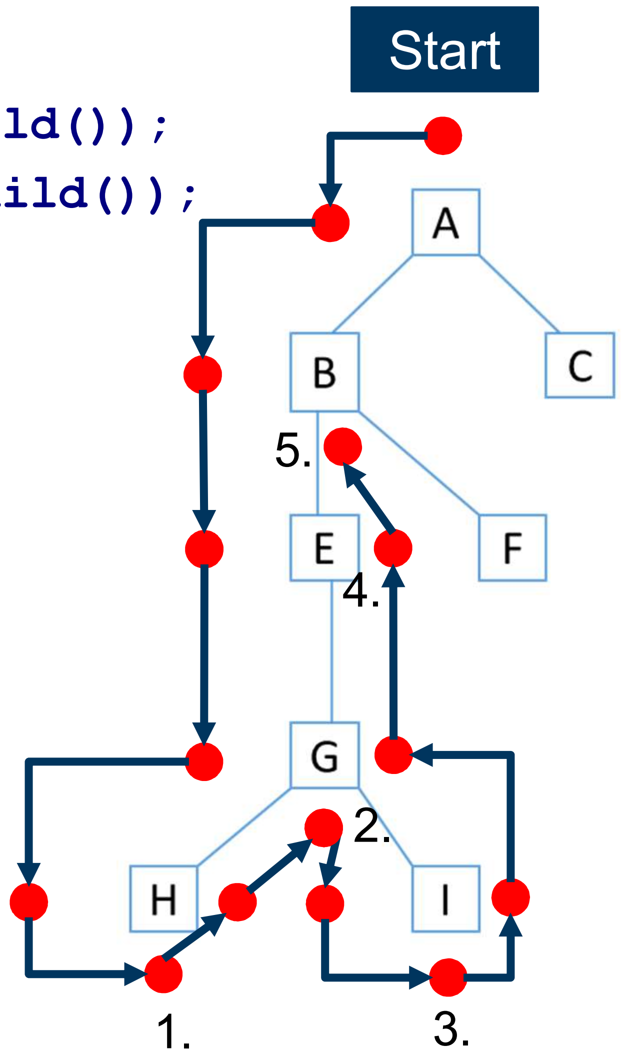


Inorder Traversal

```
InOrder(Node root)
        if (root != null)
                inOrder(root.leftChild());
                inOrder(root.rightChild());
```

Add a node to the traversal when touching its bottom side

Traversal: HGIEB

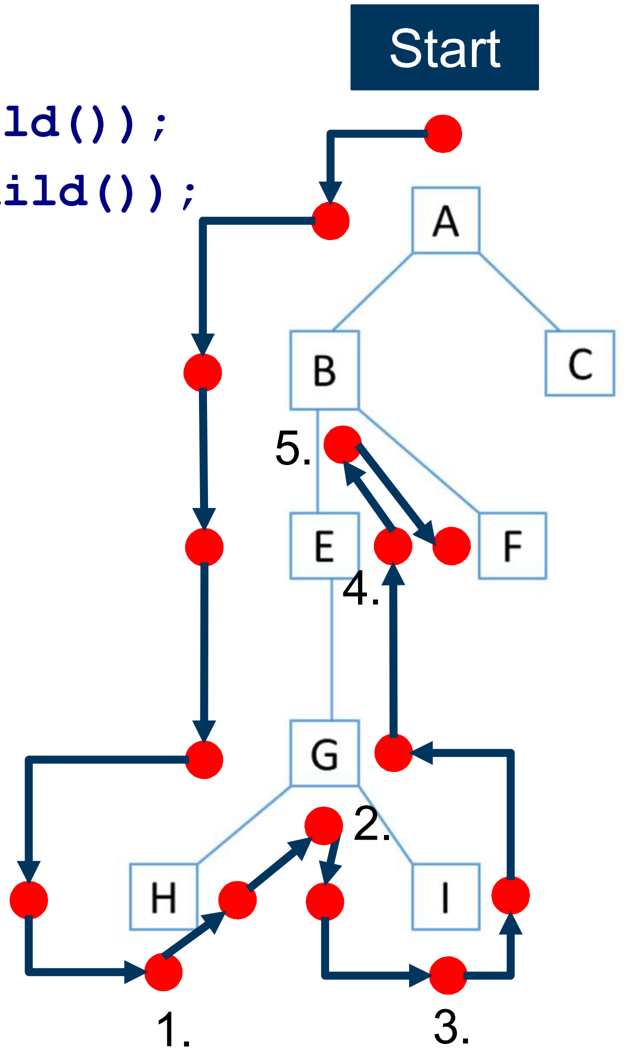


Inorder Traversal

```
InOrder(Node root)
      if (root != null)
            inOrder(root.leftChild());
            inOrder(root.rightChild());
```

Add a node to the traversal when touching its bottom side

Traversal: HGIEB

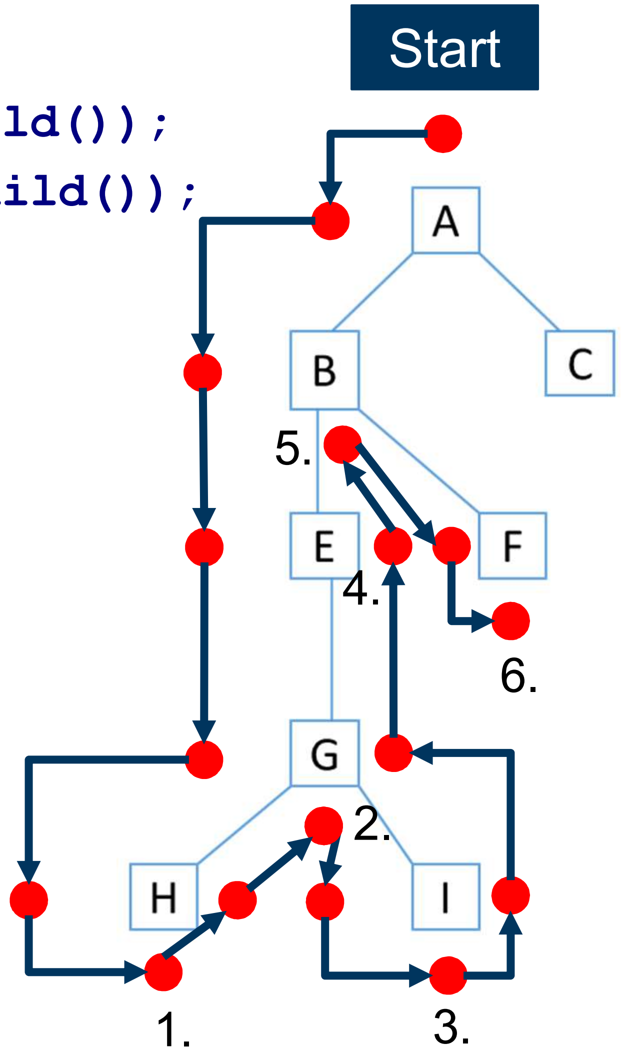


Inorder Traversal

```
InOrder(Node root)
      if (root != null)
            inOrder(root.leftChild());
            inOrder(root.rightChild());
```

Add a node to the traversal when touching its bottom side

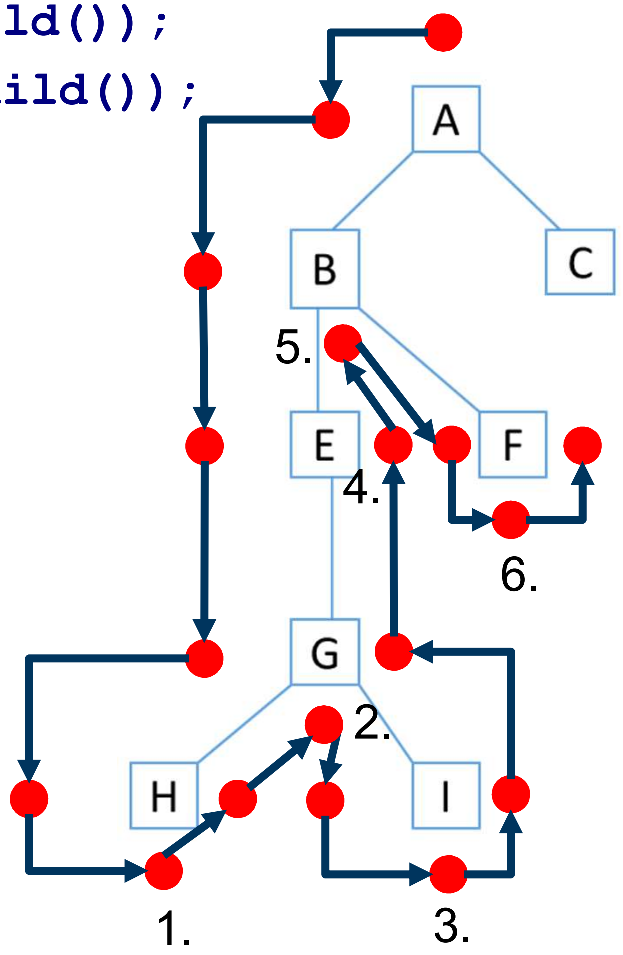
Traversal: HGIEBF



Inorder Traversal

```
InOrder(Node root)
      if (root != null)
            inOrder(root.leftChild());
            inOrder(root.rightChild());
```

Start



Add a node to the traversal when touching its bottom side

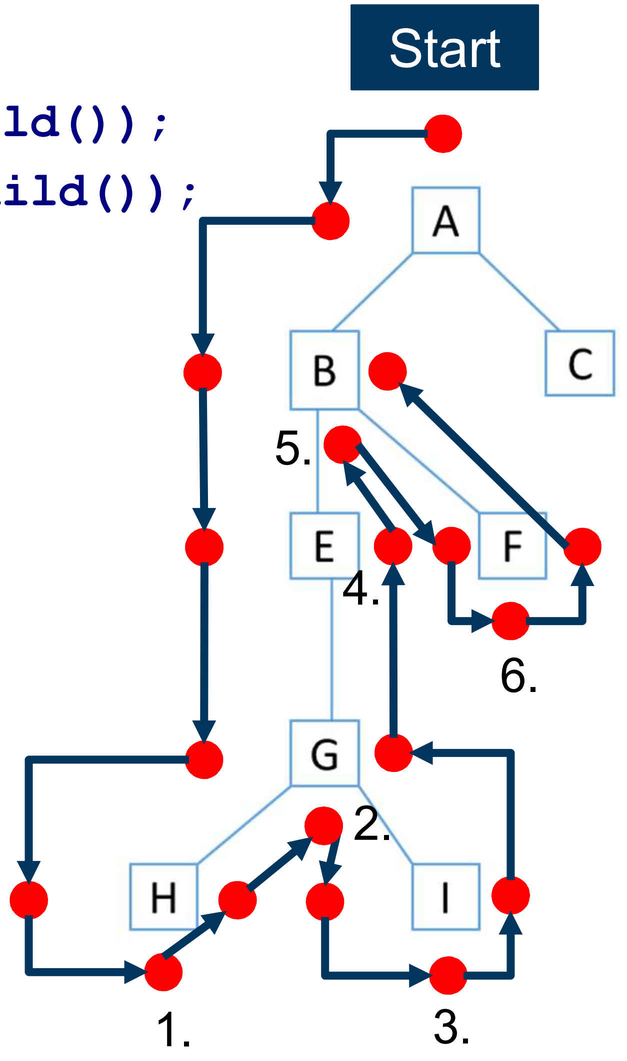
Traversal: HGIEBF

Inorder Traversal

```
InOrder(Node root)
      if (root != null)
            inOrder(root.leftChild());
            inOrder(root.rightChild());
```

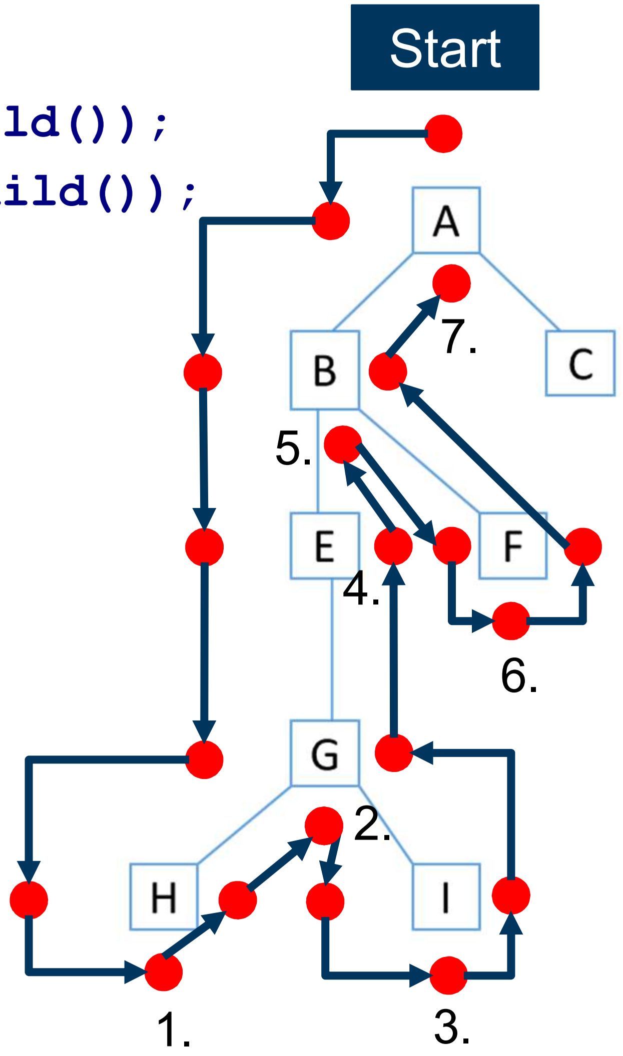
Add a node to the traversal when touching its bottom side

Traversal: HGIEBF



Inorder Traversal

```
InOrder(Node root)
      if (root != null)
            inOrder(root.leftChild());
            inOrder(root.rightChild());
```



Add a node to the traversal when touching its bottom side

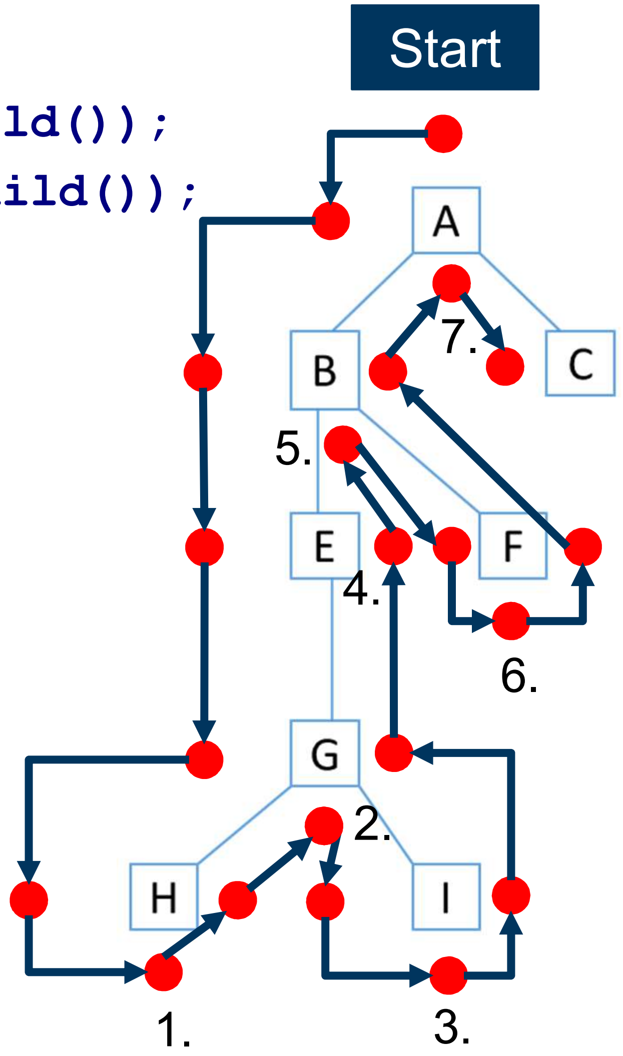
Traversal: HGIEBFA

Inorder Traversal

```
InOrder(Node root)
      if (root != null)
            inOrder(root.leftChild());
            inOrder(root.rightChild());
```

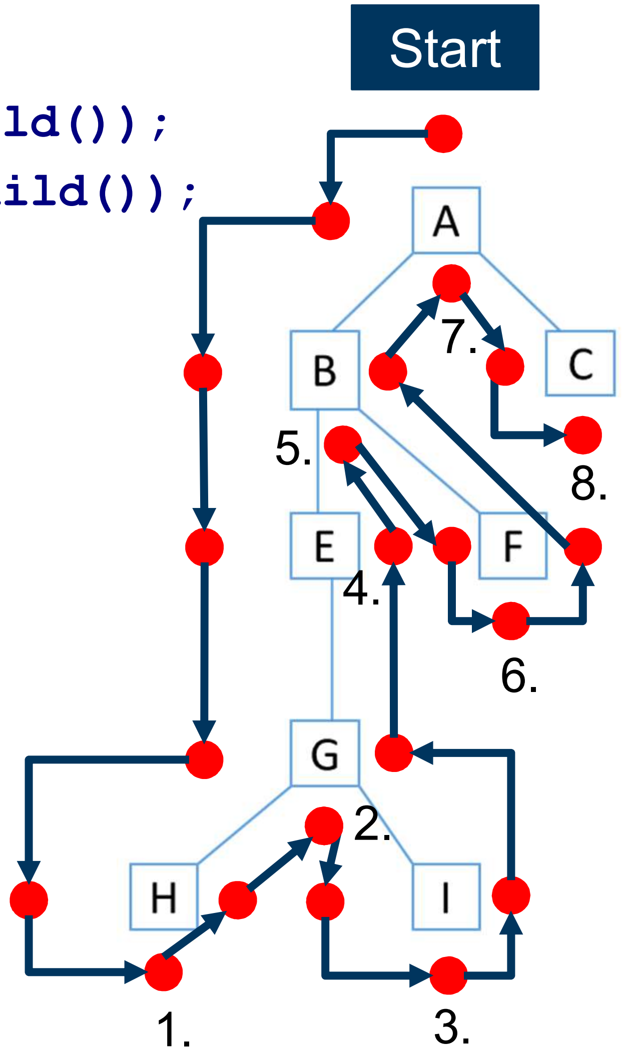
Add a node to the traversal when touching its bottom side

Traversal: HGIEBFA



Inorder Traversal

```
InOrder(Node root)
        if (root != null)
                inOrder(root.leftChild());
                inOrder(root.rightChild());
```



Add a node to the traversal when touching its bottom side

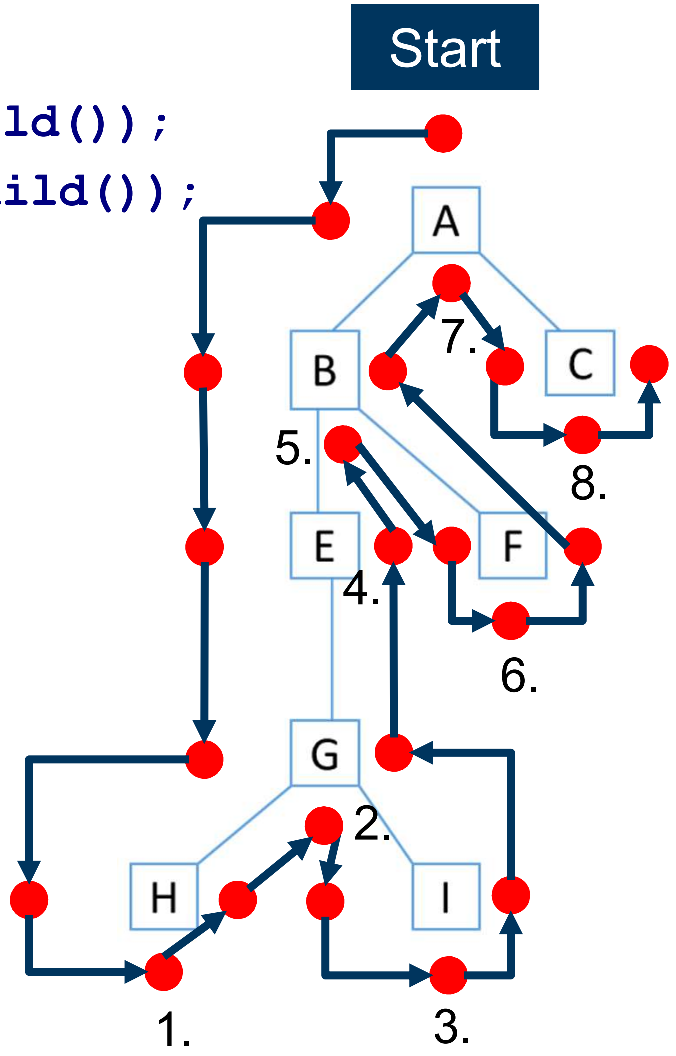
Traversal: HGIEBFAC

Inorder Traversal

```
InOrder(Node root)
      if (root != null)
            inOrder(root.leftChild());
            inOrder(root.rightChild());
```

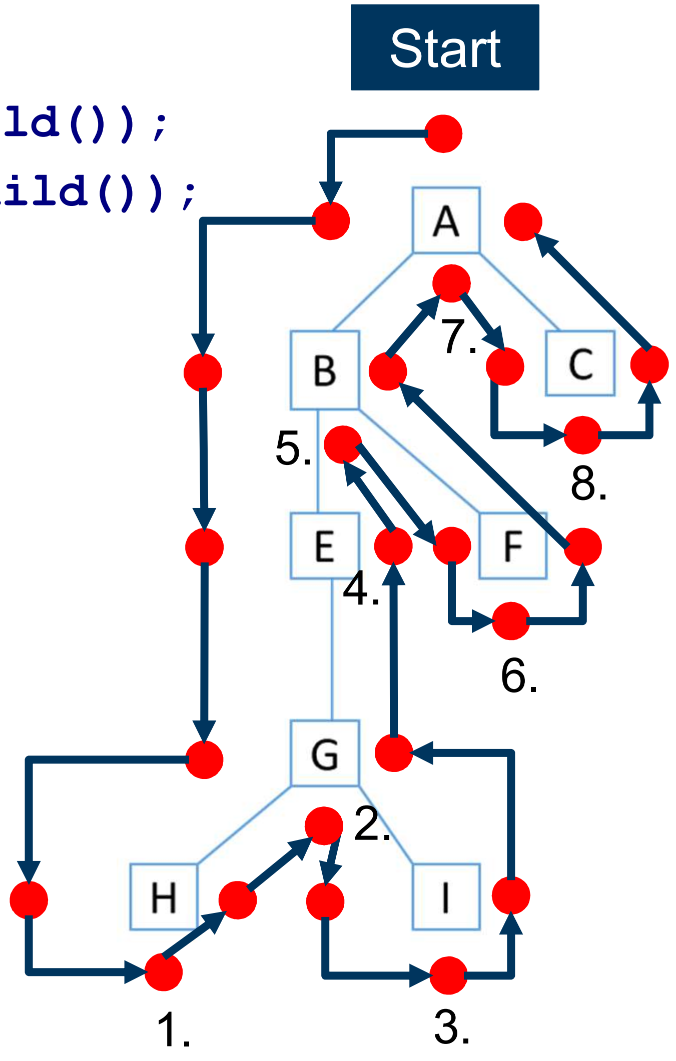
Add a node to the traversal when touching its bottom side

Traversal: HGIEBFAC



Inorder Traversal

```
InOrder(Node root)
        if (root != null)
                inOrder(root.leftChild());
                inOrder(root.rightChild());
```



Add a node to the traversal when touching its bottom side

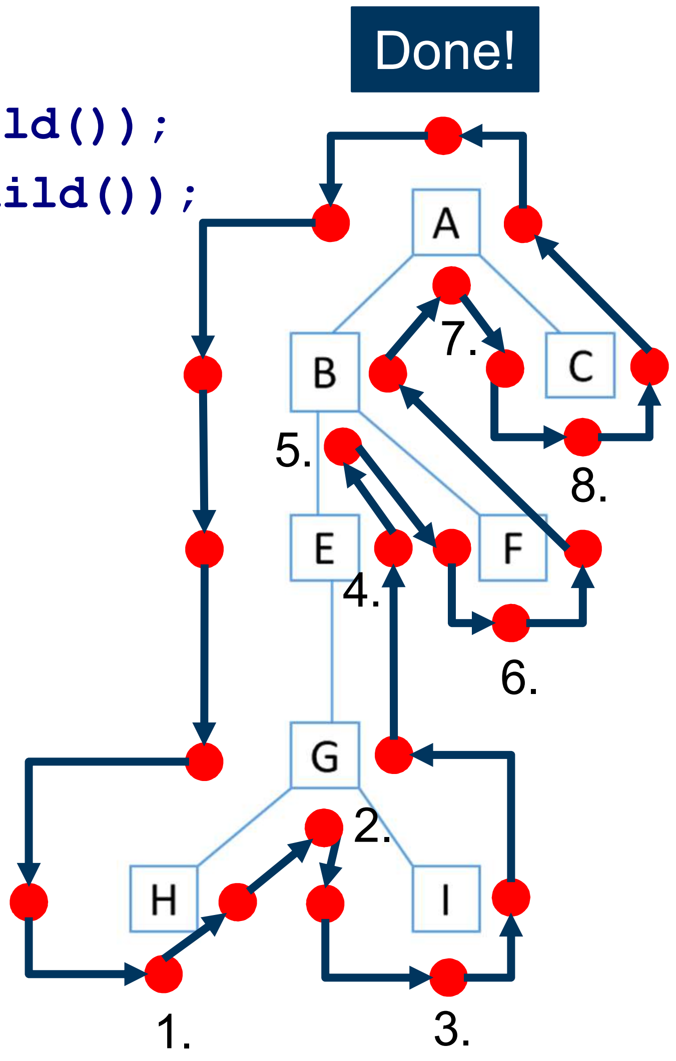
Traversal: HGIEBFAC

Inorder Traversal

```
InOrder(Node root)
        if (root != null)
                inOrder(root.leftChild());
                inOrder(root.rightChild());
```

Add a node to the traversal when touching its bottom side

Traversal: HGIEBFAC



Your Tasks

Build a binary tree from a given preorder and inorder sequence

From both of these strings we can determine how a tree is built

- We know the first node in a preorder sequence is always the root
- Knowing that, we can find where it is in the inorder sequence
  - Everything to the right of the root is in the child’s subtree
  - Everything to the left of the root is in the left child’s subtree
- Once we determine what the subtrees are, we can recursively call the method on the subtrees of the root’s child and build our tree from that

Algorithm

Root: First character in the sequence

Get Index of the root node (root_index) in the inorder sequence

Left inorder = index 0 to root_index

Right inorder = root_index to end of sequence

Left preorder = index 1 to root_index

Right preorder = root_index to end of sequence

Recursively call the method on (left inorder, left preorder) and (right inorder, right preorder) to build all subtrees

Base Case: sequence is of size 1

Example

String preorder = “BZRFTUHOL”;

String inorder = “RZTFUBOHL”;

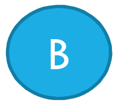
Root is B

Left inorder: “RZTFU”

Right inorder: “OHL”

Left preorder: “ZRFTU”

Right Preorder: “HOL”



Example – Left Recursive Call

String preorder = “ZRFTU”;

String inorder = “RZTFU”;

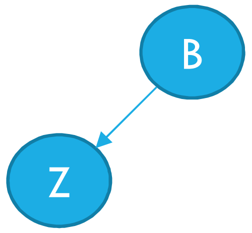
Root is Z

Left inorder: “R”

Right inorder: “TFU”

Left preorder: “R”

Right Preorder: “FTU”



Example – Left Recursive Call

String preorder = “R”;

String inorder = “R”;

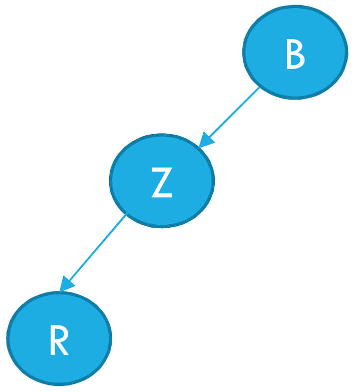
Root is R

Left inorder: “NULL”

Right inorder: “NULL”

Left preorder: “NULL”

Right Preorder: “NULL”



Example – Right Recursive Call

String preorder = “FTU”;

String inorder = “TFU”;

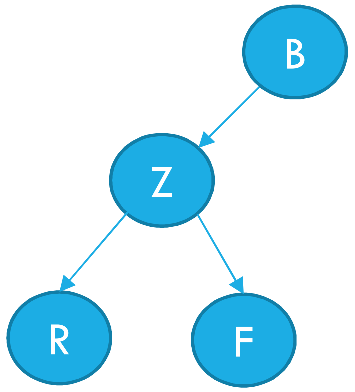
Root is F

Left inorder: “T”

Right inorder: “U”

Left preorder: “T”

Right Preorder: “U”



Example – Left Recursive Call

String preorder = “T”;

String inorder = “T”;

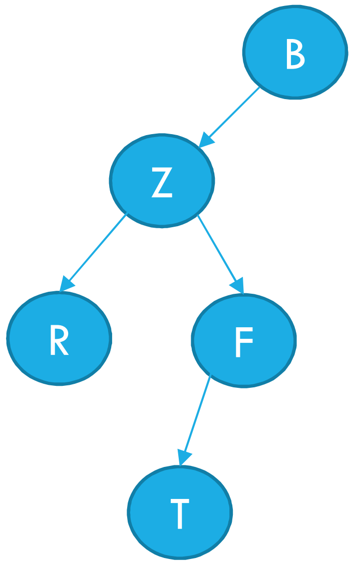
Root is T

Left inorder: “NULL”

Right inorder: “NULL”

Left preorder: “NULL”

Right Preorder: “NULL”



Example – Right Recursive Call

String preorder = “U”;

String inorder = “U”;

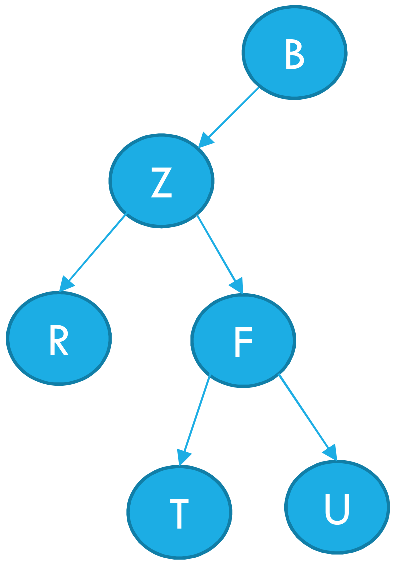
Root is U

Left inorder: “NULL”

Right inorder: “NULL”

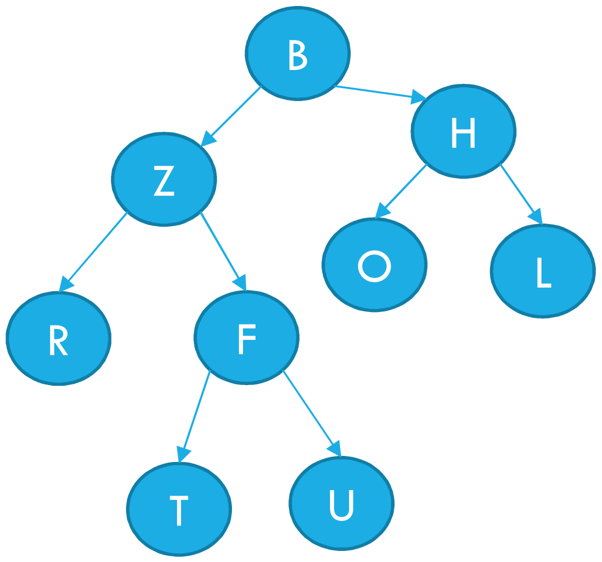
Left preorder: “NULL”

Right Preorder: “NULL”



Eventual Result

Keep following the algorithm and you will get the following tree



Your Tasks

- Download the Lab 10 instructions and Provided Code from the course website
  - http://db.cs.pitt.edu/courses/cs0445/current.term/
- Your task is to complete the rebuildTree method in RebuildBinaryTree.java.
  - You are provided with BinaryTree.java as well as a stack and queue package for tree iterators
  - You should review BinaryTree.java to see how to create a binary tree
- Test your work!