DSA
INTERNSHALA
Trees

Basics of Trees

Tree is a non-linear data structure. Where data is stored in nodes arranged in a hierarchical manner with each node having child nodes and so on.

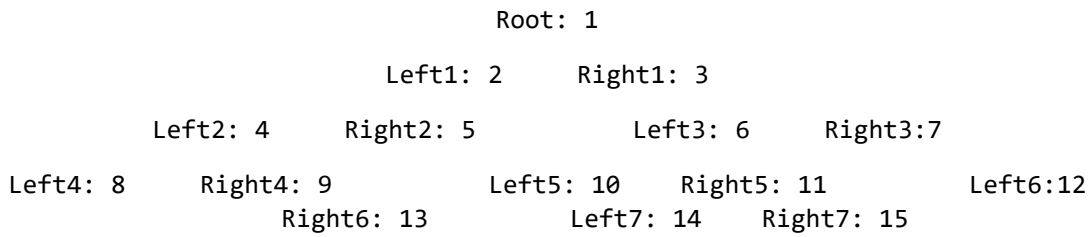
The nodes which have no child nodes are called terminal nodes or leaf nodes.

A tree is said to be a Binary tree if each node has at maximum two child nodes.

Representation of binary trees in memory

Array representation

- Root node is the very first element
- N is the location of the current node
- Left child node at 2*N
- Right child node at (2*N) + 1



Linked List Representation

Make nodes with three variables:

- One to store the value
- Two pointers:
    - leftChild

- rightChild

Height/Depth of a Binary Tree

- Longest path from the root node to any terminal node (leaf)
- (level number of that leaf)+1 = height of the tree

Level numbering starts from 0, i.e, level number for root node is 0.

Pre-order Traversal

Sequence:

- Root node
- Left subtree
- Right subtree

In-order Traversal

Sequence:

- Left subtree
- Root node
- Right subtree

Post-order Traversal

Sequence:

- Left subtree
- Right subtree
- Root node

Algorithm of Pre-order Traversal

[Algo of pre-order traversal]

1. Set top:=1, stack[top]:=NULL, temp:=root
2. Repeat steps 3,4,5 while temp!=NULL
3. Write: data[temp]
4. If right[temp]!=NULL, then:
   Set top:=top+1

```
   Set stack[top]:=right[temp]
   [end of if]
5. If left[temp] != NULL, then
   Set temp:=left[temp]
   Else
   Set temp:=stack[top]
   Set top:=top-1
   [end of if else]
6. Return
```

Algorithm of In-order Traversal

```
[Algo of in-order traversal]

1. Set top:=1, stack[top]:=NULL, temp:=root
2. Repeat steps while temp!=NULL
   (a)  Set top:=top+1
        Stack[top]:=temp
   (b)  Set temp:=left[temp]
3. Set temp:=stack[top] and top:=top-1
4. Repeat 5 to 7 while temp!=NULL
5. Write:data[temp]
6. If right[temp]!=NULL, then:
   (a)  Set temp:=right[temp]
   (b)  Goto step 2
7. Set temp:=stack[top] and top:=top-1
8. Return
```

Algorithm of Post-order Traversal

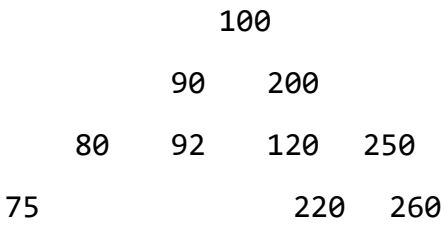
```
[Algo of post-order traversal]

1. Set top:=1, stack[1]:=NULL and ptr:=root
2. Repeat steps 3 to 5 while ptr!=NULL
3. Set top:=top+1 and stack[top]:=ptr
4. If right[ptr]!=NULL, then:
   Set top:=top+1 and
   Stack[top]:=-right[ptr]
5. Set ptr:=left[ptr]
6. Set ptr:=stack[top] and top:=top-1
7. Repeat while ptr>0
   (a)  Write:data[ptr]
   (b)  Set ptr:=stack[top] and top:=top-1
8. If ptr<0, then:
```

(a) Set ptr:=-ptr
(b) Goto step 2

9. return

Binary Search Tree

For any node, all the keys of it's right subtree nodes are bigger and all the keys of it's left subtree nodes are smaller.



If we traverse a BST using in-order traversal method, we get numbers in ascending order.

Heap

A binary tree with the following property:

- any node N will have a key value greater than or equal to all it's descendants.
- It means the root node will always be the biggest node.
- It is called a maxheap.
- Another type of heap is minheap.
- In minheap any node N will have a key value less than or equal to all it's descendants.

Inserting a node in Heap

- Insert the node at last index+1 location in array
- Very likely it will not be at its right position so we do something called rising which is to shift the node to higher levels until it reaches an appropriate position in the heap.

Deleting a root of Heap

- Assign the root value to a temporary variable
- Then the last value of the heap is assigned to the root position.

- This new value will not be at its likely position.
- We will follow sinking and shift it to lower levels until it reaches into its appropriate position.
- While sinking we sink it on the larger child node side.

Height Balanced Tree

Balance Factor = height of its left subtree – height of its right subtree

A height balanced tree has the following property:

- Each node will have a balance factor (BF) of either 0, 1 or -1.