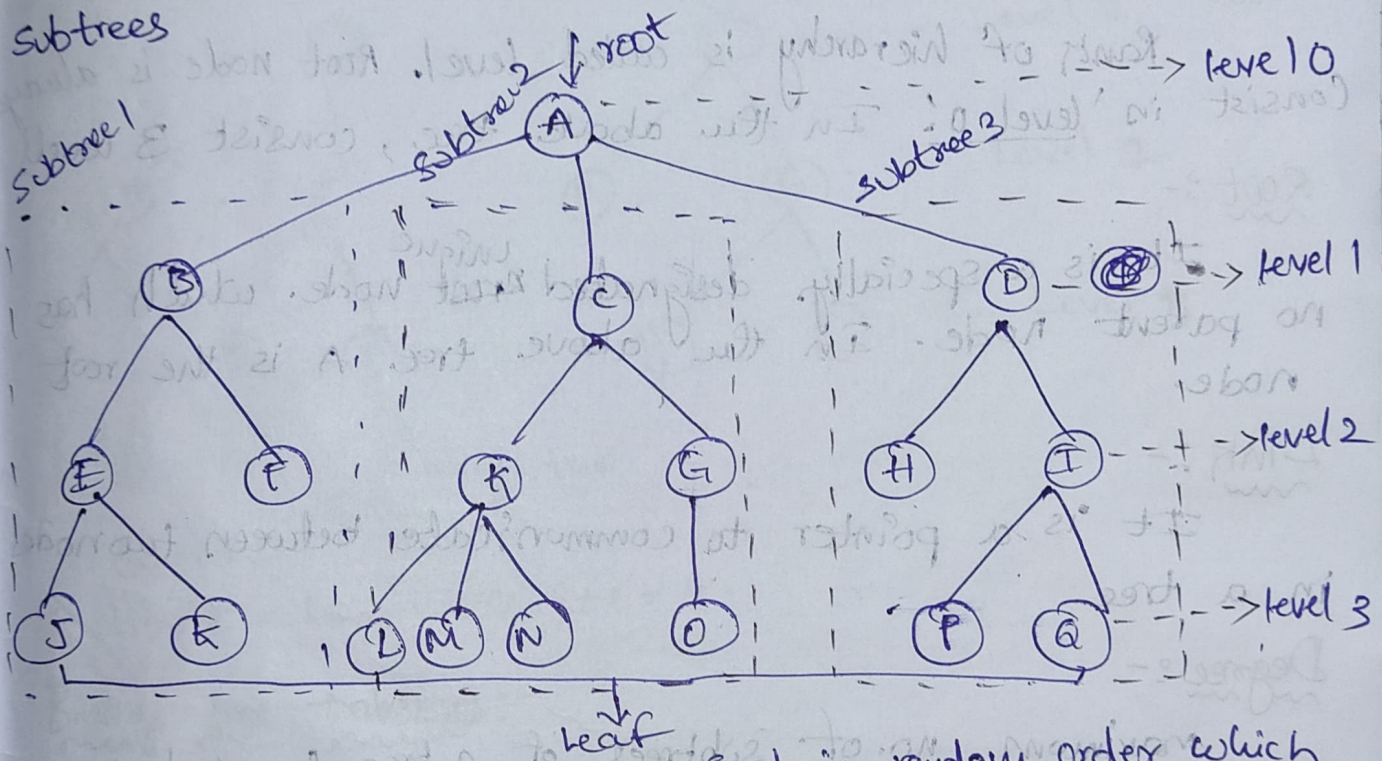
Date 26/9/24

Unit-5

Trees:-

Trees:-

Tree is a non-linear data structure, and it is a collection of elements or finite set of elements called nodes. It has specially designated unique node called root nodes and remaining nodes are partition disjoint sets to the root node called subtrees

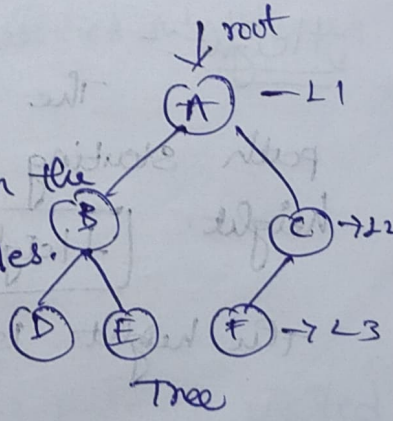


In tree data is organised in random order which is organised in hierarichal structure

Terminology:-

1. Node:-

Node of a tree is stored data. In the example A, B, C, D, E, F are called nodes.



2. Parent:-

Parent node is immediate preceeded of child node is called parent node. In the above tree A, B, C are called parent nodes.

Child :-

Child node is immediate succeeded of parent node. In the above tree B, C, D, E, F are child nodes.

Leaf :-

End of tree nodes is called Leaf node, but which have no child nodes. In the above tree D, E, F are called leaf nodes.

Level :-

Rank of hierarchy is called level. Root node is always consist in 'level 0'. In the above tree, consist 3 levels

Root :-

It is a specially designated unique root node. which has no parent node. In the above tree A is the root node.

Link :-

It is a pointer to communicate between two nodes in a tree.

Degree :-

maximum no. of subtrees of a tree connected to node is called degree. In the above diagram the degree of a tree = 2.

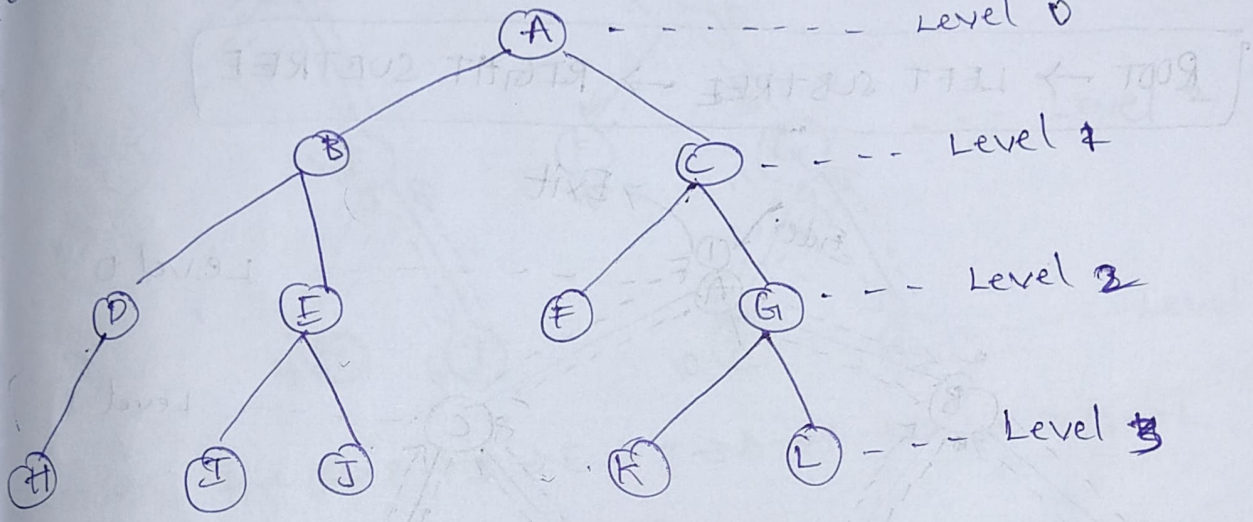
Height :-

The maximum no. of nodes that is possible in a path starting from root node to leaf node is called height. $\boxed{\text{Height} = L_{max} + 1}$. In the above example the height is $= 2 + 1 = 3$

Binary trees :-

Binary tree is a tree and it has zero or one or two child nodes. It also belongs to non linear data structure. Thus an order of binary tree is 2.

A Binary tree is either empty or consist of a one child node or two child nodes.



Binary tree

Order = 2

Height = $l_{max} + 1 \Rightarrow 3 + 1 = 4$.

Binary tree traversals :-

Binary tree is a finite set of elements called nodes. in which each node consist of two child nodes (left, right) or zero(0) or one(1). The Binary tree traversals is classified into 3 ways

1. Pre-order.
2. In-order.
3. Post-order.

(1.) Pre-Order :-

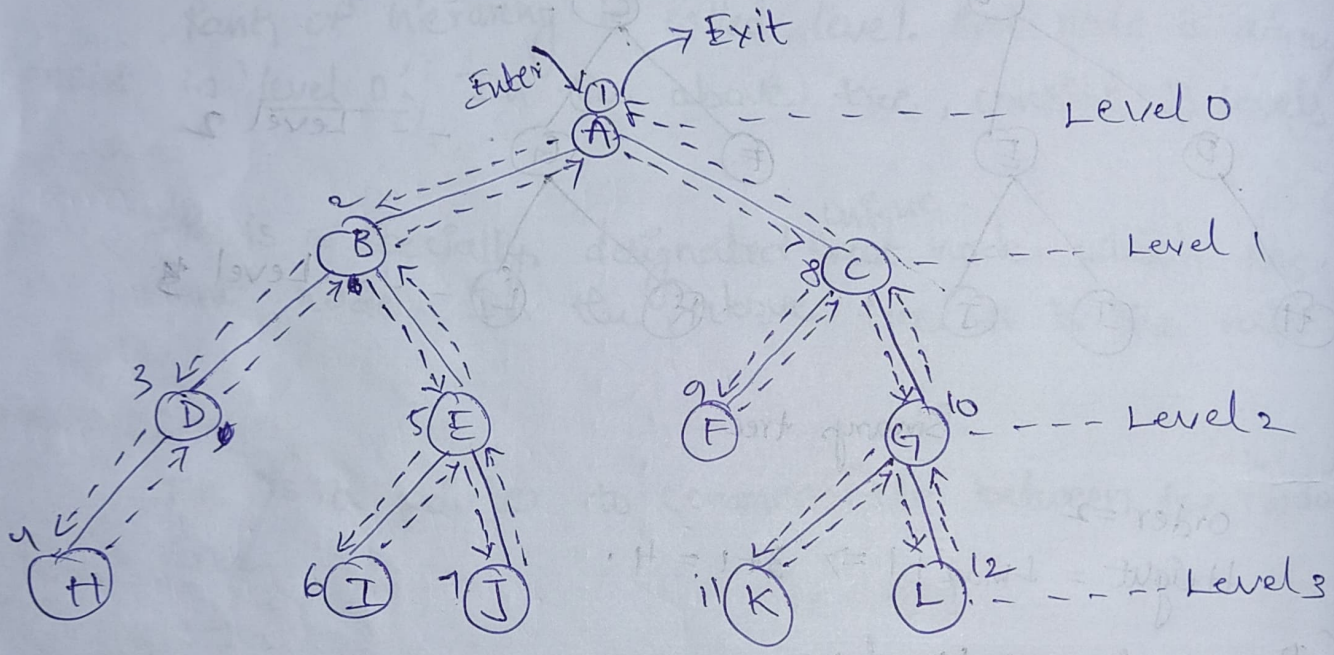
In a pre order traversal, Each root node is visited before its left and right subtrees are traversed. The pre-order search is also called as "Back tracking." The steps for traversing a binary tree in pre-order traversal.

Steps for pre order :-

Step 1: Visit the Root node (Display the data)
Step 2: traverse the left sub tree.
Step 3: Traverse the right sub tree.

order :-

ROOT → LEFT SUBTREE → RIGHT SUBTREE



procedure :- A → B → D → H → E → I → J → C → F → G → K → L

② In-order :-

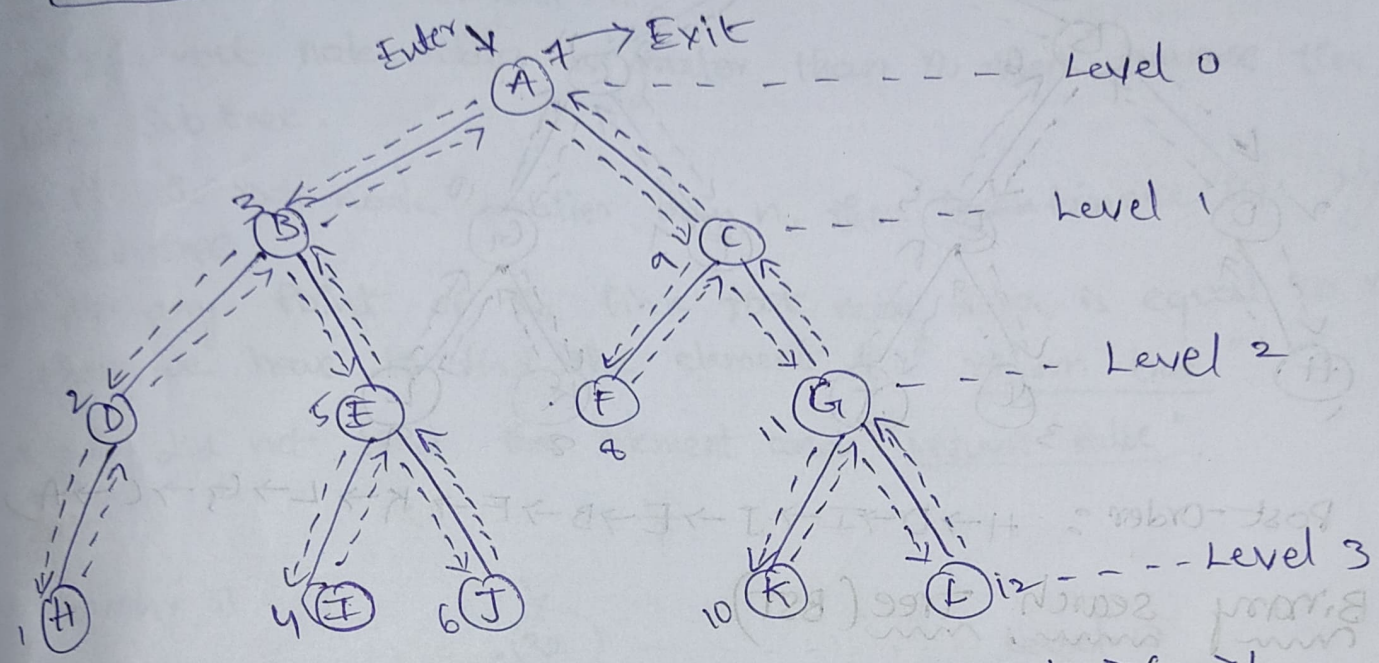
In a In-order traversal, the root node of each subtree is visited after its left sub-tree has been traversed but before the traversal of its right subtree begins. The steps for In order traversals.

Steps for In-order :-

Step 1 : Traverse the left sub-tree
Step 2 : Visit the root node (Display the data)
Step 3 : Traverse the right sub-tree.

Order :-

LEFT SUBTRE → ROOT → RIGHT SUBTREE



In-Order : H → D → B → I → E → J → A → F → C → K → G → L.

(3.) Post-Order :-

In a post order traversal tree. First traversal the left subtree and travers the right subtree after that visit the root node. The steps for post-order traversal is as follows.

steps :-

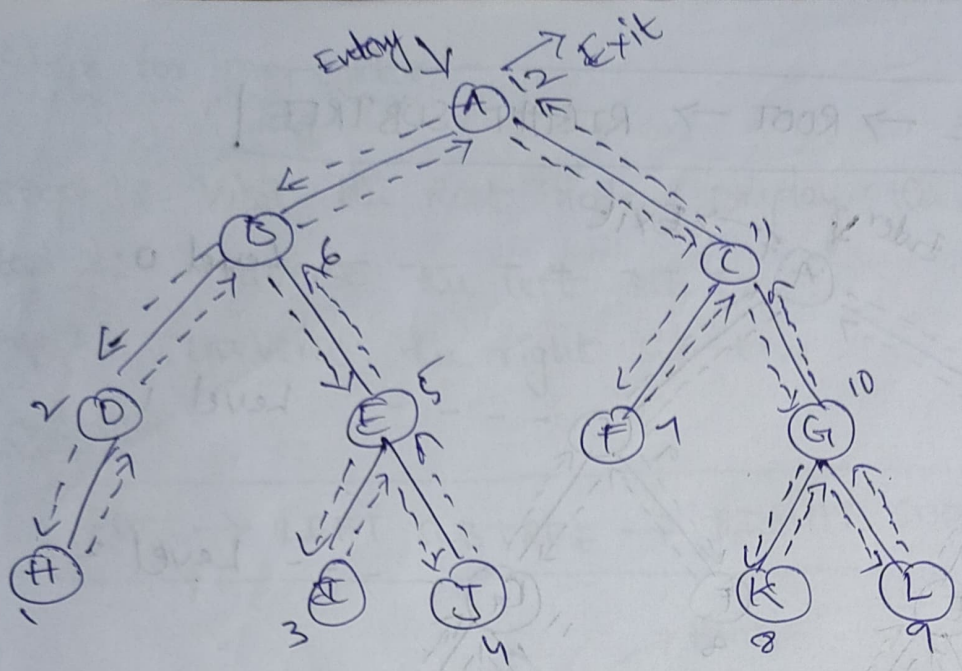
Step 1 :- Traverse the left subtree.

Step 2 :- Traverse the right subtree

Step 3 :- visit the root ~~node~~ node.

Order :-

LEFT SUBTREE → RIGHT SUBTREE → ROOT

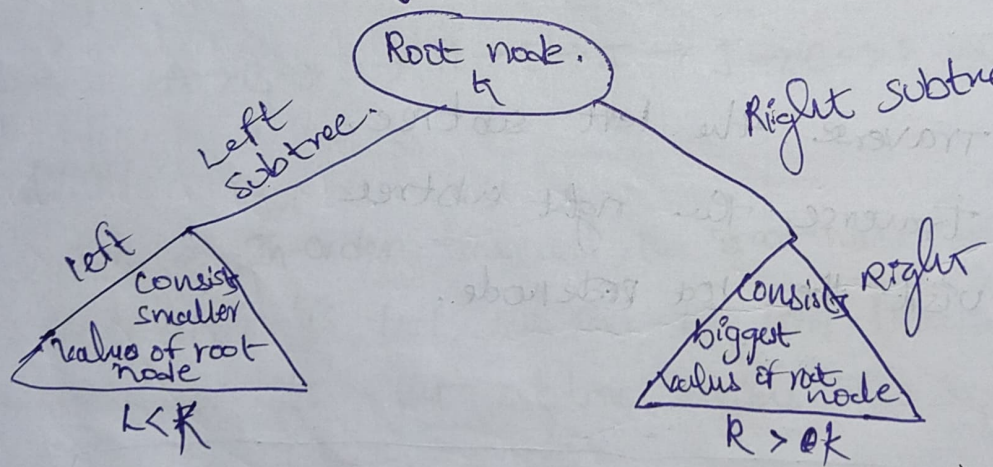


Post-Order : H → D → I → J → E → B → F → K → L → G → C → A

Binary search tree (BST)

Binary search tree is a one type of binary tree in which every node contains only smaller values in its left subtree and only larger values in its Right subtree.

Binary search tree is also known as binary sorted tree. An order of binary search tree is 2.



Binary search tree operations :-

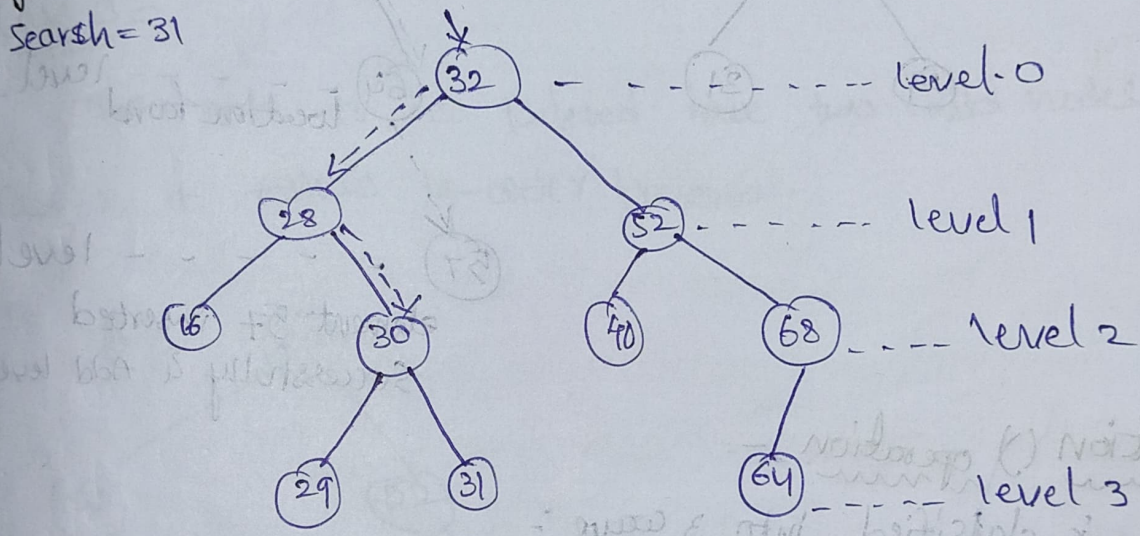
- Find (n)
- Insertion (n)
- Deletion (n)

find (n):-

find first from the root node and comparing the root node data with find element n.

* If root node data is greater than n, then traverse the left sub tree.
* If the root node smaller than n, then ~~lower~~ traverse the right sub tree.
* If any point of the time root node data is equal to n. then we have to find the element & "return true".
* If did not find the element and "return false".

Eg:-



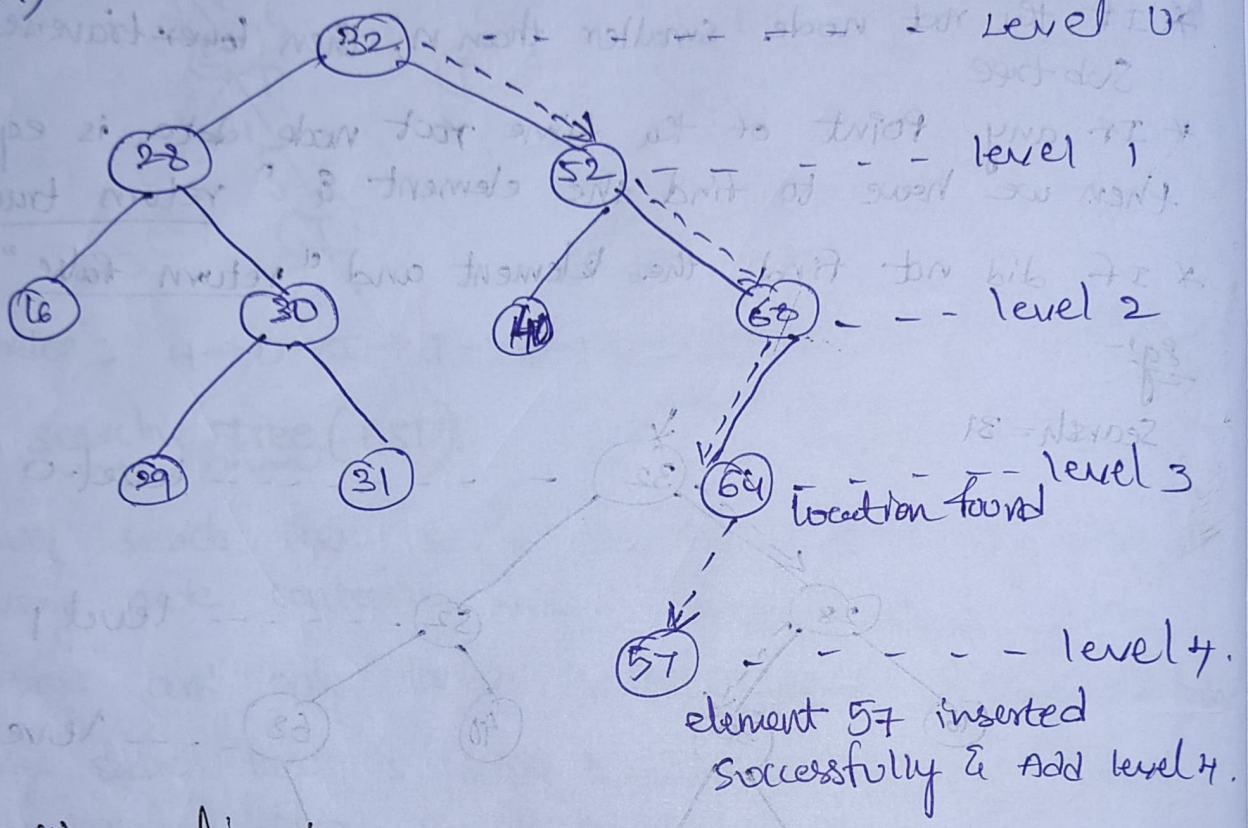
Insertion n :-

* to insert a node our first task is find the location.
* Now, take current = root node.
* start from the root node & comparing the root node data with find element n.
* If root node data is greater than n, the traverse the left subtree, vice versa.

* If any of time current is null that means we have reached to the leaf node and insert here with the help of parent node.

Eg:-

Insert (57)

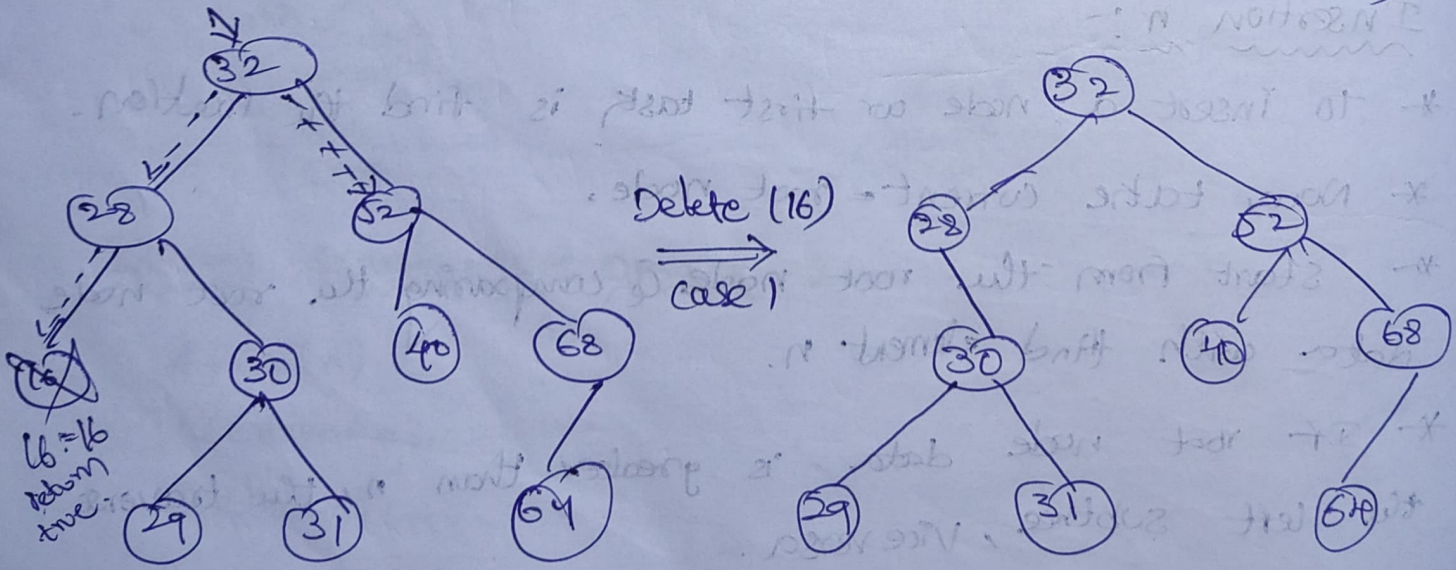


Deletion () operation :-

It is classified into 3 cases :-

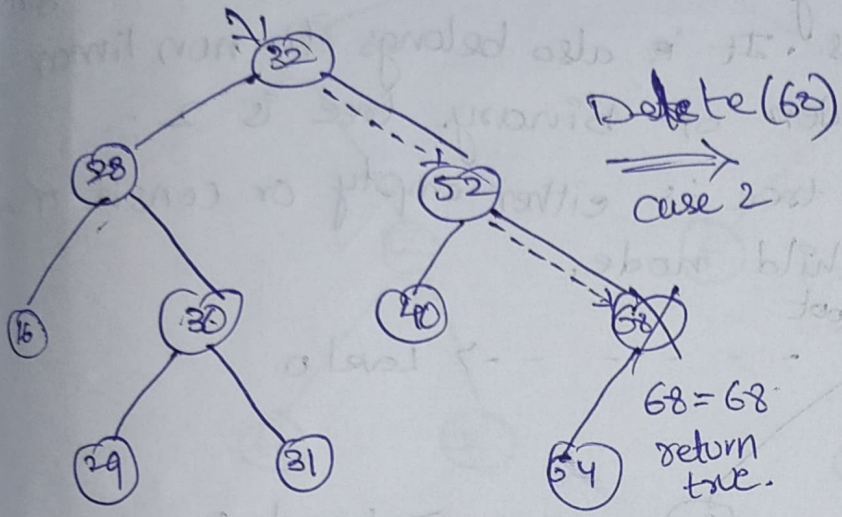
case 1 :-

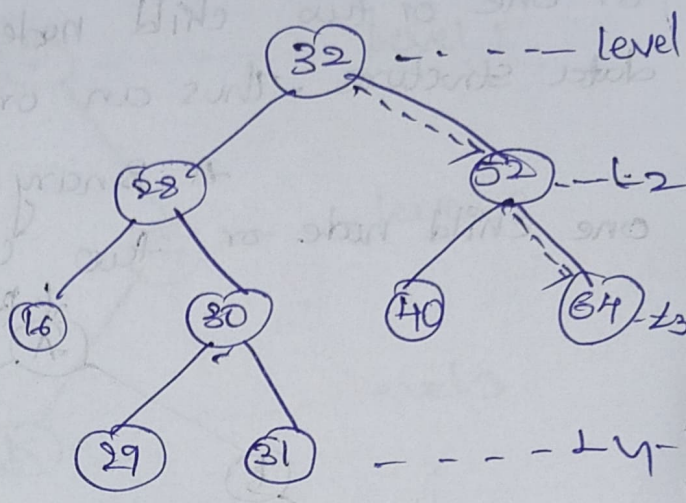
The node to be Deleted is a leaf node (Deleted leaf node)



Case 2:-

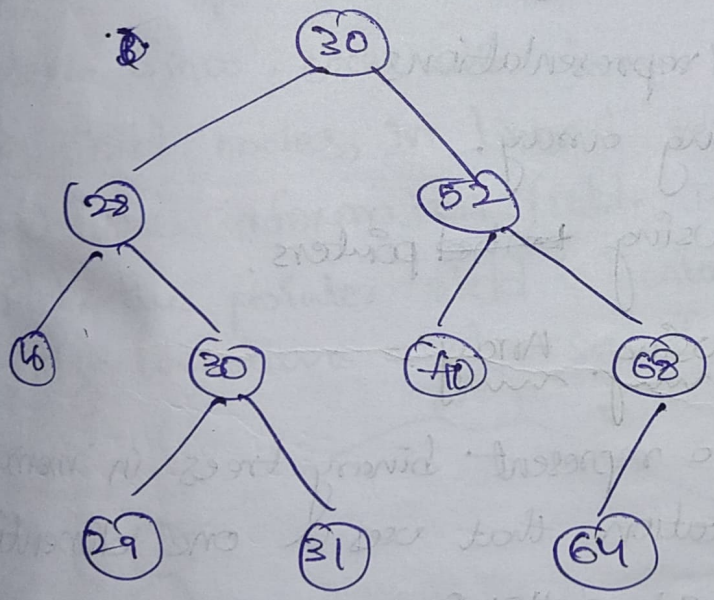
The node to be deleted has a one child node.



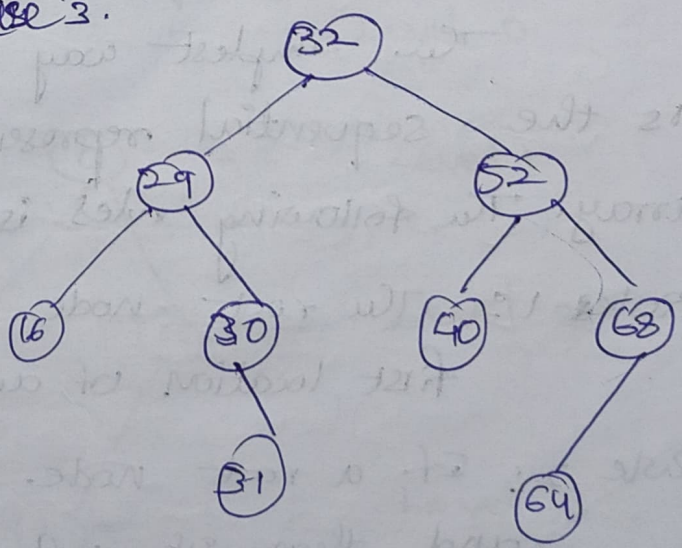


Case 3:-

The node to be Deleted have two child nodes in case 3, it follows in-order traversal.



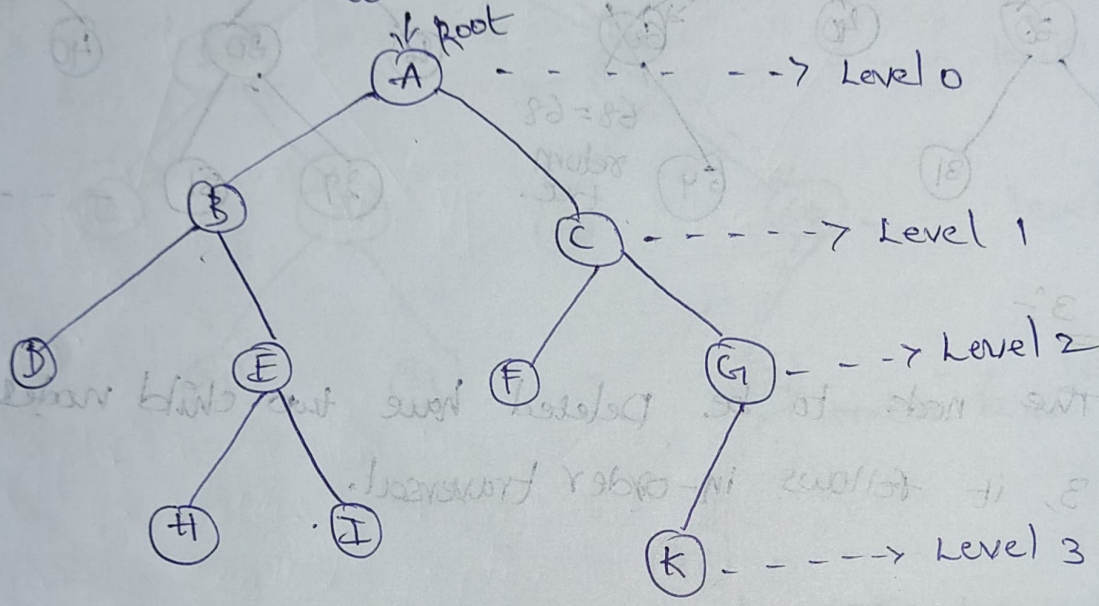
Delete(28)
case 3.



Representing the Binary tree:-

A Binary tree is a tree and it has zero or one or two child nodes. It is also belongs to non linear data structure. thus an order of binary tree is 2.

A Binary tree is either empty or consit of a one child node or two child node.



There are two types of representations.

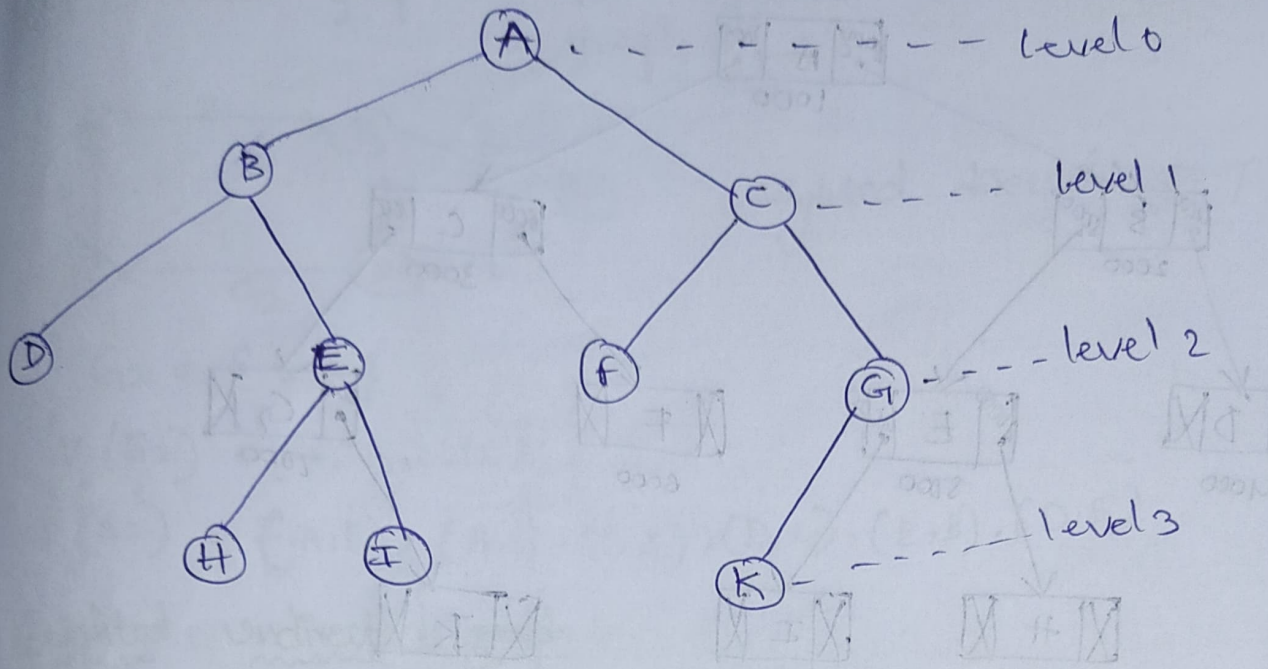
1. Sequential Representation using array.
2. Linked ~~rep~~representation using ~~linked~~ pointers.

1. Sequential Representation using Array:-

The simplest way to represent binary trees in memory is the sequential representation that uses a one dimentional array. The following rules is as follows

Rule 1: The root node of binary tree is stored in the first location of an array.

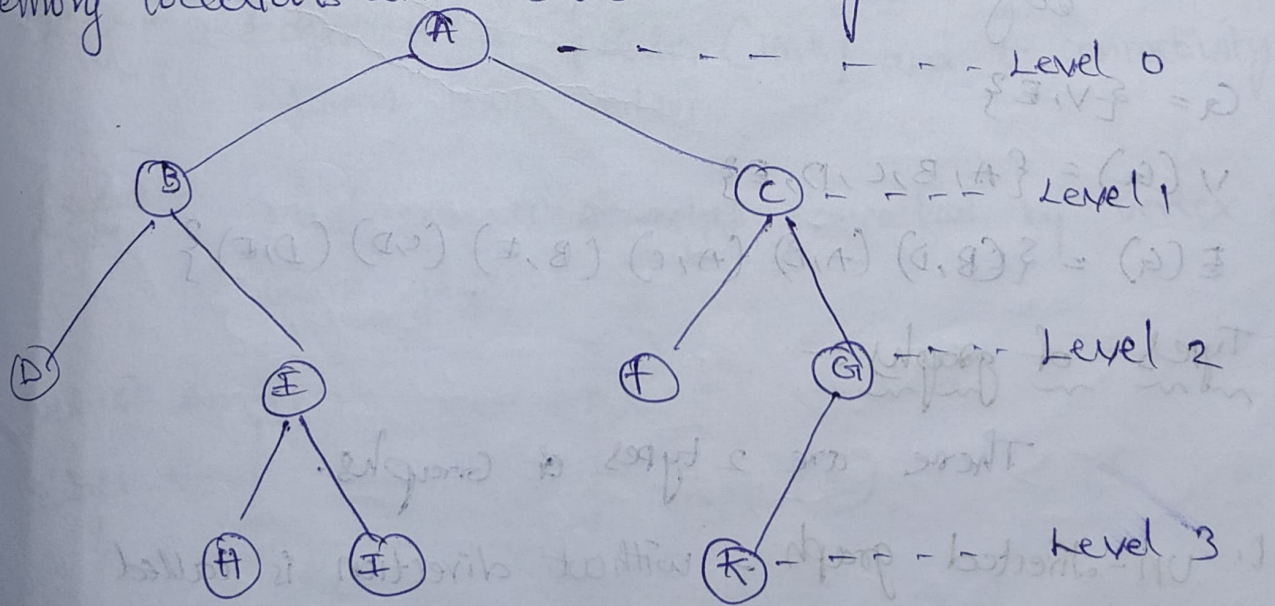
Rule 2: If a root node is J^{th} location of an array and then its left child is stored in $2J$. and right child stored in $2J+1$ location.

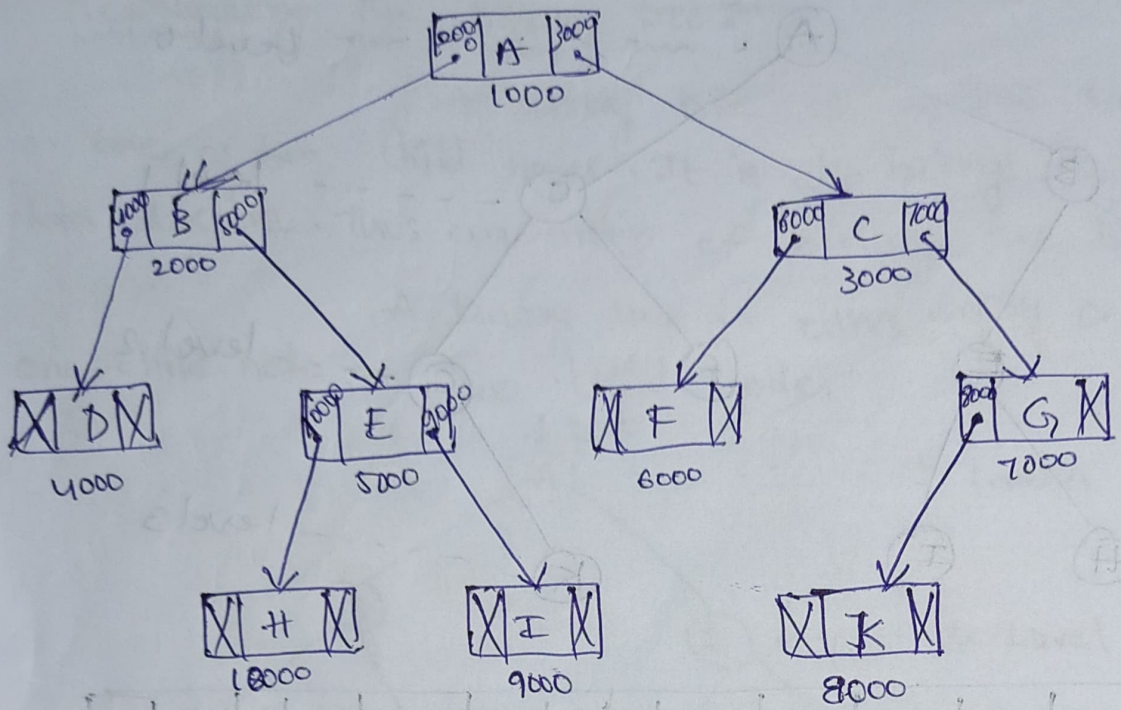


	A	B	C	D	E	F	G			H	I			K	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

2. Linked representation using pointers:-

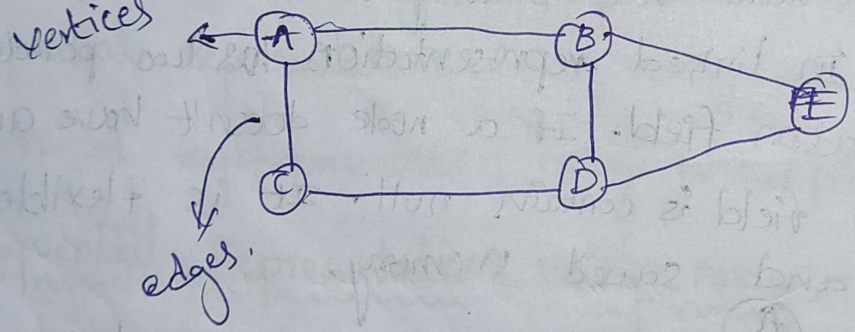
Linked representation of trees in memory is implemented using pointers. Since, each node in a binary tree can have maximum two child nodes, in linked representation has two pointers and one information field. If a node doesn't have any child, the pointer field is contain null. It is flexible memory locations and saved memory.





Graphs :-

Graphs is a non-linear data structure and its represents collection of vertices and connected by edges with cyclic formation is called graph.



undirected graph.

$G = \{V, E\}$

$V(G) = \{A, B, C, D, E\}$

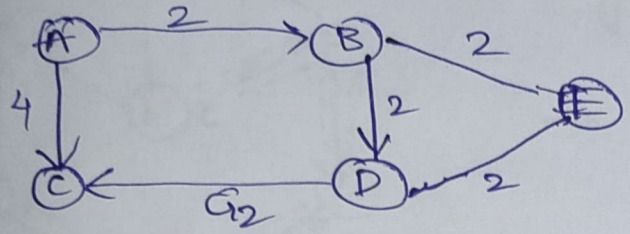
$E(G) = \{(B,D)\ (A,B)\ (A,C)\ (B,E)\ (C,D)\ (D,E)\}$

Types of graphs :-

There are 2 types of Graphs.

1. un-directed graph - without direction is called undirection.

(2) Directed graph :- with direction is called Directed graph.



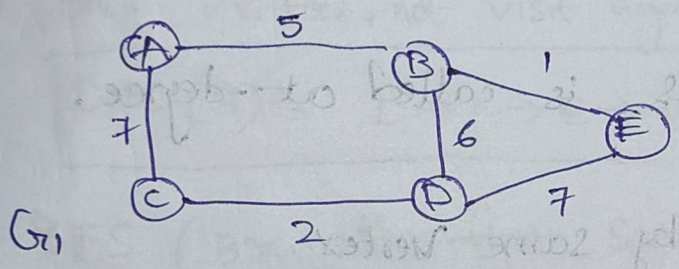
weighted directed group.

$G_2 = \{V, E\}$

$V(G_2) = \{A, B, C, D, E\}$

$E(G_2) = \{(A,B), (A,C), (B,D), (D,C), (E,B), (D,E)\}$

Weighted undirected graph :-



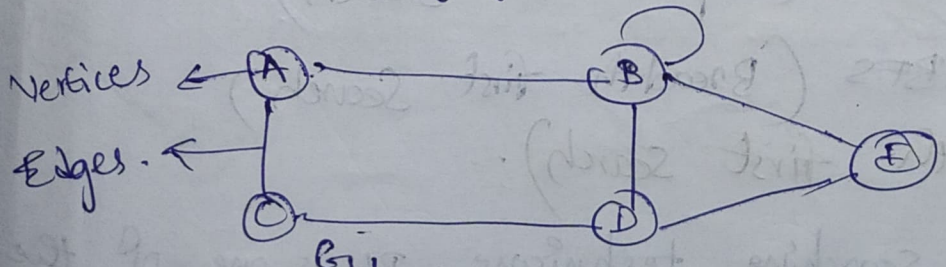
It contains weight (numbers)

Graph terminology :-

Vertex :- Vertex is nothing but "node" and it contains data. In the above graph A, B, C, D, E are called vertices.

Edge :- Edge is a pointer (link) and it connectivity between two vertices.

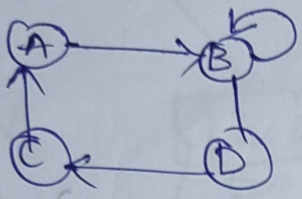
Degree :- maximum no. of edges connected by vertex in above graph.



Degree $(G_1) = \{^{2}A, ^{4}B, ^{2}C, ^{3}D, ^{2}E\}$

Degree $(G_1) = 4$.

In-degree Diagram :-



In-Degree $(G_2) = 2$

out Degree $(G_2) = 2$

Degree = 2

In Degree :-

total incoming edges is called In degree.

out Degree :-

total no of outgoing edges is called out-degree.

Loop :-

loop is a connected by same vertex

Path :-

Path is a traversed by source to destination.

Cyclic :-

with circle formation is called cyclic.

non cyclic :-

Without circle formation is called non-cyclic.

Graph Traverses :-

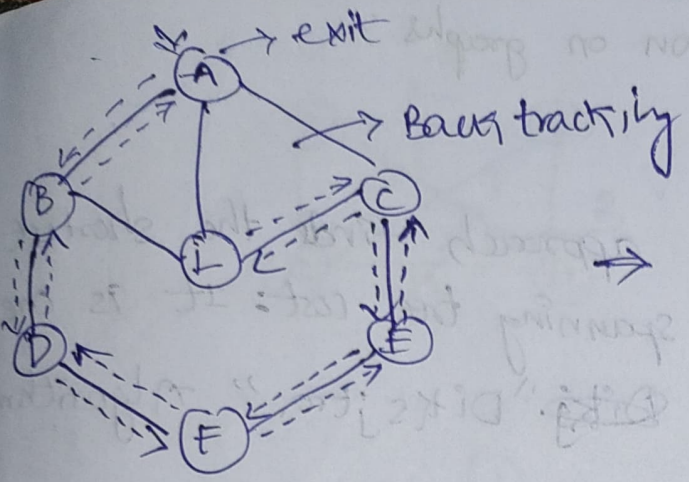
Two types of graph traverses

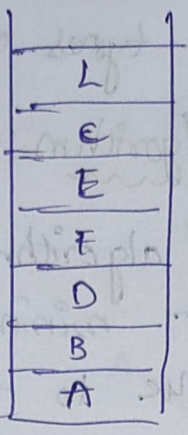
They are ① DFS (Depth first Search)

② BFS (Breadth first Search)

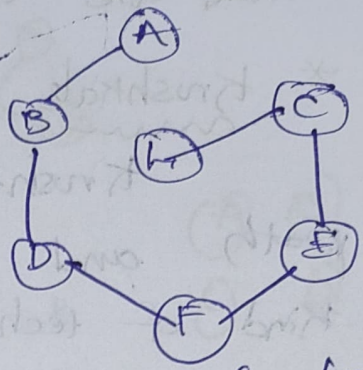
① DFS (Depth first search) :-

DFS is a searching technique. It is one of the application of stack.





Stack



Spanning tree

Rules:-

* visit all vertices
* without cyclic
* once visited, not visit again.

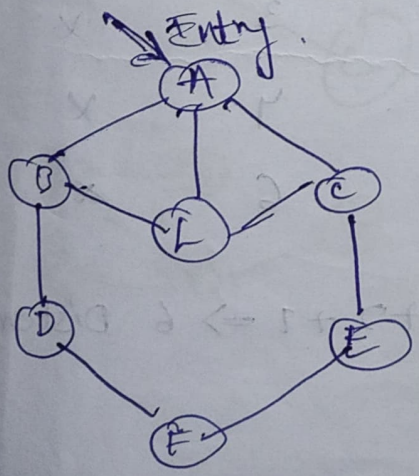
DFS (G) = L → C → E → F → D → B → A

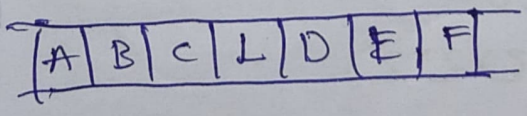
BFS (Breadth first Search)

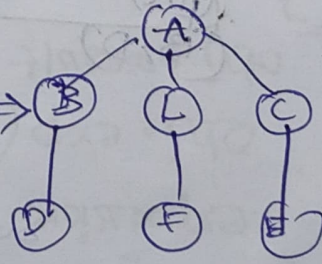
It is one of the application of queue.

Rules:-

* visit all vertices in graph.
* without cyclic
* once visited not visit again







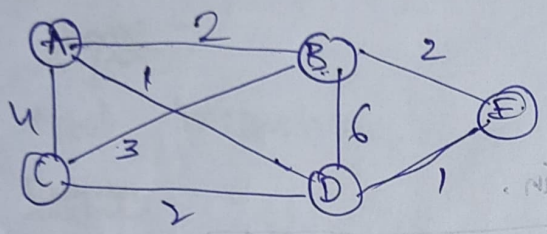
Spanning tree

BFS (G) = A → B → C → L → D → E → F

There are two types operation on graphs :-

* Krushkals Algorithm :-

Krushkal algorithm is approach finds the shortest path and get minimum spanning tree cost. It is one kind of technique to the ~~Dikj~~ "Diksjtra" Algorithm & greedy algorithm.



Spanning tree with minimum weight

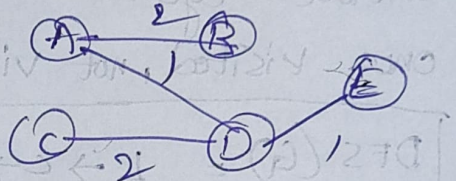


Table of krushkals :-

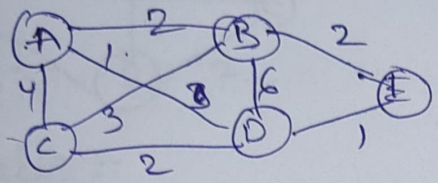
pair	min cost	selection.
(A, D)	1	✓
(D, E)	1	✓
(A, B)	2	✓
(B, E)	2	X
(C, D)	3	✓
(C, B)	4	X
(A, C)	6	X
(B, D)		

Pair	min cost	selection.
(A, D)	1	✓
(D, E)	1	✓
(A, B)	2	✓
(B, E)	2	X
(C, D)	2	✓
(C, B)	3	X
(A, C)	4	X
(B, D)	6	X

Sum of minimum cost = 1+2+2+1 => 6 Distance.

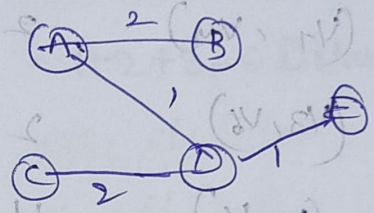
Prims Algorithm:-

Prime algorithm is a approach to find the shortest path and get minimum spanning tree cost. It follows the "matrix formation technique".



matrix formation:-

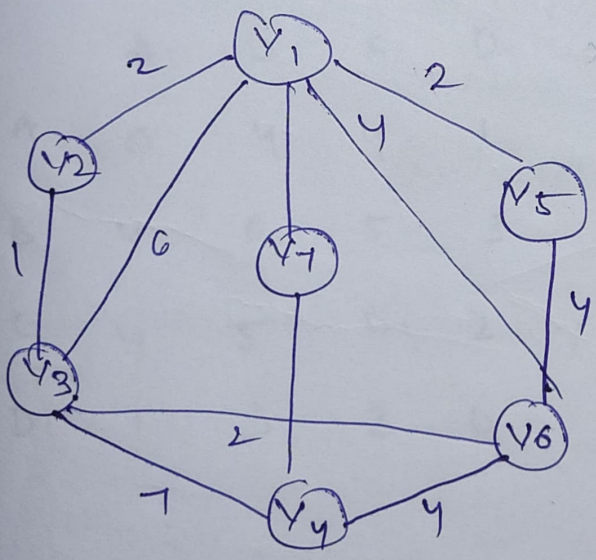
	A	B	C	D	E
A	0	2	4	1	∞
B	2	0	3	6	2
C	4	3	0	2	∞
D	1	6	2	0	1
E	∞	2	∞	1	0



minimum cost

= 1+1+2+2 = 6.

Krushkal's Algorithm:-



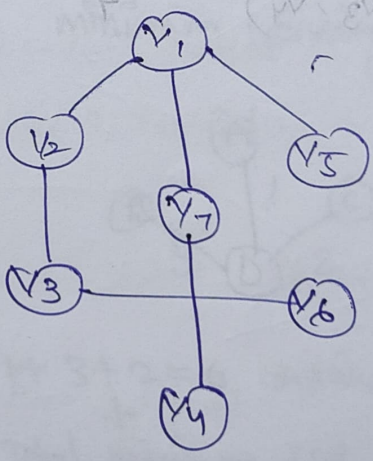
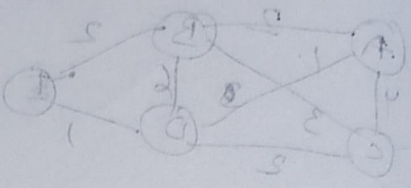


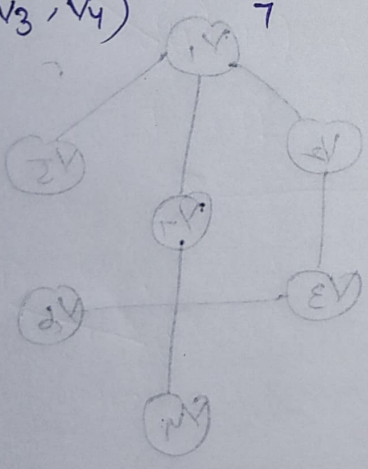
Table:-

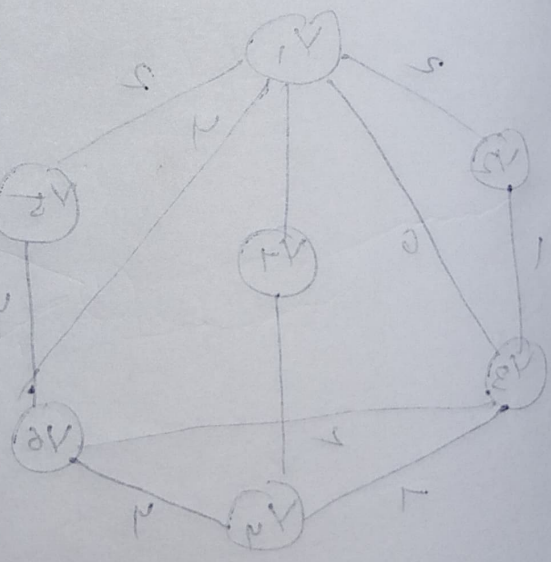
Pair	min cost	selection
(V_2, V_3)	1	✓
(V_1, V_2)	2	✓
(V_1, V_4)	2	✓
(V_7, V_5)	2	✓
(V_7, V_4)	2	✓
(V_3, V_6)	2	✓
(V_1, V_6)	4	✗
(V_4, V_6)	4	✗
(V_5, V_6)	4	✗
(V_1, V_3)	6	✗
(V_3, V_4)	7	✗



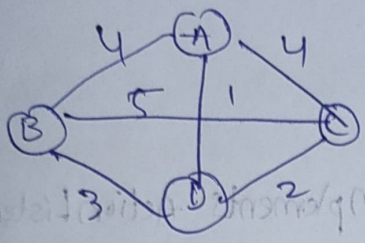
$1+2+2+2+2+2$

$= 11$





Eg:- To solve the following graphs by greedy algorithms such as kruskal's & prims.



minimum spanning tree

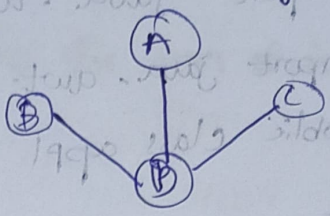


table :-

Pair	min cost	selection
(A,D)	1	✓
(C,D)	2	✓
(B,D)	3	✓
(A,B)	4	X
(A,C)	4	X
(B,C)	5	X

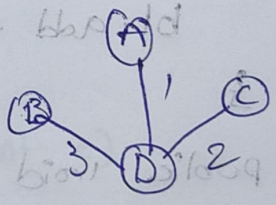
total min cost

= 1+2+3 = 6 Distance

Matrix formation :-

	A	B	C	D
A	0	4	4	1
B	4	0	5	3
C	4	5	0	2
D	1	3	2	0

minimum spanning tree



1+3+2 = 6 distance.

↓

Total minimum cost

Applets:-

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
Public class appl extends Applet implements ActionListener
{
  Text-field t1 = new Text-field (10);
  Text field t2 = new Text-field (10);
  Text-field t3 = new text field (10);
  Label l1 = new label (" Enter a value :");
  label l2 = new label (" Enter b value :");
  Label l3 = new label (" the Result :");
  Button b1 = new Button ("Addition");
  Public void init()
  {
    SetBackground (color.blue);
    add (l1); add (t1); add (l2); add (t2);
    add (l3); add (t3); add (b1);
    b1.add ActionListener (this);
  }
public void action performed (ActionEvent e)
{
  if (e.getSource == b1)
    {
      int a = Integer.parseInt (t1.getText());
      int b = Integer.parseInt (t2.getText());
      t3.setText (" " + (a+b));
    }
}
```

3

```
/* <applet code = "app1.class" width=500 height=500>
   </applet> */
```

write a programe in java, to compute the sum of n natural numbers?

```
impat java.awt.*;
import java.applet.*;
import java.awt.event.*;
Public class app2 extends Applet implements Actionlistener
{
 Textfield t1 = new Textfield (10);
 Textfield t2 = new Textfield (10);
 Label l1 = new Label ("Enter n value");
 Label l2 = new Label ("The Result");
 Button b1 = new Button ("factorial no's);
 Public void init ()
 Set Background (color.Black);
 add(t1); add(l1); add(l2); add(b1); add(t2);
 b1.add Action Listener (this);
}
Public void actionperformed (Action Event e)
{
  if (e.get source == b1)
  { int f=1, a=1;
    int a= Integer.parse Int (t1.get text ());
      while (a<=n)
       {
         f= f*a;
          a++;
```

```
    }
t3.setText("The factorial = "+f);
    }
  }
}
/* <applet code = "app2.class" width=600
   height=600 > </applet> */
```