☆ Generic trees / Nary trees.

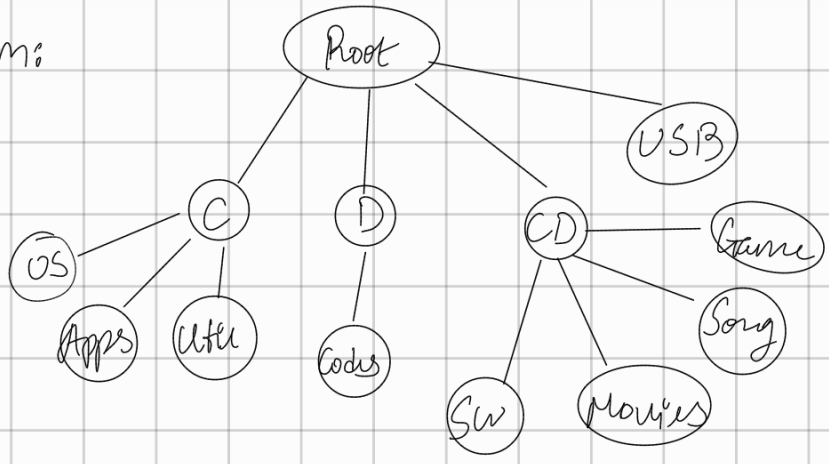
Data structures

- → Array / String } Contiguous
- → Array List / String Builder } Contiguous
- → Linked list } Non Contiguous.
- → Stack & Queue } Non Contiguous.

All of the above: Linear

- → Generic tree / Nary tree } Non contiguous & Non linear
- → Binary tree
- → Binary tree

Eg: File System:



For a tree node, the structure can be

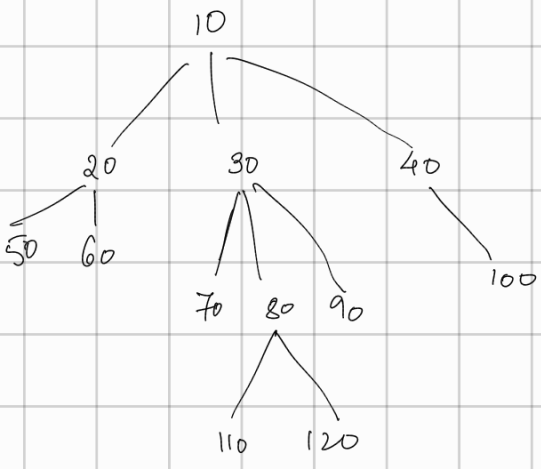
```
TreeNode {
    int data;
    List<TreeNode> children;
}
```

CODE

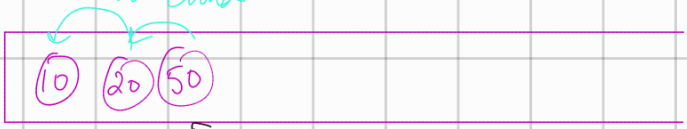
```
private static class Node {
    int data;
    List<Node> children;

    Node() {
        this.data = 0;
        this.children = new ArrayList<>();
    }

    Node(int data) {
        this.data = data;
        this.children = new ArrayList<>();
    }
}
```

abstraction

[10 20 50 -1 60 -1 -1 30
70 -1 80 110 -1 120 -1
-1 90 -1 -1 40 100 -1
-1 -1]



Stack



for(-1) No child, so pop 50

If arr[i] != -1 , add child & push in stack
else , pop from stack

CODE

```
public static Node construct(int[] arr) {
    Node root = new Node(arr[0]);
    Stack<Node> st = new Stack<>();
    st.push(root);
    for(int i=1; i<arr.length; i++) {
        if(arr[i] == -1) st.pop();
        else {
            Node currParent = st.peek(), currChild = new Node(arr[i]);
            currParent.children.add(currChild);
            st.push(currChild);
        }
    }
    return root;
}
```

→ Can also use st.size()>0 if arr might have extra elements

③ Pre & Post Order Traversal

```
public static void traversals(Node node) {
    Sysо("Node Pre " + node.data);
    for(Node child : node.children) {
        Sysо("Edge Pre " + node.data + "--" + child.data);
        traversals(child);
        Sysо("Edge Post" + node.data + "--" + child.data);
    }
    Sysо("Node Post" + node.data);
}
```

④ Max of Tree

CODE

```
int max = root.data;
for(Node child : root.children)
    max = Math.max(max, findMax(child));
return max;
```