12.5. Binary Tree Traversals ¶

12.5.1. Binary Tree Traversals

Often we wish to process a binary tree by “visiting” each of its nodes, each time performing a specific action such as printing the contents of the node. Any process for visiting all of the nodes in some order is called a **traversal**. Any traversal that lists every node in the tree exactly once is called an **enumeration** of the tree’s nodes. Some applications do not require that the nodes be visited in any particular order as long as each node is visited precisely once. For other applications, nodes must be visited in an order that preserves some relationship.

12.5.1.1. Preorder Traversal

For example, we might wish to make sure that we visit any given node *before* we visit its children. This is called a **preorder traversal**.

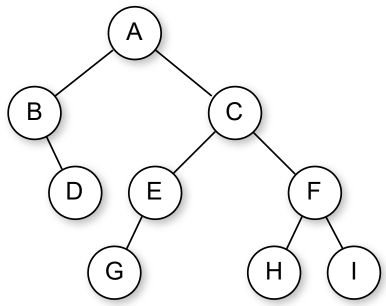


Figure 12.5.1: A binary tree for traversal examples.

Example 12.5.1

The preorder enumeration for the tree of Figure 12.5.1 is **A B D C E G F H I**.

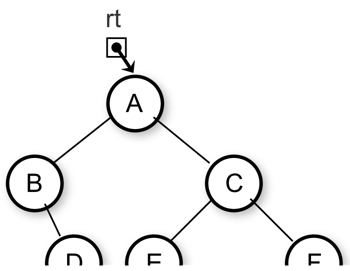
The first node printed is the root. Then all nodes of the left subtree are printed (in preorder) before any node of the right subtree.

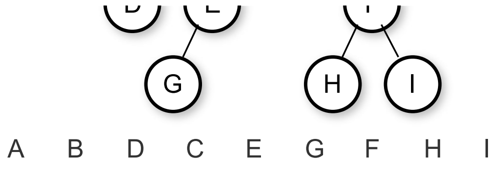
56 / 56

<< < > >>

Finished the preorder traversal on A.

```
static void preorder(BinNode rt) {
  if (rt == null) return; // Empty subtree - do nothing
  visit(rt);              // Process root node
  preorder(rt.left());    // Process all nodes in left
  preorder(rt.right());   // Process all nodes in right
}
```





12.5.1.2. Postorder Traversal

Alternatively, we might wish to visit each node only *after* we visit its children (and their subtrees). For example, this would be necessary if we wish to return all nodes in the tree to free store. We would like to delete the children of a node before deleting the node itself. But to do that requires that the children's children be deleted first, and so on. This is called a **postorder traversal**.

Example 12.5.2

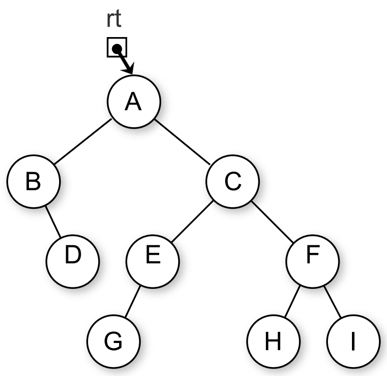
The postorder enumeration for the tree of Figure **12.5.1** is **D B G E H I F C A**.

1 / 56

<< < > >>

Postorder traversal begins.

```
static void postorder(BinNode rt) {
  if (rt == null) return;
  postorder(rt.left());
  postorder(rt.right());
  visit(rt);
}
```



12.5.1.3. Inorder Traversal

An **inorder traversal** first visits the left child (including its entire subtree), then visits the node, and finally visits the right child (including its entire subtree). The **binary search tree** makes use of this traversal to print all nodes in ascending order of value.

Example 12.5.3

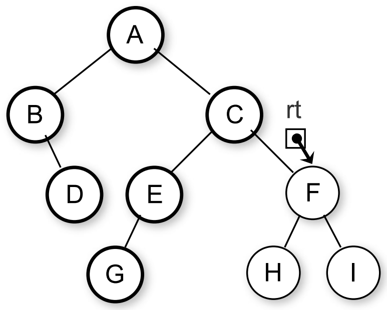
The inorder enumeration for the tree of Figure **12.5.1** is **B D A G E C H F I**.

35 / 56

<< < > >>

Make a recursive call on right child.

```
static void inorder(BinNode rt) {
  if (rt == null) return;
  inorder(rt.left());
  visit(rt);
  inorder(rt.right());
}
```



12.5.1.4. Implementation

Now we will discuss some implementations for the traversals, but we need to define a node ADT to work with. Just as a linked list is composed of a collection of link objects, a tree is composed of a collection of node objects. Here is an ADT for binary tree nodes, called `BinNode`. This class will be used by some of the binary tree structures presented later. Member functions are provided that set or return the element value, return a pointer to the left child, return a pointer to the right child, or indicate whether the node is a leaf.

Java | Java (Generic)

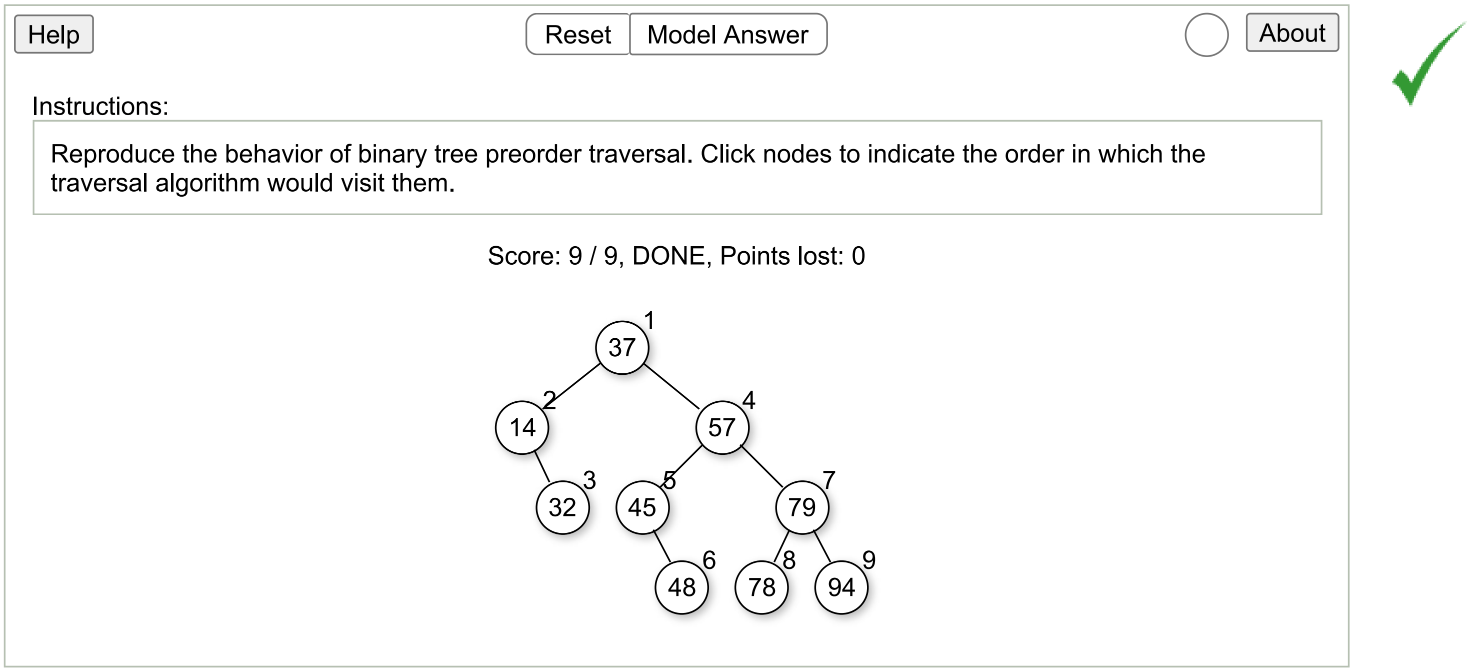
```java
interface BinNode { // Binary tree node ADT
  // Get and set the element value
  public Object value();
  public void setValue(Object v);

  // return the children
  public BinNode left();
  public BinNode right();

  // return TRUE if a leaf node, FALSE otherwise
  public boolean isLeaf();
}
```

A traversal routine is naturally written as a recursive function. Its input parameter is a pointer to a node which we will call `rt` because each node can be viewed as the root of a some subtree. The initial call to the traversal function passes in a pointer to the root node of the tree. The traversal function visits `rt` and its children (if any) in the desired order. For example, a preorder traversal specifies that `rt` be visited before its children. This can easily be implemented as follows.

Java | Java (Generic)

```
static void preorder(BinNode rt) {
  if (rt == null) return; // Empty subtree - do nothing
  visit(rt);              // Process root node
  preorder(rt.left());    // Process all nodes in left
  preorder(rt.right());   // Process all nodes in right
}
```

Function `preorder` first checks that the tree is not empty (if it is, then the traversal is done and `preorder` simply returns). Otherwise, `preorder` makes a call to `visit`, which processes the root node (i.e., prints the value or performs whatever computation as required by the application). Function `preorder` is then called recursively on the left subtree, which will visit all nodes in that subtree. Finally, `preorder` is called on the right subtree, visiting all nodes in the right subtree. Postorder and inorder traversals are similar. They simply change the order in which the node and its children are visited, as appropriate.



Instructions:

Reproduce the behavior of binary tree preorder traversal. Click nodes to indicate the order in which the traversal algorithm would visit them.

Score: 9 / 9, DONE, Points lost: 0

12.5.2. Postorder Traversal Practice

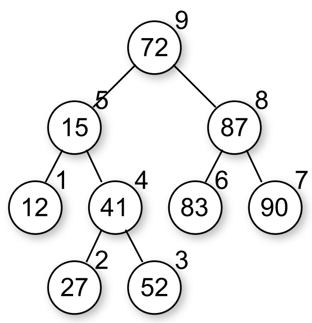
Help Reset Model Answer About

✓

Instructions:

Reproduce the behavior of binary tree postorder traversal. Click nodes to indicate the order in which the traversal algorithm would visit them.

Score: 9 / 9, DONE, Points lost: 0



12.5.3. Inorder Traversal Practice

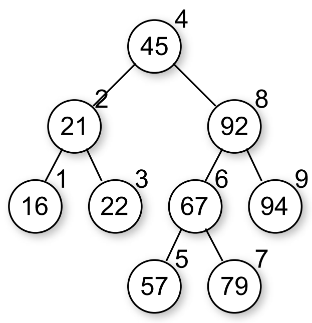
Help Reset Model Answer About

✓

Instructions:

Reproduce the behavior of binary tree inorder traversal. Click nodes to indicate the order in which the traversal algorithm would visit them.

Score: 9 / 9, DONE, Points lost: 0



12.5.4. Summary Questions