Binary Tree Traversals

- Binary Tree Traversal classification
- BreadthFirst traversal
- DepthFirst traversal
- Accept method of BinaryTree class
- Binary Tree Iterator

Tree Traversal (Definition)

- The process of systematically visiting all the nodes in a tree and performing some computation at each node in the tree is called a tree traversal.

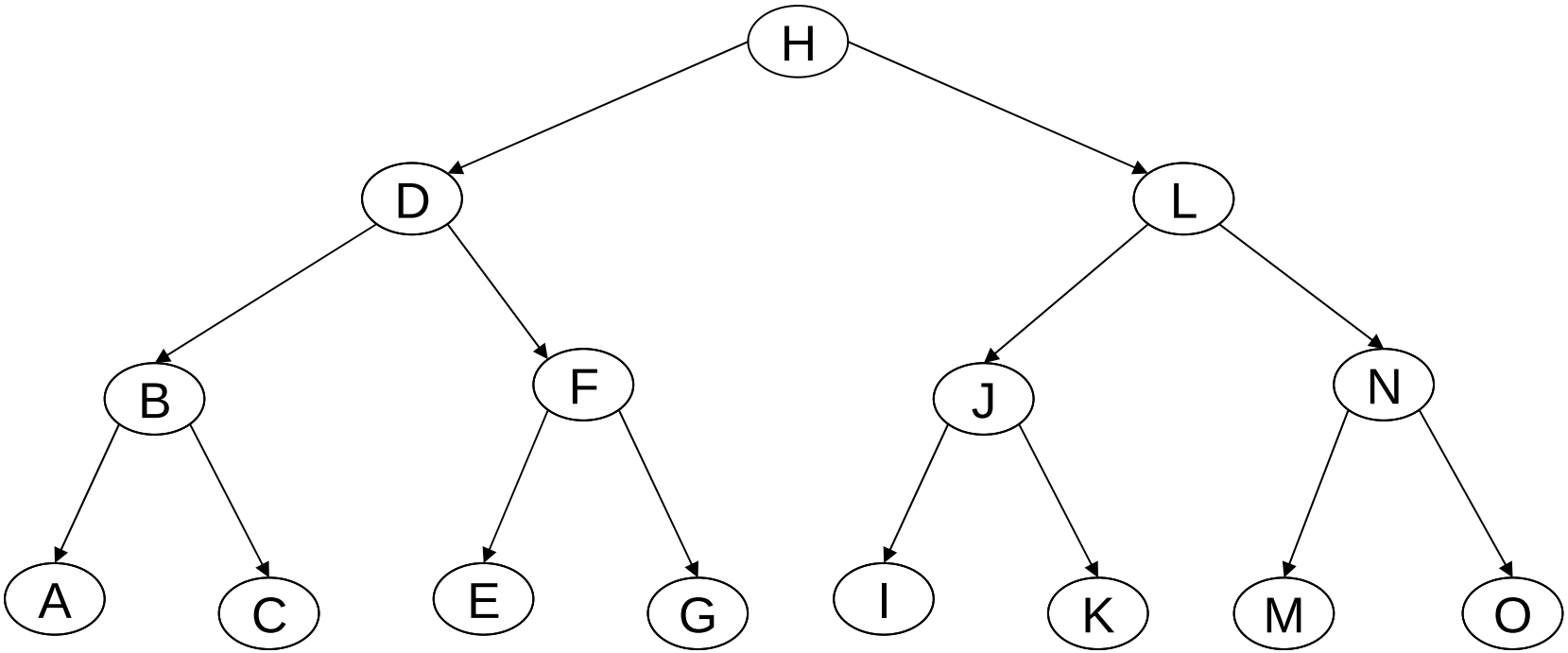
- There are two methods in which to traverse a tree:
  1. Breadth-First Traversal.
  2. Depth-First Traversal:
     - Preorder traversal
     - Inorder traversal (for binary trees only)
     - Postorder traversal

Breadth-First Traversal

- The BinaryTree class breadthFirstTraversal method:

```
public void breadthFirstTraversal(Visitor visitor){
   QueueAsLinkedList queueaslinkedlist =
                                  new QueueAsLinkedList();
   if(!isEmpty()) queueaslinkedlist.enqueue(this);
   while(!queueaslinkedlist.isEmpty() &&
                                       !visitor.isDone()){
      BinaryTree tree =
               (BinaryTree)queueaslinkedlist.dequeue();
      visitor.visit(tree.getKey());
      if (!tree.getLeft().isEmpty())
         queueaslinkedlist.enqueue(tree.getLeft());
      if (!tree.getRight().isEmpty())
         queueaslinkedlist.enqueue(tree.getRight());
   }
}
```

Breadth-First Traversal



H	D	L	B	F	J	N	A	C	E	G	I	K	M	O

Depth-First Traversals

Name	for each Node:	CODE

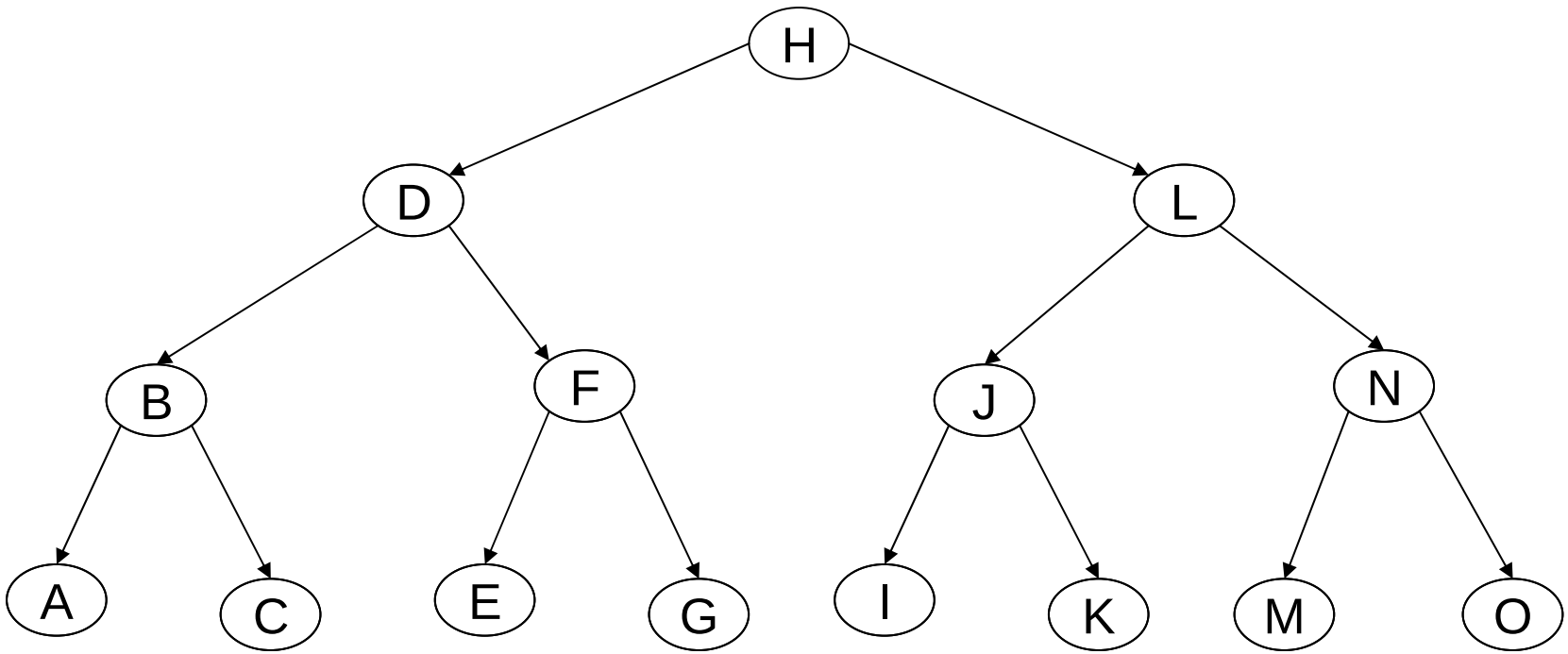
Preorder (N-L-R)	•Visit the node
	•Visit the left subtree, if any.
	•Visit the right subtree, if any.
		`public void preorderTraversal(Visitor v){`
		`  if(!isEmpty() && ! v.isDone()){`
		`  v.visit(getKey());`
		`  getLeft().preorderTraversal(v);`
		`  getRight().preorderTraversal(v);`
		`  }`
		`}`

Inorder (L-N-R)	•Visit the left subtree, if any.
	Visit the node
	•Visit the right subtree, if any.
		`public void inorderTraversal(Visitor v){`
		`  if(!isEmpty() && ! v.isDone()){`
		`  getLeft().inorderTraversal(v);`
		`  v.visit(getKey());`
		`  getRight().inorderTraversal(v);`
		`  }`
		`}`

Postorder (L-R-N)	•Visit the left subtree, if any.
	•Visit the right subtree, if any.
	•Visit the node
		`public void postorderTraversal(Visitor v){`
		`  if(!isEmpty() && ! v.isDone()){`
		`  getLeft().postorderTraversal(v) ;`
		`  getRight().postorderTraversal(v);`
		`  v.visit(getKey());`
		`  }`
		`}`

Depth-first Preorder Traversal

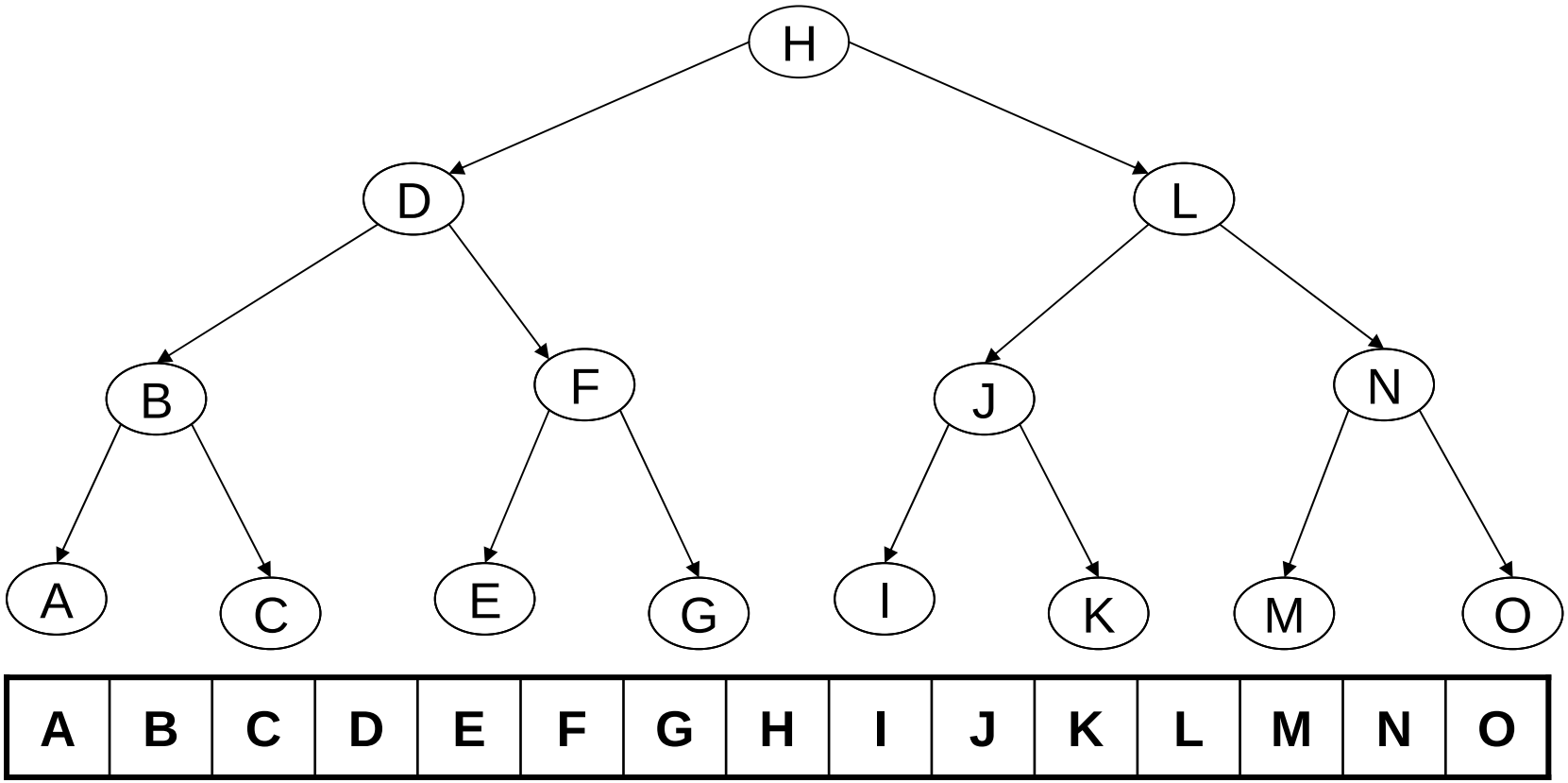
N-L-R



H	D	B	A	C	F	E	G	L	J	I	K	N	M	O

Depth-first Inorder Traversal

L-N-R

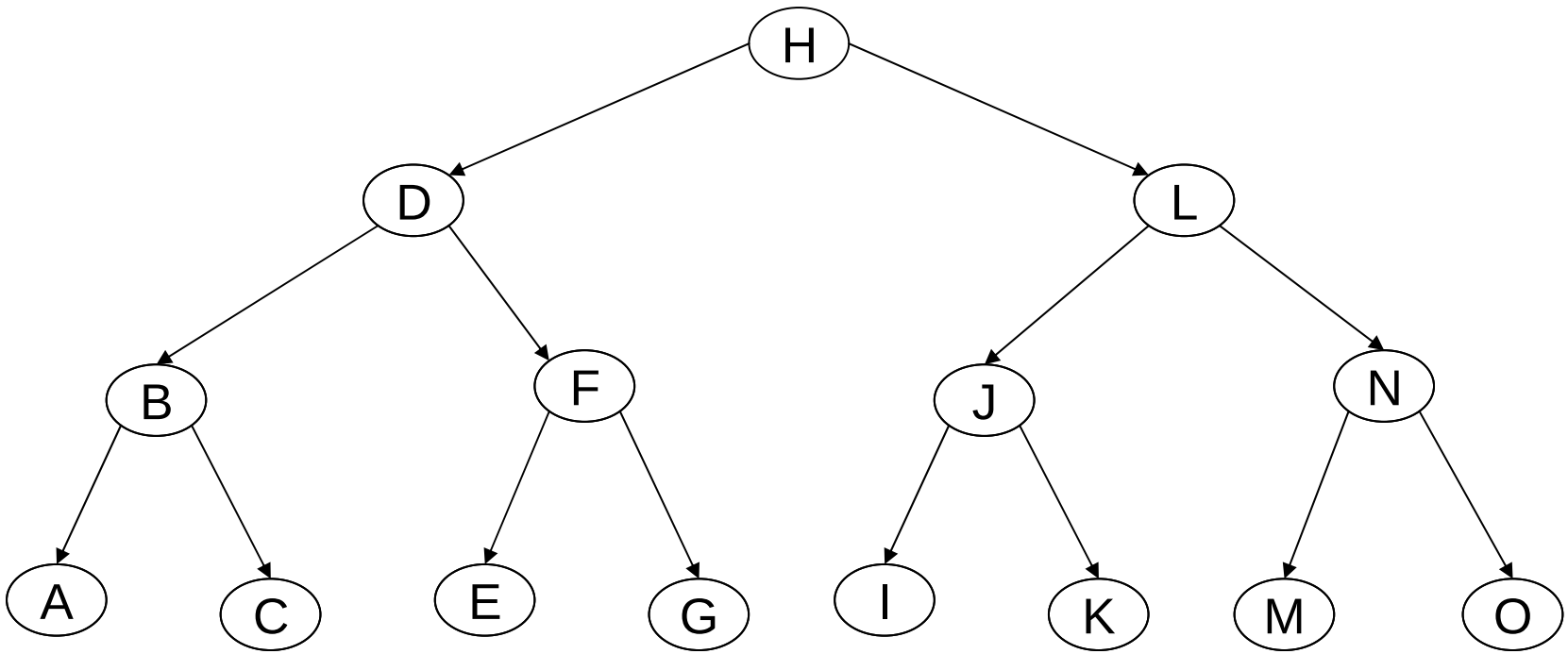


A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

Note: An inorder traversal of a BST visits the keys sorted in increasing order.

Depth-first Postorder Traversal

L-R-N



A	C	B	E	G	F	D	I	K	J	M	O	N	L	H

Traversals

- The following code illustrates how to display the contents of a Binary tree using each traversal method.

```
Visitor v = new PrintingVisitor() ;
BinaryTree t = new BinaryTree() ;
// . . .
t.breadthFirstTraversal(v) ;
t.postorderTraversal(v) ;
t.inorderTraversal(v) ;
t.postorderTraversal(v) ;
```

The accept method of the BinaryTree class

- Usually the accept method of a container is allowed to visit the elements of the container in any order.
- A depth-first tree traversal visits the nodes in either preoder or postorder and for Binary trees inorder traversal is also possible.
- The BinaryTree class accept method does a preorder traversal:

```
public void accept(Visitor visitor)
{
   preorderTraversal(visitor) ;
}
```

Binary Tree Iterator

- The BinaryTree class provides a tree iterator that does a preorder traversal. The iterator is implemented as an inner class:

```
private class BinaryTreeIterator implements Iterator{
      Stack stack;

   public BinaryTreeIterator(){
      stack = new StackAsLinkedList();
      if(!isEmpty())stack.push(BinaryTree.this);
   }

   public boolean hasNext(){return !stack.isEmpty();}

   public Object next(){
      if(stack.isEmpty())throw new NoSuchElementException();
      BinaryTree tree = (BinaryTree)stack.pop();
      if (!tree.getRight().isEmpty()) stack.push(tree.getRight());
      if (!tree.getLeft().isEmpty()) stack.push(tree.getLeft());
      return tree.getKey();
   }
}
```

Using a Binary Tree Iterator

- The iterator() method of the BinaryTree class returns a new instance of the BinaryTreeIterator inner class each time it is called:

```
public Iterator iterator(){
   return new BinaryTreeIterator();
}
```

- The following program fragment shows how to use a tree iterator:

```
BinaryTree tree = new BinaryTree() ;
// . . .
Iterator i = tree.iterator() ;
while(i.hasNext(){
   Object obj = e.next() ;
   System.out.print(obj + "   ") ;
}
```