Unit 4 Trees and Binary Trees

Structure:

4.1 Introduction

In the previous unit, we have discussed about the linear data structure, stack and queue and also we discussed about the array and linked representation of stack and queue. In this unit we are going to extend the concept of linked data structure (linked list, stack, and queue) to a structure that may have multiple relations among its nodes. Such a structure is called a "**Tree"**. This unit begins with a detailed description of trees from basic concepts to their representation and manipulation. We are also going to discuss about the variant of tree called "**Binary tree"** with its representation in the memory and various operations that can be performed on the binary tree.

Objectives:

After studying this unit you should be able to

- discuss the structure and terminology of tree structure

- define the concept of binary tree
- list and discuss some of the types of binary tree
- describe the different traversal on binary tree
- explain the various representations of binary tree

4.2 Tree: Definition and Concepts

A tree is a non-linear data structure that consists of a root node and potentially many levels of additional nodes that form a hierarchy. A tree can be empty with no nodes called the null or empty tree or a tree is a structure consisting of one node called the **root** and one or more subtrees as referred in the figure 4.1

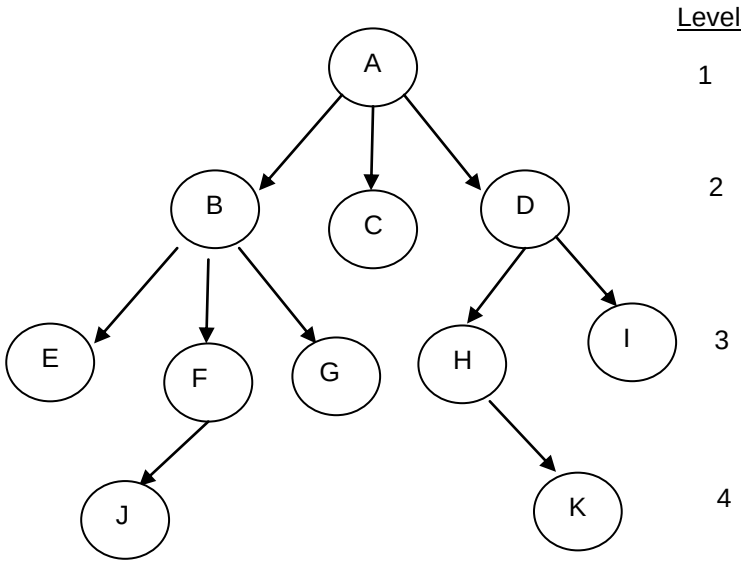

Figure 4.1: Tree structure

General properties of trees are:

- One node is distinguished as a root.
- Every node (exclude a root) is connected by a directed edge from exactly one other node; direction is: *parent -> children*

Now we will see the various terminology used in tree with reference to figure 4.1

Root **:** top most node in a tree. (node A)

parent **:** node that has a child. (nodes A, B, D, F, H)

***siblings* :** node with the same parent. (nodes E, F, G siblings of B)

***leaves* :** nodes with no children. (nodes C, E, G, I, J, K)

Internal nodes: nodes with atleast one child. (nodes B, D, F, H)

degree : number of sub trees of a node (node A has 3 degree)

edge : connection between one node to another

height: The level of a node is defined by initially letting the root be at level 1. Then children are at level l+1. Figure 4.1 shows the levels of all the nodes in that tree. The ***height*** or ***depth*** of a tree is defined to be the maximum level of any node in the tree.

forest: A *forest* is a set of $n \geq 0$ disjoint trees. The notion of a forest is very close to that of a tree because if we remove the root of a tree we get a forest. For example, in figure 4.1 if we remove *A* we get a forest with three trees.

There are other useful ways to draw a tree. One useful way is as a list. The tree of figure 4.1 could be written as the list. The information in the root node comes first followed by a list of the subtrees of that node.

(A(B(E,F(J),G),(C),D(H(K),I)))

Now, we will see how do we represent a tree in memory? If we wish to use linked lists, then a node must have a varying number of fields depending upon the number of branches.

|DATA |LINK 1 |LINK 2| |LINK n |

It is often simpler to write algorithms for a data representation where the node size is fixed. Using data and pointer fields we can represent a tree using the fixed node size list structure. The data object tree is a special instance of the data object list and we can implement the list representation scheme to them. Figure 4.2 illustrates the list representation for the tree represented in figure 4.1.

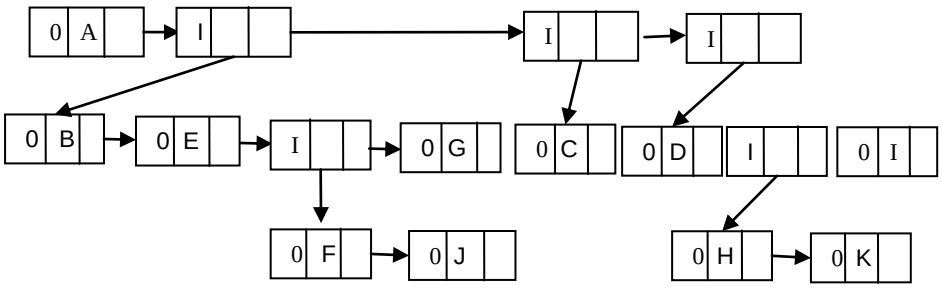

Figure 4.2: List representation of tree

4.3 Binary Tree: Definition and Concepts

A binary tree is an important type of tree structure which occurs very often. It is characterized by the fact that any node can have at most two branches, i.e., there is no node with degree greater than two. For binary trees we distinguish between the subtree on the left and on the right, whereas for trees the order of the subtrees was irrelevant. Also a binary tree may have zero nodes. Thus a binary tree is really a different object than a tree

Definition: A *binary tree* is a special case of tree where no node of a tree can have a degree of more than two. Therefore, a binary tree is a set of zero or more nodes T such that:

i) there is a specially designated node called the root of the tree

ii) the remaining nodes are partitioned into two disjointed sets, T_1 and T_2, each of which is a binary tree. T_1 is called the left subtree and T_2 is called right subtree.

A binary tree is shown in figure 4.3, here you can find for the root A, B is the left sub tree and C is the right sub tree.

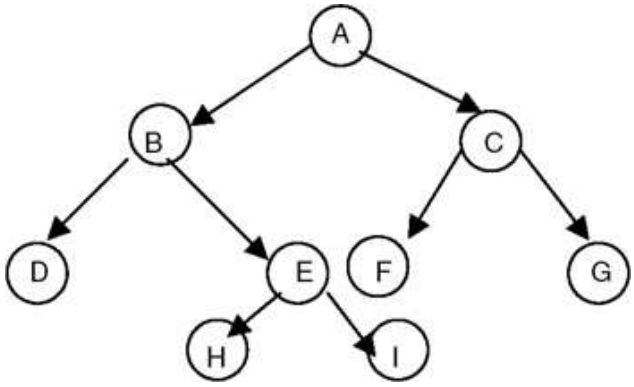

Figure 4.3: Binary tree structure

So, for a binary tree we find that:

i) The maximum number of nodes at level i will be 2^{i-1}

ii) If k is the depth of the tree then the maximum number of nodes that the tree can have is

$2^k - 1 = 2^{k-1} + 2^{k-2} + \ldots + 2^0$

Proof:

i) The proof is by induction on i.

Induction Base: The root is the only node on level *i* = 1. Hence the maximum number of nodes on level i = 1 is 2 = 2i - 1.

Induction Hypothesis: For all j, 1 j < i, the maximum number of nodes on level j is 2j - 1.

Induction Step: The maximum number of nodes on level *i* - 1 is 2*i* - 2 , by the induction hypothesis. Since each node in a binary tree has maximum degree 2, the maximum number of nodes on level *i* is 2 times the maximum number on level *i* - 1 or 2*i* -1.

(ii) The maximum number of nodes in a binary tree of depth k $\sum_{i=1}^{k}$ is (maximum number of nodes on level i*)* $\sum\sum$

$$\sum_{i=1}^{k} 2^{i-1} = 2^k - 1.$$

4.4 Types of Binary Tree

Now we are going to discuss few very important types of binary tree.

1) Skewed binary tree
2) Full binary tree
3) Complete binary tree

Skewed binary tree

A binary tree which has only left subtree is called left skewed tree and has only right subtree is called right skewed tree as shown in the figure 4.4. Skewed trees are not efficient in memory management because generally, an n node binary tree needs an array whose length is between n+1 and $2^{n,}$ but while storing skewed binary tree it wastes most of the memory space.

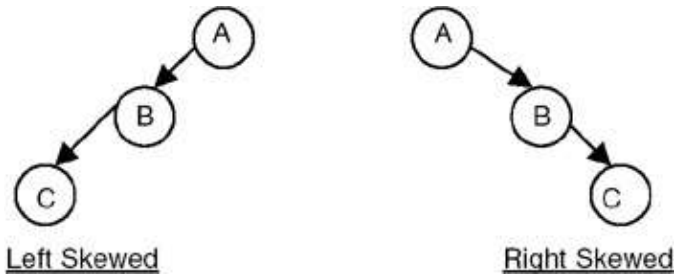

Figure 4.4: Skewed trees

Full binary tree

A binary tree is a full binary tree if and only if :-

- Each non leaf node has exactly two child nodes
- All leaf nodes are at the same level

A full binary tree is a binary of depth k having $2^k - 1$ nodes as referred in the figure 4.5. If it has $< 2^k - 1$, it is not a full binary tree. For example, for k = 3, the number of nodes = $2^k - 1 = 2^3 - 1 = 8 - 1 = 7$. A full binary tree with depth k = 3 is shown in Figure 4.5. We use numbers from 1 to $2^k - 1$ as labels of the nodes of the tree. If a binary tree is full, then we can number its nodes sequentially from 1 to 2^{k-1}, starting from the root node, and at every level numbering the nodes from left to right.

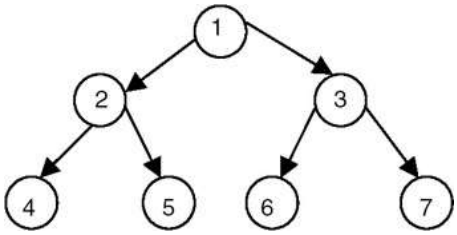

Figure 4.5: A full binary tree

Complete binary tree

A *complete binary tree* is a binary tree in which every level, *except possibly the last*, is completely filled, and all nodes are as far left as possible.

Figure 4.6 shows a complete binary tree filled at each depth from left to right.

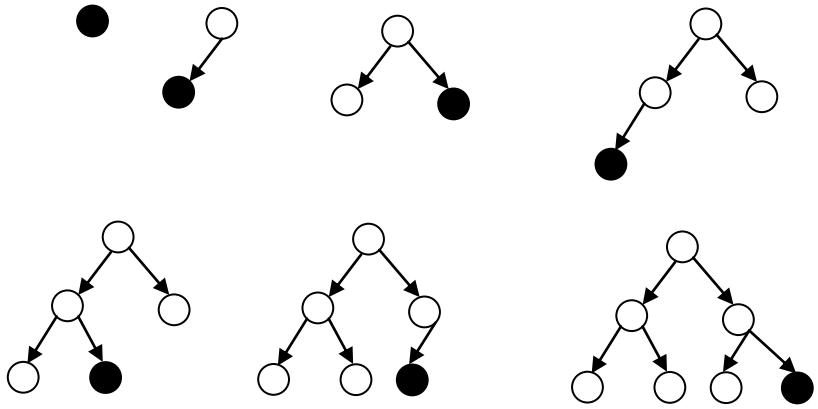

Figure 4.6: Complete binary tree from left to right

A complete binary tree of depth *k* is a tree with *n* nodes in which these *n* nodes can be numbered sequentially from 1 to *n*, as if it would have been the first *n* nodes in a full binary tree of depth *k*. A complete binary tree with depth *k* = 3 is shown in Figure 4.7.

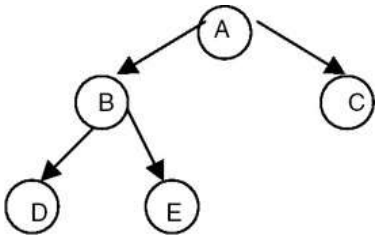

Figure 4.7: A complete binary tree

Self Assessment Questions

1. Tree is a non linear data structure state. True/False
2. List of nodes belongs to same parent is called _____.
3. Tree with only left subtree is called _____.

4.5 Traversal on Binary Tree

There are many operations that we often want to perform on trees. One notion that arises frequently is the idea of traversing a tree or visiting each node in the tree exactly once. A full traversal produces a linear order for the information in a tree. This linear order may be familiar and useful. When traversing a binary tree we want to treat each node and its subtrees in the same fashion. If we let *L, D, R* stand for moving left, printing the data, and moving right at a node then there are six possible combinations of traversal: *LDR, LRD, DLR, DRL, RDL,* and *RLD* If we adopt the convention that we traverse left before right then only three traversals remain: *LDR, LRD* and *DLR* . To these we assign the names inorder, postorder and preorder because there is a natural correspondence between these traversals and producing the infix, postfix and prefix forms of an expression. Consider the binary tree of figure 4.8. We will define three types of traversals and show the results for this tree.

Representation of binary tree node

Binary tree node can be represented as follows

```
struct tree_node
{
   tree_node *left;
   tree_node *right;
   int data;
};
```

Inorder traversal

Inorder traversal helps to print the binary search tree in sorted order. Here the traversal starts with left most subtree then prints the parent node and then proceeds with the right subtree.

1) Traverse the left subtree
2) Visit the root node
3) Traverse the right subtree

```
void print_inorder(tree_node *p)
 {
    if (p != NULL)
   {
        print_inorder(p->left);  // print left subtree
        cout << p->data << endl; // print this node
        print_inorder(p->right); // print right subtree
    }
}
```

Following algorithm explain each step of Inorder traversal in detail

Step 1: Start from the root node.

Step 2: Now go to the leftmost node from the root node in the left subtree.

Step 3: When the leftmost node is reached. Print it down.

Step 4: Now visit/ go the leftmost node's parent node. Print it down.

Step 5: Now visit/ go to the first node of the right subtree.

Step 6: Proceed in the same way as in Step 2 - 5 till 'printing' of all the nodes is finished

Figure 4.8 represents the inorder traversal of binary tree.

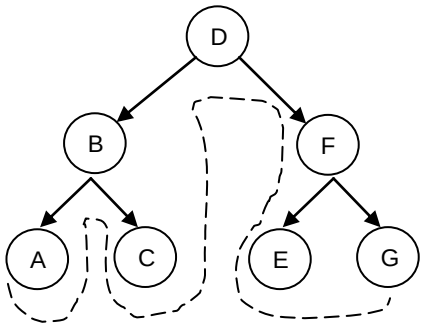

Figure 4.8 (i): Binary tree inorder visit

Nodes of tree visited in the inorder traversal of figure 4.8(i) would be: A B C D E F G

Postorder traversal

In postorder traversal, traversal starts with the left most subtree then proceeds with right sub tree and print the parent of those nodes.

1) Traverse the left subtree
2) Traverse the right subtree
3) Visit the root node

```
 void print_postorder(tree_node *p)
  {
     if (p != NULL)
    {
         print_postorder(p->left); //print left subtree
         print_postorder(p->right);// print right subtree
         cout << p->data << endl; // print this node
     }
}
```

Following algorithm explain each step of postorder traversal in detail

Step 1: Start from the root node .

Step 2: Now go to the leftmost node from the root node in the left subtree.

Step 3: When the leftmost node is reached . print it down.

Step 4: Now visit/ go the leftmost node's parent node.

Step 5: Now visit/ go to the first node of the right subtree.

Step 6: Proceed in the same way as in Step 2 - 5 till 'printing' of all the nodes is finished.

Figure 4.8(ii) represents the post order traversal of a binary tree.

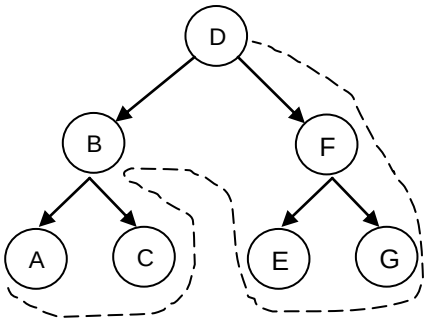

Figure 4.8 (ii): Binary tree postorder visit

Nodes of tree visited in the post order traversal of figure 4.8(ii) would be : A C B E G F D.

Preorder traversal

Here in preorder traversal, we start the traversal technique with the root node then proceeds towards the end of the left subtree and then towards

the right sub tree. Figure 4.8(iii) shows the traversal pattern using preorder traversal.

1) Start at the root node
2) Traverse the left subtree
3) Traverse the right subtree

```
void print_preorder(tree_node *p)
 {
   if (p != NULL)
  {
      cout << p->data << endl; // print this node
      print_preorder(p->left); //print left subtree
      print_preorder(p->right);// print right subtree
   }
```

Following algorithm explain each step of preorder traversal in detail.

Step 1: Start from the root node. print it down.
Step 2: Now go to the leftmost node from the root node in the left sub tree. On it's way to the left most node, print all the nodes as it comes.
Step 3: When the leftmost node is reached. print it down.
Step 4: Now visit the right sub tree of the leftmost node's parent node.
Step 5: Write down the first node of the right subtree.
Step 6: Now proceed in the same way as in Step 2 - 5 till 'printing' of all the nodes is finished.

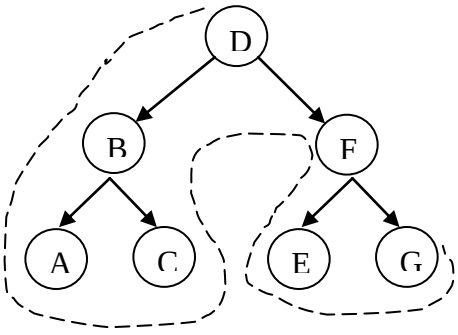

Figure 4.8 (iii): Binary tree preorder visit

Nodes of tree visited in the pre order traversal of figure 4.8(iii) would be: D B A C F E G.

The above discussed three traversal is called as ***depth first traversal*** because all these patterns takes the visiting pattern from top to bottom that is deep towards its left subtree and travelling across.

Breadth first traversal

Trees can also be traversed in ***level-order***, where we visit every node on a level before going to a lower level. This is also called ***breadth first traversal.*** Level order traversal will be using queue to print the traversal pattern. Initially queue will be empty and then the root node will be inserted and look for the subtrees. Then root will be removed from the queue and printed and the left and right nodes will be inserted into the queue this process proceeds until the tree reaches Null.

```
void levelOrder(ptnode ptr)
{
  int front = rear = 0;
  if (!ptr) return; /* empty queue */
  enqueue(front, &rear, ptr);
  for (;;) {ptr = dequeue(&front, rear);
   if (ptr) { printf("%d", ptr->data);
      if (ptr->left)
        enqueue(front, &rear, ptr->left);
      if (ptr->right)
        enqueue(front, &rear, ptr->right);
    }
    else break; } }
```

Step 1: Initialize queue with front and rear as 0

Step 2: Add root to queue

Step 3: Get node from the queue and print data, then add the left and right child to queue

Step 4: Go to step 3 until node is NULL

Step 5: If child node is NULL, skip and not to add to queue

Figure 4.8 (iv) represents the binary tree traversal in the level order pattern.

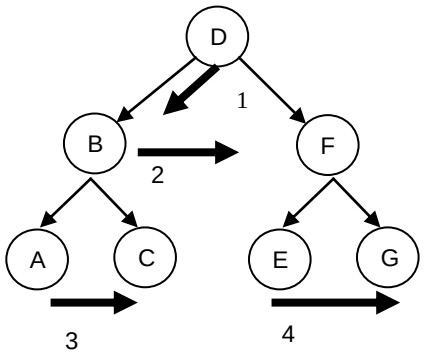

Figure 4.8 (iv): Binary tree level order visit

4.6 Representation of Binary Tree

- Array representation of binary tree
- Linked storage representation of binary tree.

4.6.1 Array representation of binary tree

If a binary tree is a *complete binary tree*, it can be represented using an array which is capable of holding *n* elements where n are the number of nodes in a complete binary tree. If the tree is an array of *n* elements, we can store the data values of the i^{th} node of a complete binary tree with *n* nodes at an index i in an array tree.

Following is the rule to decide the location of any node of a tree in the array.

1) The root node is at location 1
2) For any node with index i, $1< i \leq n$
   i) parent(i)=i/2
   ii) Left child (i) = 2*i
   iii) Right child (i) =2*i+1

Binary tree and array representation of binary tree is exhibited in the figure 4.9(i) and 4.9(ii) respectively.

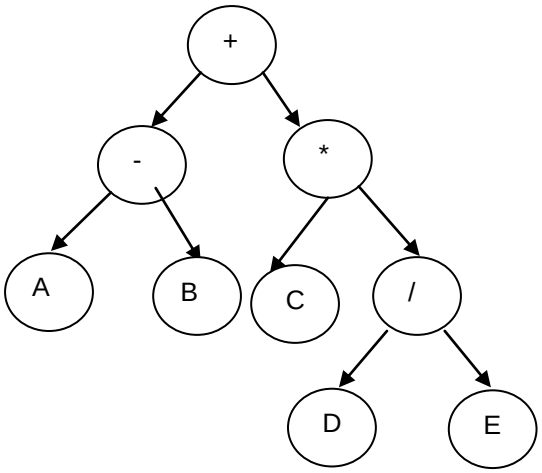

Figure 4.9 (i): Binary Tree

1	2	3	4	5	6	7	8	9	10	11	12	13	14
+	-	*	A	B	C	/						D	E

Figure 4.9 (ii): Sequential representation of Binary Tree

Advantages of linear representation of binary tree

1) Node can be accessed directly with the help of the index through which we can improve the efficiency of execution time of an algorithm.
2) Data are stored without pointer reference of successor or predecessor
3) This representation is very much useful where the language does not support the dynamic memory allocation.

Disadvantages of Linear representation of binary tree

1) Memory is wasted here because it will be allocated for all the nodes absence of node will leads to empty entries in array.
2) Since the array size is limited enhancement of tree structure is restricted.
3) Inefficiency in processing time during insertion or deletion of nodes because considerable movement of up and down is taking place in array structure.

4.6.2 Linked storage representation of binary tree

The most popular way to present a binary tree is Linked storage representation. Binary tree consists of nodes which can have at most two child, linked representation of such tree will be stored in the form shown Fig 4.10, where LPTR and RPTR denote the address or locations of the left and right subtrees respectively of a particular root node. Empty subtrees are represented with NULL. DATA specifies the information associated with the node. The space required by an *n* node binary tree is *n* * size of (binaryTreeNode)

Figure 4.10

Figure 4.11 contains an example of linked storage representation of binary tree shown in Fig 4.9(i). Linked representation of binary tree is considered as dynamic in nature as the block of memory required for the tree can be allocated on demand.

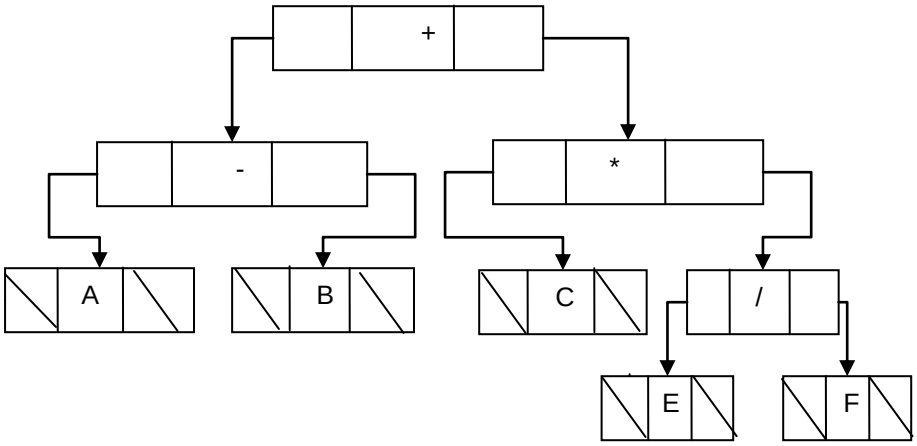

Figure 4.11: Linked representation of binary tree

Advantages of linked representation of binary tree

1) Utilization of memory is efficient.
2) Enhancement of tree is possible in this representation.
3) Insertion and deletion of nodes can be easily handled with the pointers without data movement.

Disadvantages of linked representation of tree

1) It is difficult to implement with the language which does not support the dynamic allocation of memory.
2) Since pointer is involved for data reference it occupies more memory.

Self Assessment Questions

4. _____ order traversal follow pattern as root, left, right.
5. In an array representation of binary tree the right child of root will be at location at _____ .
6. In a complete binary tree right child of a node can be identified by _____.
7. Breadth first traversal otherwise called as _____.
8. Linked representation of tree facilitates the efficiency of memory. True/False.

4.7 Conversion of General Tree to Binary Tree

In the section 4.2, we discussed about the general and some concepts of binary tree, so we know the various advantages of binary tree over general tree. General tree can be converted into an equivalent binary tree, but this process makes the tree imbalance. Here, each node requires only two references, but, these are not designated as left or right. Instead they are designated as the reference to the first child and the reference to next sibling. Therefore the usual left pointer really points to the first child of the node and the usual right pointer points to the next sibling of the node. In this way, moving right from a node accesses the siblings of the node (that is all of those nodes on the same level as the node in the general tree). Moving left and then right accesses all of the children of the node (that is the nodes on the next level of the general tree).

The process of converting the general tree to a binary tree is as follows:

- The root of the general tree must be the root of the binary tree.
- Determine the first child of the root which is the leftmost node in the general tree at the next level.
- insert this node. The child – parent relationship in general tree would be considered in binary tree also.
- continue finding the first child of each parent node and insert it below the parent node.

- when no more first children exist in the path just used, move back to the parent of the last node entered and determine the next sibling, then insert right to the previous left sibling (where sibling in general tree would be child in binary tree).
- In order to complete the tree, the following steps to be executed:
  - Look for the completion of siblings at one generation level.
  - Next move to left most child of the successor generation and repeat the process as same until the siblings of the generation are inserted.

If you apply the above procedure sequentially on the tree referred in the figure 4.12(i) you can construct a binary tree as shown in the figure 4.12(ii).

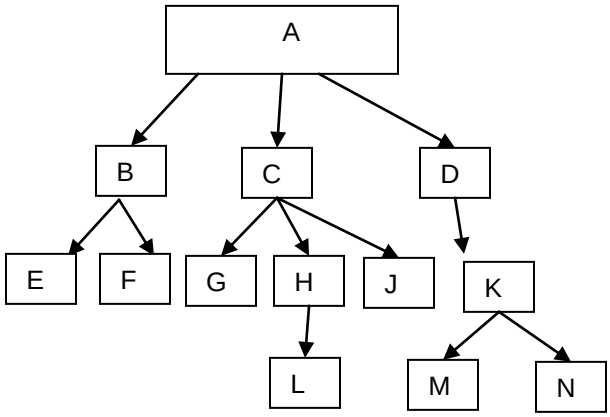

Figure 4.12 (i): General Tree

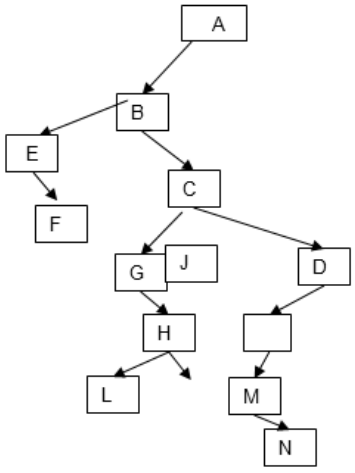

Figure 4.12 (ii): Binary Tree

4.8 Sequential and Other Representations of Binary Tree

Here we are going to discuss the various representations of trees based on sequential allocation technique. During the transaction of deletion or insertion, on the sequential representation the structure does not change much which makes this representation is considered as very efficient and effective for processing.

4.8.1 Sequential representation of complete binary tree

One of the subclass of binary tree is complete binary tree. A complete binary tree is a binary tree in which there is one node at the root level, two nodes at level 2, four nodes at level 3 and so on. An example for complete binary tree along with its sequential representation is shown in the figure 4.13 (i) and 4.13(ii) respectively.

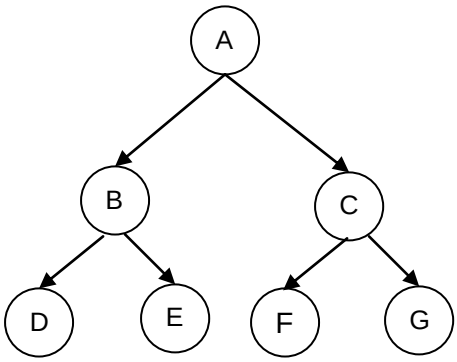

Fig. 4.13 (i): Complete Binary tree

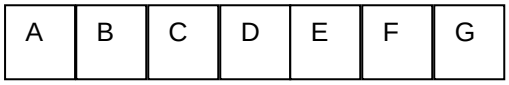

Figure 4.13 (ii): Sequential representation of tree

In this representation the locations of the left and right children of node i are 2i and 2i+1, respectively. For example in fig 4.13 (i) for the node in position 3 (that is c) having its left child at the index location 6. Similarly the index of the right child is 7. On the other hand, the position of the parent node of j will be in the index TRUNC (j/2), for example in fig 4.13 (i) the parent of nodes 6 and 7 is in node 3.

4.8.2 Sequential representation of incomplete binary tree

Binary trees which have more or less than 2^n-1 nodes called incomplete binary tree can also be represented using previous approach, but for incomplete binary trees extra storage is required and overhead of NULL node checking takes place. Incomplete binary tree and its sequential representation is referred in the figure 4.14(i) and 4.14(ii) respectively.

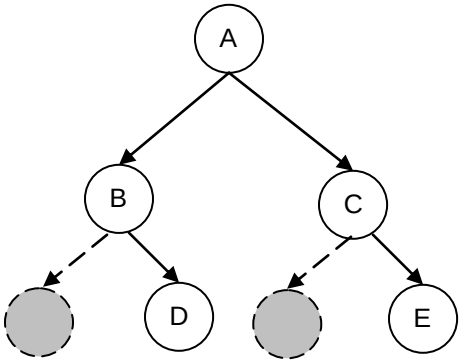

Figure 4.14 (i): Incomplete binary tree

Figure 4.14 (ii): Sequential representation of incomplete tree

4.8.3 Preorder sequential representation of binary tree

General method for sequential representation of binary tress uses the physical adjacency relationship of the computer memory to replace one of the link fields in the linked representation method. Consider an alternative representation of a tree structure, in which the left link (LPTR) from the usual doubly linked list representation has been omitted. We can represent this tree sequentially if its nodes appear in preorder, using this approach you can find a tree and its preorder representation in figure 4.15(i) and 4.15(ii) respectively. Here we are using only RPTR, INFO, and TAG and not the LPTR, since for a non-null link it would point to the node to its immediate right. The bit TAG denotes with logical value of 1, a leaf node.

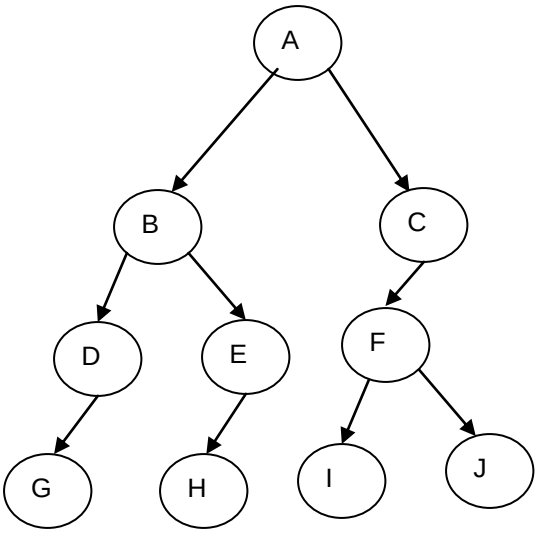

Figure 4.15 (i): Binary tree

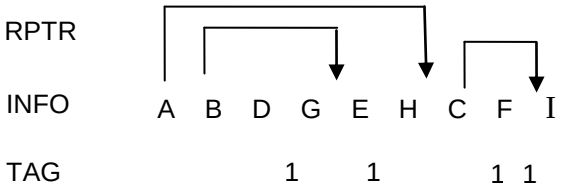

Figure 4.15 (ii): Preorder sequential representation of a binary tree

4.8.4 Postorder sequential representation of binary tree

Now we are going to discuss one more popular method to represent a tree sequentially using postorder traversal. Here it avails two vectors for representation one to represent the nodes called POST and the second vector denotes the number of children of the nodes called DEGREE as referred in the figure 4.16(ii) for the tree depicted in figure4.16(i). This postorder representation of a tree is useful in evaluating functions that are defined on certain nodes of the tree.

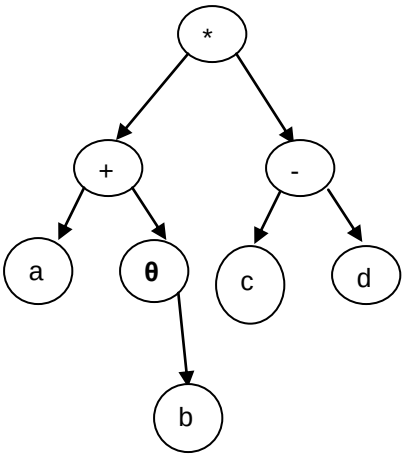

Figure 4.16(i): Binary tree

POST	a	b	θ	+	c	d	-	*
DEGREE	0	0	1	2	0	0	2	2

Figure 4.16(ii): Postorder representation of binary tree

Self Assessment Questions

9. Conversion of tree to binary tree makes the tree _____.
10. Incomplete tree is very efficient in memory management. (True/False)
11. _____ is used to denote the leaf node representation as 1 in preorder sequential representation.
12. _____ and _____ are the vectors used for postorder sequential representation.

4.9 Summary

Information and discussion of trees structure may seem complex in the beginning, but they are fairly powerful data structures. We have discussed about two representations for trees – '' the array'', and '' the linked structure'' and discussed about some operation we could do on trees. The concept of a binary search tree was introduced as an alternative to store data in an array or a linked list. We also discussed about the various types of binary trees, because each type of binary tree having its own character helpful for different situation. Traversal is one of the important operations that we can do on binary tree, because while inserting, deleting or searching an element,

it is mandatory to travel in the tree. Hope this data structure was interesting and useful in its application.

4.10 Terminal Questions

1. Define tree. Explain its terminology with proper example.
2. What is binary tree? Explain with example.
3. List and explain the types of binary tree.
4. Explain the different types of traversal on binary tree.
5. Mention the types of representation of binary tree in memory
6. How will you construct a binary from the given tree?
7. Explain various sequential representation of binary tree

4.11 Answers

Self Assessment Questions

1. True
2. Siblings
3. Left skewed tree
4. Preorder
5. 3
6. 2*i+1
7. Level order traversal
8. True.
9. Improper
10. False
11. TAG
12. POST and DEGREE

Terminal Questions

1. Tree is one of the non linear data structure with root and many other nodes to store data. (Refer section 4.2 for detail)
2. Binary tree is a variant of tree with restricted two nodes called left and right. (Refer section 4.3)
3. Binary tree can be represented in different types for conveyance. (Refer section 4.4)
4. Tree traversal is a technique used to visit nodes in a systematic way. (Refer section 4.5)

5. Representation of binary tree in memory with array and linked list. (Refer section 4.6)
6. General tree can be converted into an equivalent binary tree, but this process makes the tree. (Refer section 4.7 for detail)
7. Sequential representation is very efficient during the process of insertion or deletion. (Refer section 4.8 for detail)