Many binary tree operations are done by performing a traversal of the binary tree. In a traversal, each element of the binary tree is visited exactly once. During the visit of an element, all action (make a clone, display, evaluate the operator, etc.) with respect to this element is taken.

Traversing Trees

Depth-first traversal: This strategy consists of searching deeper in the tree whenever possible. Certain depth-first traversals occurs frequently enough that they are given names of their own.

Breadth-first traversal: This is a very simple idea which consists of visiting the nodes based on their level in the tree. It visits all nodes with depth 0, then depth 1, then depth 2, and so on.

Special cases of depth-first traversal for binary trees

- In binary trees there are three basic ways to traverse a tree using the a depth-first search idea:
  - Preorder: We visit a node, then visit the left and the right subtrees
  - Inorder: We visit the left subtree then we visit the node, then we visit the right subtree
  - Postorder: We visit the left and right subtree and then we visit the node. This is what normally authors mean if they mention just depth-first traversal.
Binary Tree Traversal

Method

- Preorder
- Inorder
- Postorder

- In a traversal of a binary tree, each element of the binary tree is visited exactly once.
- During the visit of an element, all action (make a clone, display, evaluate the operator, etc.) with respect to this element is taken.

Preorder Traversal:

J E A H T M Y

Visit left subtree second
Visit right subtree last

Preorder Traversal (cont.)

Visit the root of the tree first, then visit the nodes in the left subtree, then visit the nodes in the right subtree.

```java
void preOrder(BinaryTreeNode t)
{
    if (t != null)
    {
        visit(t);
        preOrder(t.leftChild);
        preOrder(t.rightChild);
    }
}
```

Inorder Traversal:

A E H J M T Y

Visit left subtree first
Visit right subtree last

Inorder Traversal (cont.)

Visit the nodes in the left subtree, then visit the root of the tree, then visit the nodes in the right subtree.

```java
void inOrder(BinaryTreeNode t)
{
    if (t != null)
    {
        inOrder(Left(t));
        visit(t);
        inOrder(Right(t));
    }
}
```
Inorder Traversal (cont.)

```java
void inOrder(BinaryTreeNode t)
{
    if (t != null)
    {
        inOrder(t.leftChild);
        visit(t);
        inOrder(t.rightChild);
    }
}
```

Postorder Traversal

Visit the nodes in the left subtree first, then visit the nodes in the right subtree, then visit the root of the tree

```java
void postOrder(BinaryTreeNode t)
{
    if (t != null)
    {
        postOrder(t.leftChild);
        postOrder(t.rightChild);
        visit(t);
    }
}
```

Breadth-first traversal of a tree

A breadth-first traversal consists of visiting the nodes based on their level in the tree. It visits all nodes with level depth 0, then depth 1, then depth 2, and so on.

Use a queue to implement breadth-first traversal

Traversals Applications

- Make a clone
- Determine height
- Determine number of nodes
**Binary Tree Construction**

- Suppose that the elements in a binary tree are distinct.
- Can you construct the binary tree from which a given traversal sequence came?
- When a traversal sequence has more than one element, the binary tree is not uniquely defined.

**Some Examples**

<table>
<thead>
<tr>
<th>Preorder</th>
<th>Inorder</th>
<th>Postorder</th>
</tr>
</thead>
<tbody>
<tr>
<td>ab</td>
<td>ab</td>
<td>ab</td>
</tr>
</tbody>
</table>

**Preorder And Postorder**

- Preorder and postorder do not uniquely define a binary tree.
- Nor do preorder and level order (same example).
- Nor do postorder and level order (same example).