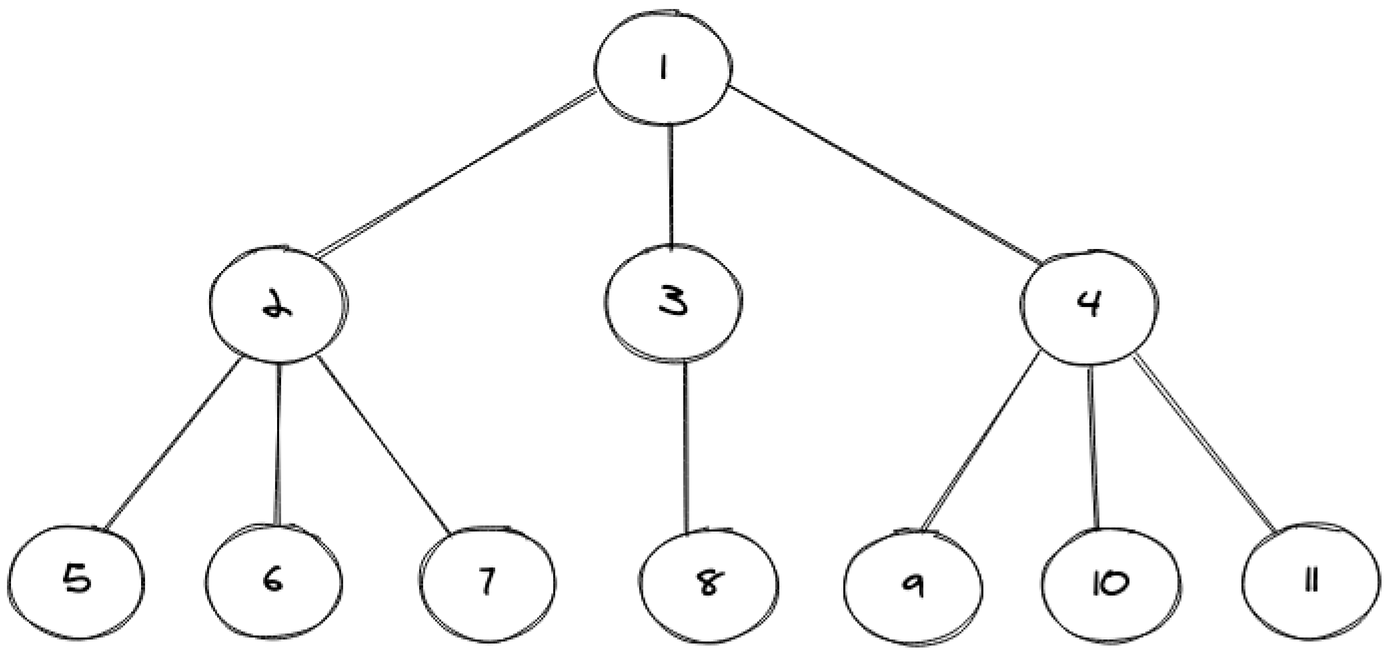
N-ary Tree Data Structure

The N-ary tree is a tree that allows us to have `n` number of children of a particular node, hence the name **N-ary**, making it **slightly complex than the very common binary trees** that allow us to have at most 2 children of a particular node.

A pictorial representation of an N-ary tree is shown below:



In the N-ary tree shown above, we can notice that there are **11 nodes** in total and **some nodes have three children** and some have had one only. In the case of a binary tree, it was easier to store these children nodes as we can assign two nodes (i.e. left and right) to a particular node to mark its children, but here it's not that simple. To store the children nodes of any tree node we make use of another data structure, mainly vector in C++ and LinkedList in Java.

Implementation of N-ary Tree

The first thing that we do when we are dealing with non-linear data structures is to create our own structure (constructors in Java) for them. Like in the case of a binary tree, we make use of a class `TreeNode` and inside that class, we create our constructors and have our class-level variables.

Consider the code snippet below:

```
public static class TreeNode{

        int val;

        List<TreeNode> children = new LinkedList<>();



        TreeNode(int data){

            val = data;

        }



        TreeNode(int data,List<TreeNode> child){

            val = data;

            children = child;
        }

}
```

Copy
In the above code snippet, we have a class named `TreeNode` which in turn contains two constructors with the same name, but they are overloaded (same method names but with different parameters) in nature. We also have two identifiers, one of them is the val that stores the value of any particular node, and then we have a List to store the children nodes of any node of the tree.

The above snippet contains the basic structure of our tree, and now are only left with making one and then later we will see how to print the tree using level order traversal. To build a tree we will make use of the constructors that we have defined in the above classes.

Consider the code snippet below:

```
public static void main(String[] args) {

        // creating an exact replica of the above pictorial N-ary Tree

        TreeNode root = new TreeNode(1);

        root.children.add(new TreeNode(2));

        root.children.add(new TreeNode(3));

        root.children.add(new TreeNode(4));

        root.children.get(0).children.add(new TreeNode(5));

        root.children.get(0).children.add(new TreeNode(6));

        root.children.get(0).children.add(new TreeNode(7));

        root.children.get(1).children.add(new TreeNode(8));

        root.children.get(2).children.add(new TreeNode(9));

        root.children.get(2).children.add(new TreeNode(10));

        root.children.get(2).children.add(new TreeNode(11));

        printNAryTree(root);

}
```

Copy
First thing first, we created the **root node of our N-ary Tree**, then we have to assign some children to this root node, we do this by making use of the dot(`.`) operator and accessing the children property of the root node and then adding different children to the root nodes by using the add method provided by the `List` interface. Once we have added all the children of the root node, it is time to add children of each of the new level nodes, we do that by first accessing that node by using the get method provided by the list interface and then adding the respective children nodes to that node.

Lastly, we print this N-ary Tree, we did it by calling the `printNAryTree` method.

Now, since printing a tree is not as simple as looping through a collection of items, we have different techniques (algorithms precisely) at our disposal. These are mainly:

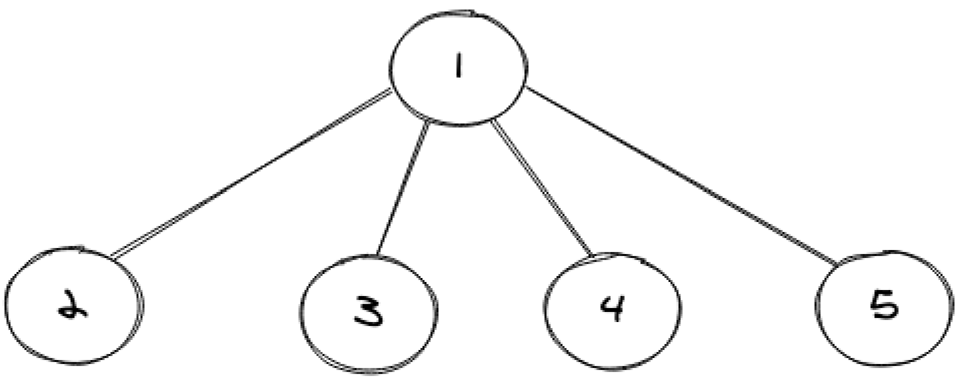
- **Inorder Traversal**
- **Preorder Traversal**
- **PostOrder Traversal**
- **Level Order Traversal**

For the sake of this tutorial, we will use the Level Order Traversal approach, as it is easier to understand, provided if you have seen how it works on a binary tree before.

Level Order Traversal (Printing the N-ary Tree)

The Level Order traversal of any tree takes into the fact that we want to print the nodes at a root level first, then move on to the next level and keep repeating this process until we are at the last level. We make use of the Queue data structure to store the nodes at a particular level.

Consider a simple N-ary Tree shown below:



Level Order Traversal for the above tree will look like this:

```
1

2 3 4 5
```

Copy
Consider the code snippet below:

```
private static void printNAryTree(TreeNode root){

        if(root == null) return;

        Queue<TreeNode> queue = new LinkedList<>();

        queue.offer(root);

        while(!queue.isEmpty()) {

            int len = queue.size();

            for(int i=0;i<len;i++) { // so that we can reach each level

                TreeNode node = queue.poll();

                System.out.print(node.val + " ");

                for (TreeNode item : node.children) { // for-Each loop
to iterate over all childrens

                    queue.offer(item);
                }

            }

            System.out.println();

        }

}
```
Copy
The entire code looks something like this:

```
import java.util.LinkedList;

import java.util.List;

import java.util.Queue;


public class NAryTree {

    public static class TreeNode{

        int val;

        List<TreeNode> children = new LinkedList<>();


        TreeNode(int data){

            val = data;

        }


        TreeNode(int data,List<TreeNode> child){

            val = data;

            children = child;

        }

    }


    private static void printNAryTree(TreeNode root){

        if(root == null) return;

        Queue<TreeNode> queue = new LinkedList<>();

        queue.offer(root);

        while(!queue.isEmpty()) {
            int len = queue.size();

            for(int i=0;i<len;i++) {

                TreeNode node = queue.poll();

                assert node != null;

                System.out.print(node.val + " ");

                for (TreeNode item : node.children) {

                    queue.offer(item);

                }

            }

            System.out.println();

        }

    }


    public static void main(String[] args) {

        TreeNode root = new TreeNode(1);

        root.children.add(new TreeNode(2));

        root.children.add(new TreeNode(3));

        root.children.add(new TreeNode(4));

        root.children.get(0).children.add(new TreeNode(5));

        root.children.get(0).children.add(new TreeNode(6));

        root.children.get(0).children.add(new TreeNode(7));

        root.children.get(1).children.add(new TreeNode(8));

        root.children.get(2).children.add(new TreeNode(9));

        root.children.get(2).children.add(new TreeNode(10));

        root.children.get(2).children.add(new TreeNode(11));

        printNAryTree(root);

    }

}
```
Copy
The output of the above code is:

```
1

2 3 4

5 6 7 8 9 10 11
```

We can compare this output to the pictorial representation of the N-ary tree we had in the starting, each level node contains the same values.

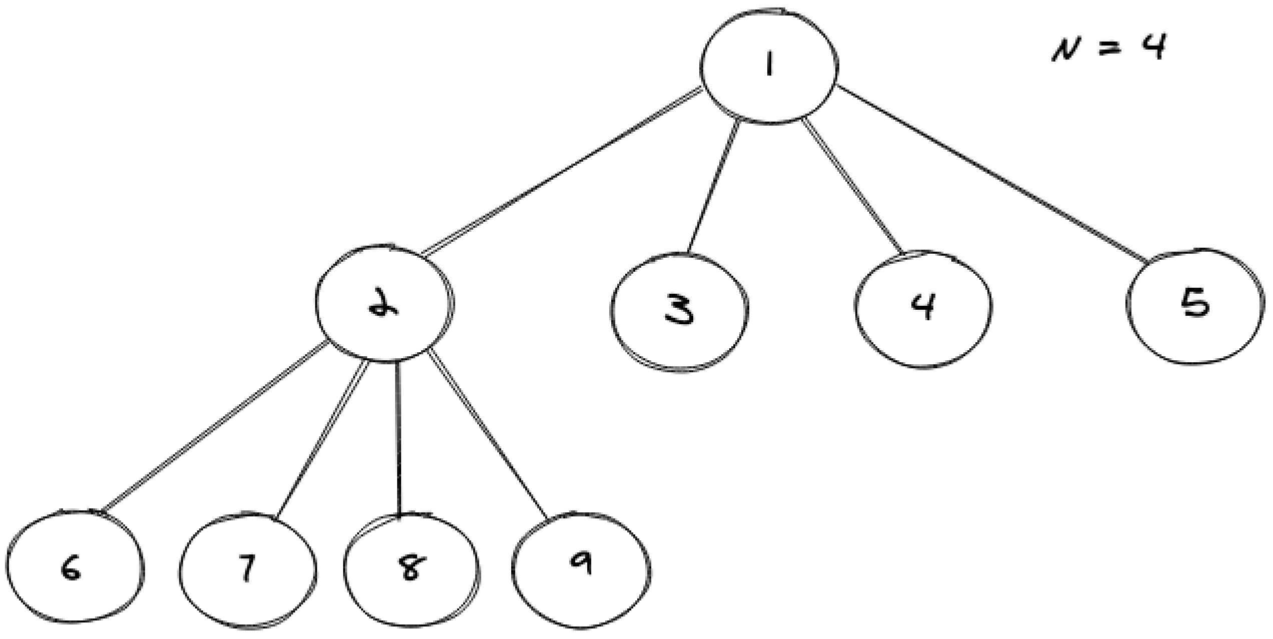
Types of N-ary Tree

Following are the types of N-ary tree:

1. Full N-ary Tree

A Full N-ary Tree is an N-ary tree that allows each node to have either 0 or N children.

Consider the pictorial representation of a full N-ary tree shown below:

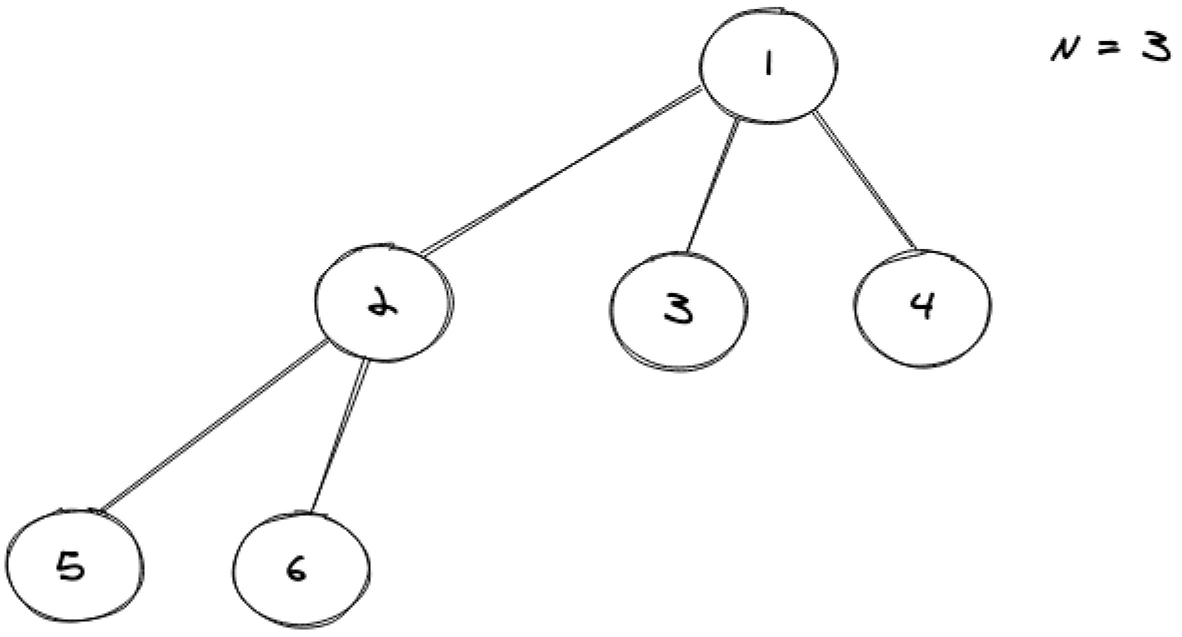


Notice that all the nodes of the above N-ary have either 4 children or 0, hence satisfying the property.

2. Complete N-ary Tree

A complete N-ary tree is an N-ary tree in which the nodes at each level of the tree should be complete (should have exactly **N children**) except the last level nodes and if the last level nodes aren't complete, the nodes must be "as left as possible".

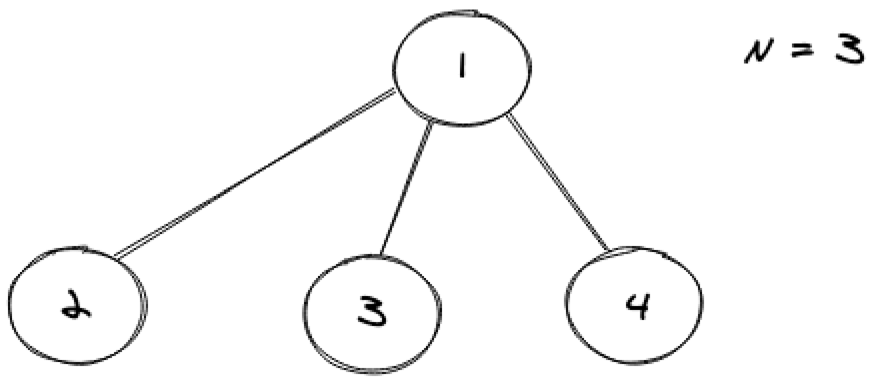
Consider the pictorial representation of a complete N-ary tree shown below:



3. Perfect N-ary Tree

A perfect N-ary tree is a full N-ary tree but the level of the leaf nodes must be the same.

Consider the pictorial representation of a perfect N-ary tree shown below:



Conclusions

- We learned what an N-ary tree is.
- We also learned how to implement an N-ary tree in Java(via level order traversal).
- Then we learned what different types of N-ary trees are there in total.