Binary Trees

1. **Binary Tree Examples**

Without telling you what a binary tree is, here are some examples (that I will draw on the board):

- The dots/circles are called **nodes** or **vertices** (singular: **vertex**).
- The lines (that connect 2 nodes) are called **edges**, or sometimes **arcs**.

2. **Binary Tree Definition**

We can define a binary tree recursively, as follows:

- A **binary tree** is either
  - an empty tree
  - a root node $r$, connected by an edge to up to two non-empty trees

Every node except the root has exactly one parent node (above it).

- The number of edges in a tree is one less than the number of nodes.

A tree has no cycles (i.e., loops).

- Every node has a unique path from the root to it.

A forest is a collection of trees.

3. **Tree Vocabulary**

Trees come with a lot of special vocabulary that you should get familiar with:

- A **leaf node** is a node with no non-empty children.
- An **internal node** is any node that is not a leaf node.
- A (simple) **path** is a sequence of consecutive edges that starts and ends at the same vertex (a tree is acyclic: there are no cycles).
- A **cycle** is a sequence of consecutive edges that starts and ends at the same vertex.
- The **length of a path** is the number of edges in that path.
- The **depth**, or level, of a node $v$ is the length of the path from the root to $v$.
- The height of a tree is the length of the longest path from the root to any node.
- The number of nodes in a path is $|v| + 1$, the number of edges in the path is $|v| - 1$.
- The number of nodes is the number of edges plus one (above it).
- A (simple) **path** is a sequence of consecutive edges.
- A **cycle** is a sequence of consecutive edges that starts and ends at the same vertex.
- A **node** is a node that is not a leaf node.

4. **Binary Tree Definition**

We can define a binary tree recursively, as follows:

- A **binary tree** is a tree where the edges are arrows from one node to the other.
- A root is a node that has no edges coming into it.
- A edge is a connection between two nodes.
- An empty tree is one where the edges are arrows from one node to the other.

A tree has no cycles (i.e., loops).

- Every node has a unique path from the root to it.

We can define a binary tree recursively, as follows:

- A **binary tree** is either
  - an empty tree
  - a root node $r$, connected by an edge to up to two non-empty trees

Every node except the root has exactly one parent node (above it).

- The number of edges in a tree is one less than the number of nodes.

A tree has no cycles (i.e., loops).

- Every node has a unique path from the root to it.

A forest is a collection of trees.

4. **Binary Tree Definition**

We can define a binary tree recursively, as follows:

- A **binary tree** is a tree where the edges are arrows from one node to the other.
- A root is a node that has no edges coming into it.
- A edge is a connection between two nodes.
- An empty tree is one where the edges are arrows from one node to the other.

A tree has no cycles (i.e., loops).

- Every node has a unique path from the root to it.

We can define a binary tree recursively, as follows:

- A **binary tree** is either
  - an empty tree
  - a root node $r$, connected by an edge to up to two non-empty trees

Every node except the root has exactly one parent node (above it).

- The number of edges in a tree is one less than the number of nodes.

A tree has no cycles (i.e., loops).

- Every node has a unique path from the root to it.

A forest is a collection of trees.

4. **Binary Tree Definition**

We can define a binary tree recursively, as follows:

- A **binary tree** is a tree where the edges are arrows from one node to the other.
- A root is a node that has no edges coming into it.
- A edge is a connection between two nodes.
- An empty tree is one where the edges are arrows from one node to the other.

A tree has no cycles (i.e., loops).

- Every node has a unique path from the root to it.

We can define a binary tree recursively, as follows:

- A **binary tree** is either
  - an empty tree
  - a root node $r$, connected by an edge to up to two non-empty trees

Every node except the root has exactly one parent node (above it).

- The number of edges in a tree is one less than the number of nodes.

A tree has no cycles (i.e., loops).

- Every node has a unique path from the root to it.

A forest is a collection of trees.

4. **Binary Tree Definition**

We can define a binary tree recursively, as follows:

- A **binary tree** is a tree where the edges are arrows from one node to the other.
- A root is a node that has no edges coming into it.
- A edge is a connection between two nodes.
- An empty tree is one where the edges are arrows from one node to the other.

A tree has no cycles (i.e., loops).

- Every node has a unique path from the root to it.

We can define a binary tree recursively, as follows:

- A **binary tree** is either
  - an empty tree
  - a root node $r$, connected by an edge to up to two non-empty trees

Every node except the root has exactly one parent node (above it).

- The number of edges in a tree is one less than the number of nodes.

A tree has no cycles (i.e., loops).

- Every node has a unique path from the root to it.

A forest is a collection of trees.

4. **Binary Tree Definition**

We can define a binary tree recursively, as follows:

- A **binary tree** is a tree where the edges are arrows from one node to the other.
- A root is a node that has no edges coming into it.
- A edge is a connection between two nodes.
- An empty tree is one where the edges are arrows from one node to the other.

A tree has no cycles (i.e., loops).

- Every node has a unique path from the root to it.

We can define a binary tree recursively, as follows:

- A **binary tree** is either
  - an empty tree
  - a root node $r$, connected by an edge to up to two non-empty trees

Every node except the root has exactly one parent node (above it).

- The number of edges in a tree is one less than the number of nodes.

A tree has no cycles (i.e., loops).

- Every node has a unique path from the root to it.

A forest is a collection of trees.

4. **Binary Tree Definition**

We can define a binary tree recursively, as follows:

- A **binary tree** is a tree where the edges are arrows from one node to the other.
- A root is a node that has no edges coming into it.
- A edge is a connection between two nodes.
- An empty tree is one where the edges are arrows from one node to the other.

A tree has no cycles (i.e., loops).

- Every node has a unique path from the root to it.

We can define a binary tree recursively, as follows:

- A **binary tree** is either
  - an empty tree
  - a root node $r$, connected by an edge to up to two non-empty trees

Every node except the root has exactly one parent node (above it).

- The number of edges in a tree is one less than the number of nodes.

A tree has no cycles (i.e., loops).

- Every node has a unique path from the root to it.

A forest is a collection of trees.
Binary Tree Operations

A binary tree stores data in its nodes, like a linked list. Some tree operations:

- `size()` - return the number of nodes in the tree
- `isEmpty()` - return whether this tree has any nodes
- `isLeaf()` - return whether this tree is a single leaf node
- `height()` - return the height of the tree
- `leafCount()` - return the number of leaves in the tree
- `nodeCount()` - return the number of nodes in the tree
- `mirrorImage()` - return a tree that is the mirror image of this tree
- `traverse()`/`iterate()` - go through the nodes of the tree in a specific order
- Many, many others

Recursive Binary Tree Implementation

Just like the recursive LL implementation, we'll have three separate public classes:

```java
public abstract class BinaryTree<T> {
    // method headers for Binary Tree methods go here
}
public class EmptyTree<T> extends BinaryTree<T> {
    // method implementations for empty trees go here
}
public class ConsTree<T> extends BinaryTree<T> {
    T data    // the data of the root node
    BinaryTree<T> left    // the left subtree
    BinaryTree<T> right   // the right subtree
    // method implementations for non-empty trees go here
}
```

To add a method to our BinaryTree class, we must do three things:

- Add the method as an abstract method header in the BinaryTree class.
- Implement the method in the EmptyTree class.
- Implement the method in the ConsTree class.

For example, let's start with the `isEmpty()` method.

```java
public abstract class BinaryTree<T> {
    public abstract boolean isEmpty();
}
```

In the EmptyTree class, we implement the method:

```java
public class EmptyTree<T> {
    public boolean isEmpty() { return true; } // always.
}
```

In the ConsTree class, we implement the method:

```java
public class ConsTree<T> {
    public boolean isEmpty() { return false; } // always.
}
```

Now let's try the `size()` method that returns the number of nodes in the tree.

```java
public abstract class BinaryTree<T> {
    public abstract int size();
}
```

In the EmptyTree class, we implement the method:

```java
public class EmptyTree<T> {
    public int size() { return 0; } // "base case"
}
```

We implement size in the Cons class, where we use recursion to implement size correctly:

```java
public class ConsTree<T> {
    T data
    BinaryTree<T> left
    BinaryTree<T> right
    public int size() {
        return 1 + left.size() + right.size();
    }
}
```

Just like the recursive LL implementation, we have three separate public classes.

Recursive Binary Tree Implementation
Recursive Binary Tree Implementation

Now let's try the height() method, which is the length of the longest root-leaf path in the tree.

```java
class BinaryTree<T> {
    public abstract int height();
}
```

```java
class EmptyTree<T> {
    public int height() { return -1; }
}
```

```java
class ConsTree<T> {
    T data;
    BinaryTree<T> left;
    BinaryTree<T> right;
    public int size() {
        return 1 + Math.max(left.height(), right.height());
    }
}
```

Binary Tree Traversals

There are many ways to traverse the tree such that each node is "visited" exactly once, in some order. The three steps for common tree traversals:

1. visit the current node
2. recursively traverse the left subtree
3. recursively traverse the right subtree

Different orderings of these three steps leads to three different traversals!

- **Pre-order traversal**: 1-2-3 (traverse the root node FIRST (pre), then do the recursive calls.)
- **Post-order traversal**: 2-3-1 (do the recursive calls first, then traverse the root node (POST).)
- **In-order traversal**: 2-1-3 (traverse the root node IN between the two recursive calls.)

A breadth-first search (BFS) is a level-order traversal, where you visit the nodes in the tree level-by-level from the root down to the leaves. A depth-first search (DFS) visits nodes along one path until it hits a leaf, then backs up as possible before going down a different path to a leaf. A level-order traversal is a level-order traversal, where you visit the nodes in the tree level-by-level from the root down to the leaves. A breadth-first search (BFS) is a level-order traversal. Where you visit the nodes in the tree level-by-level from the root down to the leaves is the same as preorder.

Now let's try the height() method, which is the length of the longest root-leaf path in the tree.

```java
public abstract class BinaryTree<T> {
    public abstract int height();
}
```

```java
class EmptyTree<T> {
    public int height() { return -1; }
}
```

```java
class ConsTree<T> {
    T data;
    BinaryTree<T> left;
    BinaryTree<T> right;
    public int size() {
        return 1 + Math.max(left.height(), right.height());
    }
}
```