PES UNIVERSITY

Department of Computer Science and Engineering

PES UNIVERSITY

UE19CS202: Data Structures and its Applications (4-0-0-4-4)

Trees
N-ary Trees and Forest

Dr. Shylaja S S
Ms. Kusuma K V

Tree: is a Non Linear Data Structure

Definition: Finite nonempty set of elements

- One element is the root
- Remaining elements are partitioned into m≥0 disjoint subsets each of which is itself a tree

Ordered Tree: a tree in which subtrees of each node form an ordered set

- In such a tree we define first, second, ..., last child of a particular node
- First child is called the oldest child and last child the youngest child

Figure 1 shows an ordered tree with A as its root, B is the oldest child and D is the youngest child of A. E is the oldest and F is the youngest child of B.

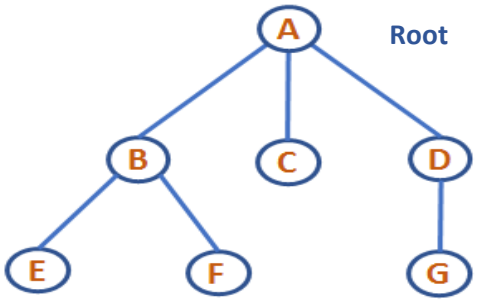


Figure 1: Ordered Tree

Note: Here on we will refer an ordered tree as just a tree

n-ary tree: A rooted tree in which each node has no more than n children.

A binary tree is an n-ary tree with n=2. Figure 2 shows an n-ary tree with n=5.

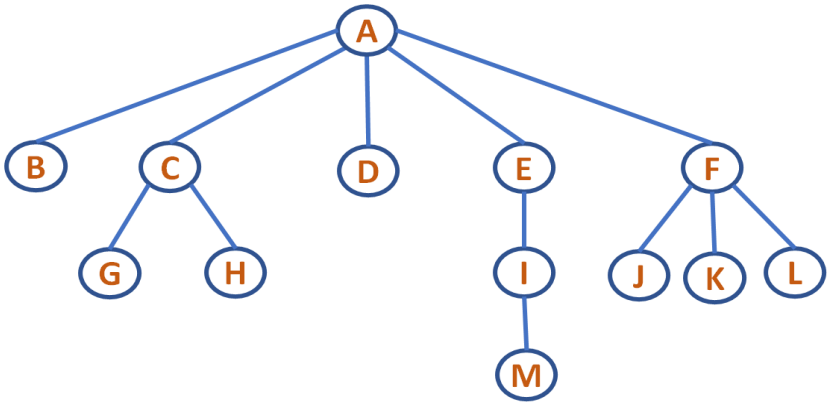


Figure 2: n-ary tree, n=5

Forest: is an ordered set of ordered trees

In the representation of a binary tree, each node contains an information field and two pointers to its two children. Trees may be represented as an array of tree nodes or a dynamic variable may be allocated for each node created. But how many pointers should a tree node contain? The number of children a node can have varies and may be as large or as small as desired.

```
#define MAXCHILD 20
struct treenode{
        int info;
        struct treenode *child[MAX];
};
```

where MAX is a constant

Restriction with the above implementation is that a node cannot have more than MAX children. Therefore the tree cannot be expanded.

Consider an alternative implementation as follows: All the children of a given node are linked and only the oldest child is linked to the parent

A node has link to first child and a link to immediate sibling

```
struct treenode{
        int info;
        struct treenode *child;
        struct treenode *sibling;
};
```

Conversion of an n-ary tree to a Binary Tree

Using Left Child – Right Sibling Representation an n-ary tree can be converted to a binary tree as follows:

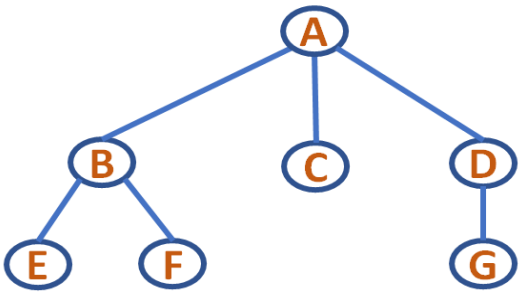
1) Link all the siblings of a node

2) Delete all links from a node to its children except for the link to its leftmost child

The **left child in binary tree** is the node which is the oldest child of the given node in an n-ary tree, and the **right child is the node to the immediate right of the given node** on the same horizontal line. Such a binary tree will not have a right sub tree.

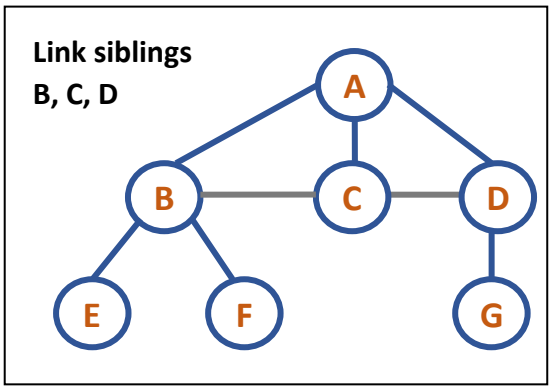
The node structure looks as shown below:

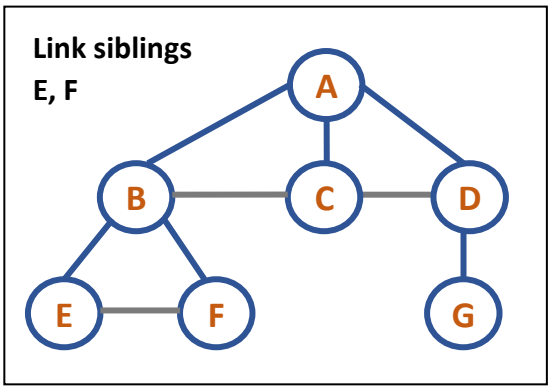
Consider the 3-ary tree shown below:



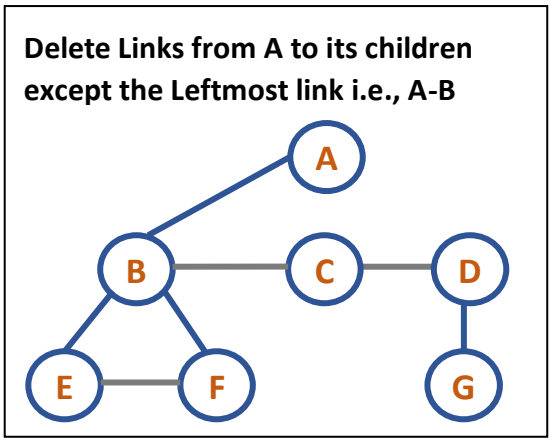
3 - ary tree

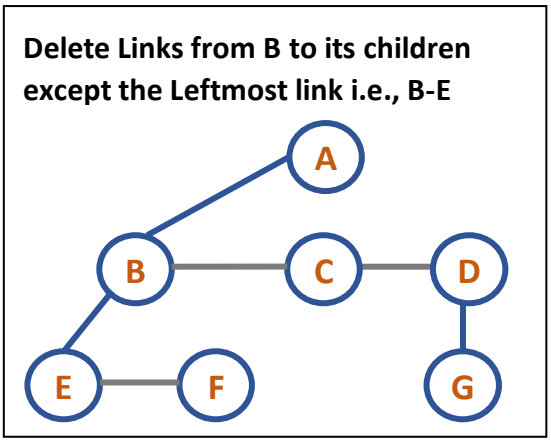
First step in the conversion is to Link all the siblings of a node.





Second step is to delete all the links from a node to its children except for the link to its leftmost child.

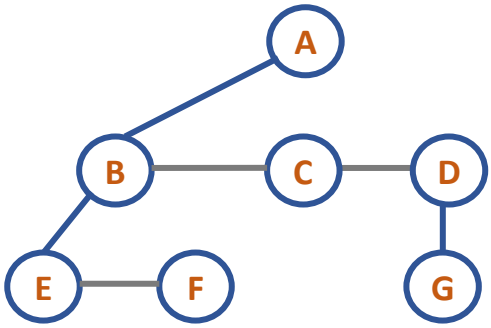




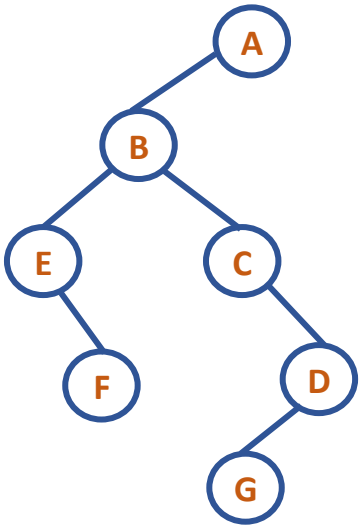
There are no more multiple links from any parent node to its children. The tree so obtained is the binary tree. But it doesn't look like one. Use the left child-right sibling relationship and make the tree look like a binary tree i.e., for any given node, its leftmost child will become the left child and immediate sibling becomes the right child. The binary tree so obtained will always have an empty right subtree for the root node. This is because the root of the tree we are transforming has no siblings.

For node A: Left child is B and has no Right child
For Node B: Left child is E and Right child is C
For Node C: No Left child and Right child is D
For node D: Left child is G and has no Right child
For Node E: No Left child and Right child is F
For Node F and Node G: No children (No Left child: because they do not have a child and no Right child: because they do not have a sibling towards right)

Left Child – Right Sibling representation of the given 3-ary tree



Corresponding Binary Tree representation of the given 3-ary tree



Conversion of a Forest to a Binary Tree

In the n-ary tree to binary tree conversion as stated above we saw that the right subtree for the root node of the binary tree is always empty. This is because the root of the tree we are transforming has no siblings.

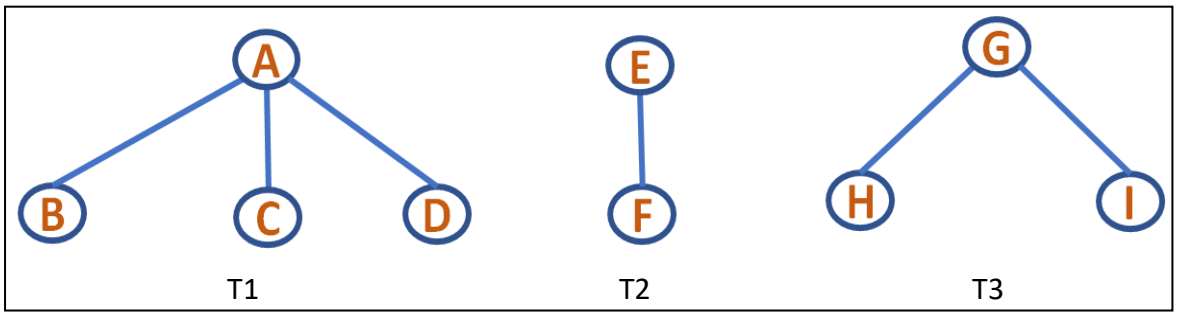
On the other hand, if we have a forest then these can all be transformed into a single binary tree as follows:

1) First obtain the binary tree representation of each of the trees in the forest
2) Link all the binary trees together through the right sibling field of the root nodes

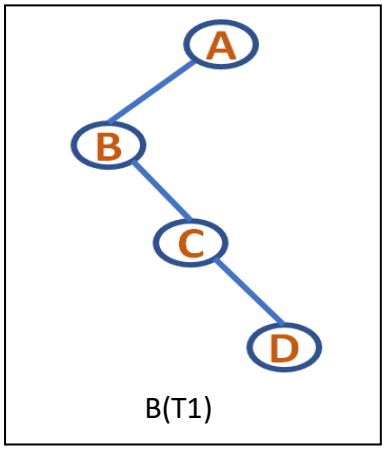
Conversion of a Forest to a Binary Tree can be formally defined as follows:

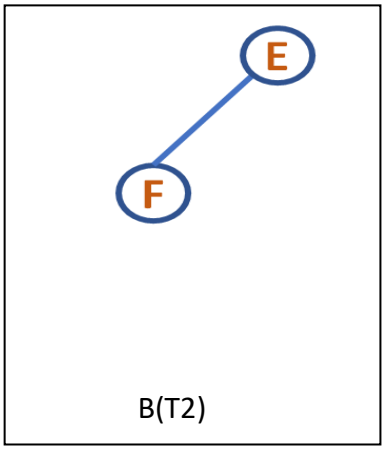
- If $T_1,...,T_n$ is a forest of n trees, then the binary tree corresponding to this forest, denoted by $B(T_1, ..., T_n)$:
  - is empty if $n = 0$
  - has root equal to root (T_1)
  - has left subtree equal to $B(T_{11}, T_{12}, ...,T_{1m})$ where T_{11} ,..., T_{1m} are the subtrees of root(T_1)
  - has right subtree $B(T_2, ..., T_n)$

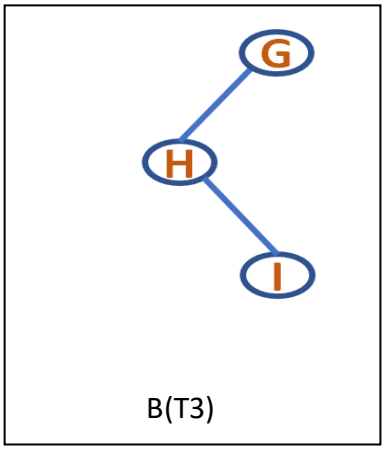
Consider the following Forest with three Trees:

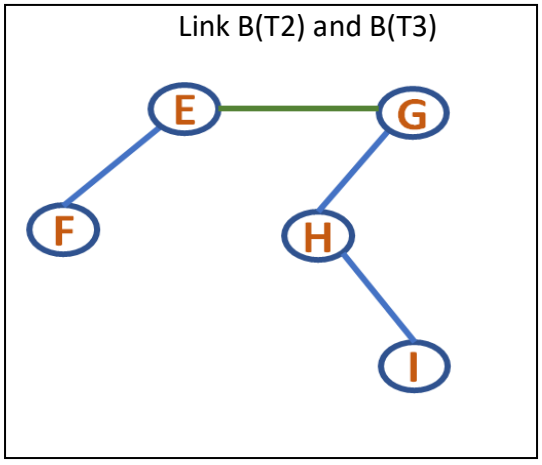


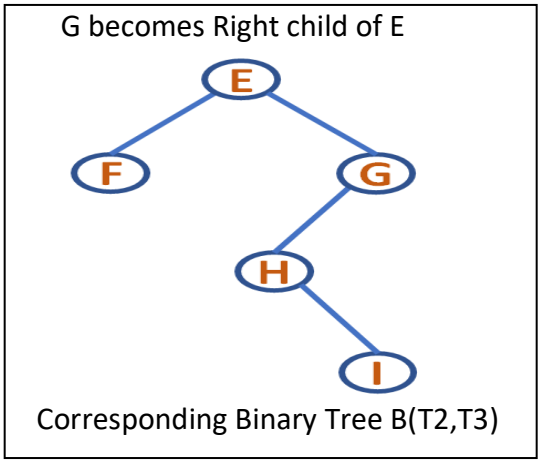
Corresponding Binary Trees:

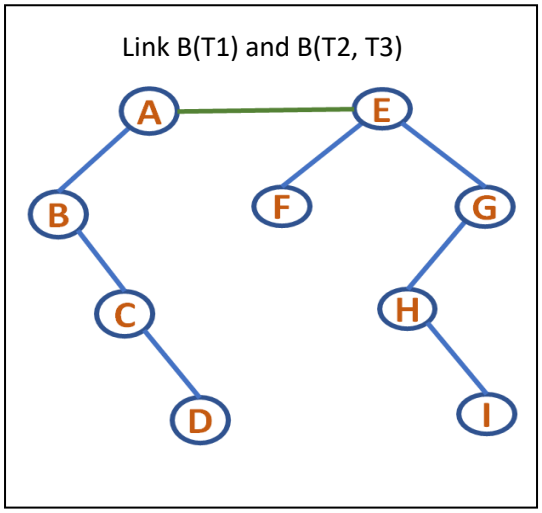


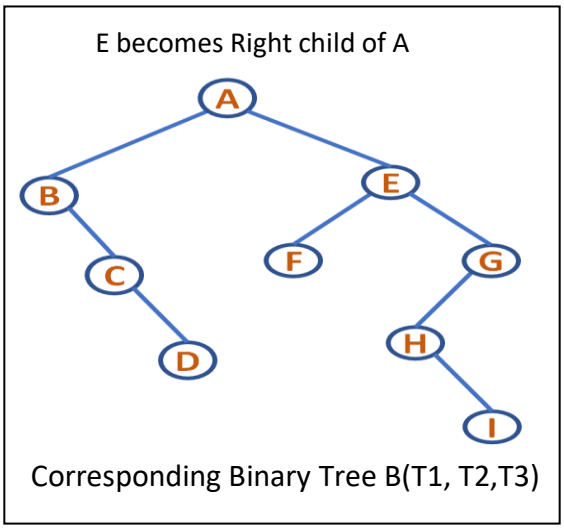












Tree Traversal

The traversal methods for binary trees induce traversal methods for forests. The preorder, inorder, or postorder traversals of a forest may be defined as the preorder, inorder, or postorder traversals of its corresponding binary tree.

```
struct treenode{
        int info;
        struct treenode *child;
        struct treenode *sibling;
};
```

With the treenode implemented as having pointers to first child and immediate sibling, the traversal preorder, inorder and postorder for a tree are defined as below:

Preorder:

1. Visit the root of the first tree in the forest
2. Traverse in preorder the forest formed by the subtrees of the first tree, if any
3. Traverse in preorder the forest formed by the remaining trees in the forest, if any

```
void preorder(TREE *root)
{
   if(root!=NULL)
   {
     printf(" %d ",root->info);
     preorder(root->child);
     preorder(root->sibling);
  }
}
```

Inorder:

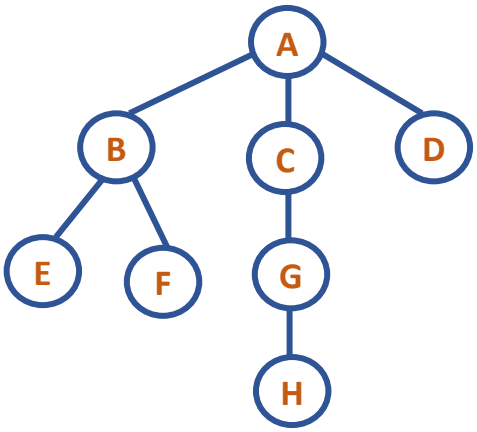
1. Traverse in inorder the forest formed by the subtrees of the first tree, if any
2. Visit the root of the first tree in the forest
3. Traverse in inorder the forest formed by the remaining trees in the forest, if any

```
void inorder(TREE *root)
{
  if(root!=NULL)
    {
      inorder(root->child);
      printf(" %d ",root->info);
      inorder(root->sibling);
    }
}
```

Postorder:

1. Traverse in postorder the forest formed by the subtrees of the first tree, if any
2. Traverse in postorder the forest formed by the remaining trees in the forest, if any
3. Visit the root of the first tree in the forest

```
void postorder(TREE *root)
{
  if(root!=NULL)
    {
      postorder(root->child);
      postorder(root->sibling);
      printf(" %d ", root->info);
    }
}
```



Traversal of the above n-ary Tree:

Preorder: ABEFCGHD

Inorder: EFBHGCDA

Postorder: FEHGDCBA