## 6-TREE

# Tree:

* A tree is defined as a finite set of one or more nodes such that
  + There is a special node called the root node R.
  + The remaining nodes are divided into n ≥ 0 disjoint sets T1, T2,.,.,. TN, where each of these sets are tree. T1, T2…., T n is called the sub tree of the root



**Forest:** A forest is a set of n ≥ 0 disjoint trees.

**Adjacent node:** Any two nodes in the graph which are connected by an edge in the graph are called adjacent nodes.

**Root Node:** In a directed tree a node which has in degree 0 is called the root node.

**Leaf Node:** In a directed tree a node which has out degree 0 is called the leaf node or terminal node.

**Directed Tree:** A directed tree is an acyclic digraph which has one node called the root node with in degree 0 while all other nodes have in degree 1.

* Figure-1 is directed tree.

**m-ary Tree**: If in a directed tree the out degree of every node is less than or equal to m then the tree is called the m – ary tree.

**Complete m - ary Tree:** if the out degree of every node is exactly equal to m or 0 and the number of nodes at level i is mi-1 then the tree is called full or complete m – ary tree.

**Binary Tree:** If in a directed tree the out degree of every node is less than or equal to 2 then the tree is called the binary tree.

**Complete Binary Tree:** if the out degree of every node is exactly equal to 2 or 0 and the number of nodes at level i is 2i-1 then the tree is called full or complete binary tree.

* **Example:**



**In above figure tree has a 23-1 =4 at level 3.**

**Ordered Tree:** If in a directed tree an ordering of the node at each level is prescribed then such a tree is called an ordered tree.

**Sibling node:** The child of a same parent is known as sibling node.

**Definitions related with tree**

**In degree:** Number of edges terminated at given node is called in degree of node.

**Out degree:** Number of edges emerging from given node is called out degree of node.

**Edge:** Two nodes are connected by edge. It is the connection of two nodes.

**Level:** In a tree the distance between two nodes are represent as level.

**Depth:** Maximum level number in a tree is called the depth of the tree.

**Height:** Total number of level in a tree is called the height of the tree.

**Weight:** Total number of leaf nodes in a tree is called the weight of the tree.

**Strictly Binary Tree:** If every non leaf node in a binary tree has nonempty left and right sub tree is called strictly binary tree.



**Difference between root node and leaf node.**

* Root node has the in degree 0 while the leaf node has the out degree 0

Root Node with in degree 0

Leaf Node Leaf Node with out degree 0

**Comparison of Tree and Linked list**

|  |  |
| --- | --- |
| **Linked List** | **Tree** |
| **(1)** Linked list is an example of linier data structure. | **(1)** Tree is an example of non linier data structure. |
| **(2)** Searching time is more required in linked list. Because we must have to traverse each node sequentially in linked list even if the list is sorted. | **(2)** Searching time is less required in tree because we can search an element using binary search method. |

**Operations on Binary Search Tree:**

* Following operations can be performed on Binary Search tree:
  1. Traversal
  2. Insertion
  3. Deletion
  4. Search
  5. Copy

**Traversal operation of Binary Search Tree**

**Traversal:**

* Traversal is the method of processing every nodes in the tree exactly once in a systematic manner.

**Types of Traversal:** There are three different types of tree traversal.

1. Preorder Traversal
2. In order Traversal
3. Post order Traversal

**(1) Preorder Traversal:**

The preorder traversal follows three steps:

1. Process the root node first.(***V***ertices or Node)
2. Traverse the left sub tree in preorder.(***L***eft node)
3. Traverse the right sub tree in preorder.(***R***ight node)

**Example:**



**Preorder Traversal: A B D E C F G**

**(2) In order Traversal:**

The In order traversal follows the three steps:

1. Traverse the left sub tree in inorder(***L***eft node).
2. Process the root node. (***V***ertices or Node)
3. Traverse the right sub tree in inorder.(***R***ight node)

**Example:**



**InOrder Traversal: D B E A F C G**

**(3) Post order Traversal:**

* The Post order traversal follows the three steps:
  1. Traverse the left sub tree in Post order. (***L***eft node)
  2. Traverse the right sub tree in Post order. (***R***ight node)
  3. Process the root node. (***V***ertices or Node)

**Example:**



**Post Order Traversal: D E B F G C A**

**Examples of tree traversal**

**Ex-3: Travers the following Tree in Pre Order, In Order, Post Order.**



**Pre Order: 42 12 7 1 9 18 55 48 90 75 110**

**In Order: 1 7 9 12 18 42 48 55 75 90 110**

**Post Order: 1 9 7 18 12 48 75 110 90 55 42**

**Algorithms for Binary Tree Traversal**

**(1) Pre Order Traversal:**

**PREORDER(T)**

* This function traverses the tree in pre order.
* T is a pointer which points to the root node of tree.

**1. [Process the root node]**

If T ≠ NULL then

Write (DATA (T))

Else

Write “Empty Tree”

**2. [Process the left sub tree]**

If LPTR (T) ≠ NULL then

Call PREORDER (LPTR (T))

**3. [Process the right sub tree]**

If RPTR (T) ≠ NULL then

Call PREORDER (RPTR (T))

**4. [Finished]**

Return

**(2) In Order Traversal:**

**INORDER(T)**

* This function traverses the tree in inorder.
* T is a pointer which points to the root node of tree.

**1. [Check for empty tree]**

If T = NULL then

Write “Empty Tree”

**2. [Process the left sub tree]**

If LPTR (T) ≠ NULL then

Call INORDER (LPTR (T))

**3. [Process the root node]**

Write (DATA (T))

**4. [Process the right sub tree]**

If RPTR (T) ≠ NULL then

Call INORDER (RPTR (T))

**5. [Finished]**

Return

**(3) Post Order Traversal:**

**POSTORDER(T)**

* This function traverses the tree in post order.
* T is a pointer which points to the root node of tree.

**1.** **[Check for empty tree]**

If T = NULL then

Write “Empty Tree”

**2.** **[Process the left sub tree]**

If LPTR (T) ≠ NULL then

Call POSTORDER (LPTR (T))

**3.** **[Process the right sub tree]**

If RPTR (T) ≠ NULL then

Call POSTORDER (RPTR (T))

**4.** **[Process the root node]**

Write (DATA (T))

**5. [Finished]**

Return

**Insertion Operation on Binary Search tree**

**Insertion Operation**

* Binary search tree has following characteristics:

(1) All the nodes to the left of root node have value less than the value of root node.

(2) All the nodes to the right of root node have value greater than the value of root node.

* Suppose we want to construct binary search tree for following set of data:

**45 68 35 42 15 64 78**

* **Step 1:** First element is 45 so it is inserted as a root node of the tree.

Root Node

* **Step 2:** Now we have to insert 68. First we compare 68 with the root node which is 45. Since the value of 68 is greater then 45 so it is inserted to the right of the root node.
* **Step 3:** Now we have to insert 35. First we compare 35 with the root node which is 45. Since the value of 35 is less then 45 so it is inserted to the left of the root node.
* **Step 4:** Now we have to insert 42. First we compare 42 with the root node which is 45. Since the value of 42 is less than 45 so it is inserted to the left of the root node. But the root node has already one left node 35. So now we compare 42 with 35. Since the value of 42 is greater than 35 we insert 42 to the right of node 35.
* **Step 5:** Now we have to insert 15. First we compare 15 with the root node which is 45. Since the value of 15 is less than 45 so it is inserted to the left of the root node. But the root node has already one left node 35. So now we compare 15 with 35. Since the value of 15 is less than 35 we insert 15 to the left of node 35.
* **Step 6:** Now we have to insert 64. First we compare 64 with the root node which is 45. Since the value of 64 is greater than 45 so it is inserted to the right of the root node. But the root node has already one right node 68. So now we compare 64 with 68. Since the value of 64 is less than 68 we insert 64 to the left of node 68.
* **Step 7:** Now we have to insert 78. First we compare 78 with the root node which is 45. Since the value of 78 is greater than 45 so it is inserted to the right of the root node. But the root node has already one right node 68. So now we compare 78 with 68. Since the value of 78 is greater than 68 we insert 78 to the right of node 68.

**Algorithm for Insertion Operation on Binary Search tree**

**INSERT(Root, Info)**

* Root is a pointer which points to the root node of tree.
* Info is an element which we want to insert
  1. **[Check for empty tree]**

If Root = NULL then

DATA(Root)🡨Info

LPTR(Root)🡨NULL

RPTR(Root)🡨NULL

* 1. **[Initialize]**

CURRENT🡨 Root

* 1. **[left sub tree]**

If Info < DATA(CURRENT) then

If LPTR(CURRENT) ≠ NULL then

CURRENT🡨LPTR(CURRENT)

Else

DATA(Root )🡨Info

LPTR(Root) = NULL

RPTR(Root) = NULL

* 1. **[right sub tree]**

If Info > DATA(CURRENT) then

If RPTR(CURRENT) ≠ NULL then

CURRENT🡨RPTR(CURRENT)

Else

DATA(Root)🡨 Info

LPTR(Root)🡨 NULL

RPTR(Root)🡨 NULL

* 1. **[Check for duplicate node]**

If DATA(CURRENT) = DATA(Root) then

Write “Item already in tree”

* 1. **[finished]**

Return

**Deletion Operation on Binary Search tree**

**Delete Operation**

* Consider the following binary tree:



* There are three possibilities when we want to delete an element from binary search tree.

(1) If a node to be deleted has empty left and right sub tree then a node is deleted directly.

Suppose we want to delete node 9. Here node 9 has no left or right sub tree so we can delete it directly. Thus tree after deleting node 9 is as follow:



(2) If a node to be deleted has only one left sub tree or right sub tree then the sub tree of the deleted node is linked directly with the parent of the deleted node.

Suppose we want to delete node 7. Here node 7 has one left sub tree so we link this left sub tree with parent of node 7 which is 12. Now the node which we want to link with node 12 is 1. Thus tree after deleting node 48 is as follow:



(3) If a node to be deleted has left and right sub tree then we have to do following steps:

(a) Find next maximum (in order successor) of the deleted node.

(b) Append the right sub tree of the in order successor to its grandparent.

(c) Replace the node to be deleted with its next maximum (in order successor).

Suppose we want to delete node 55. Here node 55 has both left and right sub tree. So first we have to find next maximum (in order successor) of node 55 which is 75.now replace the node to be deleted with its next maximum (in order successor). So we replace node 55 with node 75. Thus tree after deleting node 55 is as follow:



**Search Operation on Binary Search tree**

**Search Operation**

* First the key to be searched is compared with root node. If it is not in the root node then we have to possibilities :

(1) If the key to be searched having value less than root node then we search the key in left sub tree.

(2) If the key to be searched having value greater than root node then we search the key in right sub tree.

**Representation of binary tree**

* We can represent binary tree in memory by two methods:

(1) Array implementation

(2) Linked implementation

**Array implementation of Binary tree**

* While implementing binary tree we have to consider two cases:

**(1) Array implementation of complete binary tree:**

* A complete binary tree is a tree in which there is one node at the root level, two nodes at level 2, four nodes at level 3 etc…
* Example of such a tree and its array implementation is shown below:



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Index** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **Info** | **42** | **12** | **1** | **18** | **75** | **48** | **90** |

* In this implementation to find the location of left child of node I is given by **2\*I** and the location of right child of node I is given by **2\*I + 1**.
* For example: consider the node 2 in above example which is 12. The location of left child is given by 2\*I = 2\*2 = 4 which is 1. location of right child is given by 2\*I + 1 = 2\*2 + 1 = 5 which is 18.
* Similarly the location of the parent of left node is given by **I/2** & right node is given by **(I-1)/2**. For example location of parent of node 4 and 5 is 2.
* Thus to represent complete binary tree with 2n-1 node we require 2n-1 arrayelements.

**(2) Array implementation of incomplete binary tree:**

* Now consider the tree which has number of nodes less than or more than 2n-1
* Array implementation of such a tree is shown below:



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Index** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| **Info** | **42** | **12** | **75** | **1** | **18** | **48** | **90** | **-** | **-** | **10** | **-** | **-** | **-** | **81** | **100** |

* In this type of implementation the location of left children of node I is given by **2\*I** and the location of right children of node I is given by **2\*I + 1**.
* Thus location of left children and right children of node 3 is given by 2\*I = 2\*3 = 6 and 2\*I + 1 = 2\*3 + 1 = 7. But node 3 has no left or right node so this memory location becomes wastage in array implementation.
* Thus this method is not efficient for incomplete binary tree.

**Linked implementation of Binary tree**

* In binary tree there is a node called root node. This root node again has two parts left sub tree and right sub tree.
* Thus a node in binary tree is represented as below:

|  |  |  |
| --- | --- | --- |
| **LPTR** | **INFO** | **RPTR** |

* Each node have three parts:

(1) Info: information of node

(2) LPTR: pointer which points to left child of the node.

(3) RPTR: pointer which points to right child of the node.

* Ifa node which has no left or right child then its LPTR and RPTR fields are represented with NULL.
* **Example:**



* The above tree can be represented in linked implementation as below:



**Applications of Tree**

* Following are the application of Tree:
  + - 1. **Manipulation of arithmetic expressions**

Binary tree is used to represent formulas in prefix or postfix notation.

**For Example: Infix- A+B**

So in above example operator(+) becomes root node and operand(A and B) becomes left and right child node.

* + - 1. **Construction and maintenance of symbol table**

As an application of binary tree ,we will formulate the algorithms that will maintain a stack implemented tree structured symbol table.

So here binary tree structure is chosen for two reasons. The first reason is because if the symbol entries as encountered are uniformly distributed according to lexicographic order, then table searching becomes approximately equivalent to binary search, as long as the tree is maintained in lexicographic order. second a binary tree is easily maintained in lexicographic order in the sense that only a few pointers need be changed.

**(3)** **Syntax Analysis:** It deals with the use of grammars in syntax analysis or parsing.

**IMPORTANT QUESTIONS(Asked questions in the GTU Exam)**

1. Define following terms with necessary figure/example: **Tree, Leaf node (Terminal node), Degree of a node, Binary tree, M-array tree, complete binary tree, isolated node, weighted graph, Directed graph, root node, sibling, level.**
2. Explain all(Inorder,Preorder,Postorder) binary tree traversal method.
3. Explain List representation of Binary Tree.
4. List out operation on binary tree. Explain any one operation in detail
5. Write an algorithm for PREORDER and INORDER traversal.
6. Write short-note on: Application of trees.
7. List various tree traversal methods and explain. Construct a binary tree for the following data: 50, 55, 35, 15, 52, 65, 33, 47, 75, 72 Take 50 as root node.

Reconstruct the tree after addition of the nodes 27 and 56 and deletion of 55.

1. Write an algorithm to add a new node into a binary tree. Assume suitable data.

* **Inspirational Quotes**

**There are two things to aim at in life; first to get what you want, and after that to enjoy it. Only the wisest of mankind has achieved the second.**