Basic Traversal and Search Techniques :

These techniques are divided into two categories namely.

→ The Basic traversal techniques applicable to only to Binary trees. These techniques are referred to as Traversal methods.

→ The search Techniques applicable to graphs only. These Techniques are referred as search methods.

(1). Techniques for Binary Trees :-

Many operations can be performed on Binary Trees. Traversing is the operation on Binary Tree and visiting each node exactly once.

When traversing a Binary Tree, we have to visit each node and its subtrees in same manner. There are 3 tree traversal techniques namely.

1. Inorder traversal (LDR)
2. preorder traversal (DLR)
3. ~~post order~~ traversal (LRD)

Here 'L' stands for 'left subtree'

'D' stands for 'printing data'.

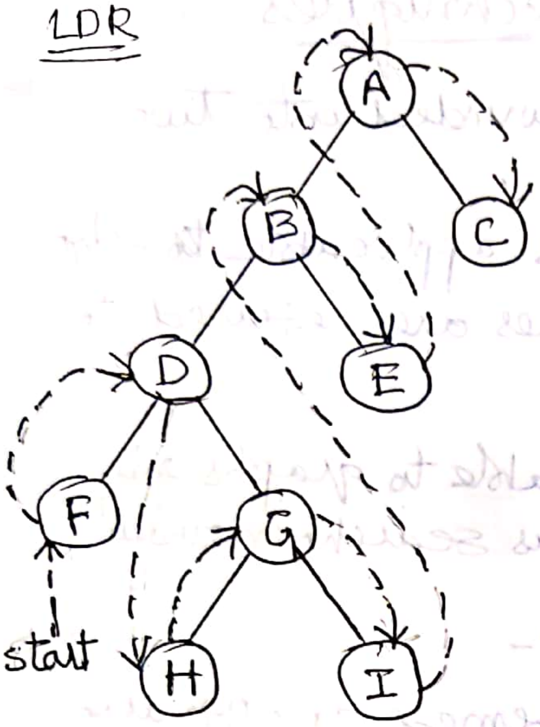
'R' stands for 'right subtree'.

(1). Inorder Traversal (LDR) In inorder traversal, first traverse the left subtree. In left subtree if a node has left child then it is visited. Next root node is visited and finally the right child is visited. This process is continued till all the nodes of a binary tree are visited.

Example

consider a Binary tree of the followings.

LDR



∴ Inorder traversal ~~of a Binary~~ order of Binary tree is

FDHGIBEAC

Recursive algorithm for Inorder traversal of a Binary tree

```
Algorithm Inorder(t)
// 't' is a Binary tree, each nodes
// has 3 fields lchild, data,
// and rchild.
{ if (t ≠ 0) then
  { Inorder(t → lchild);
    visit(t);
    Inorder(t → rchild);
  }
}
```

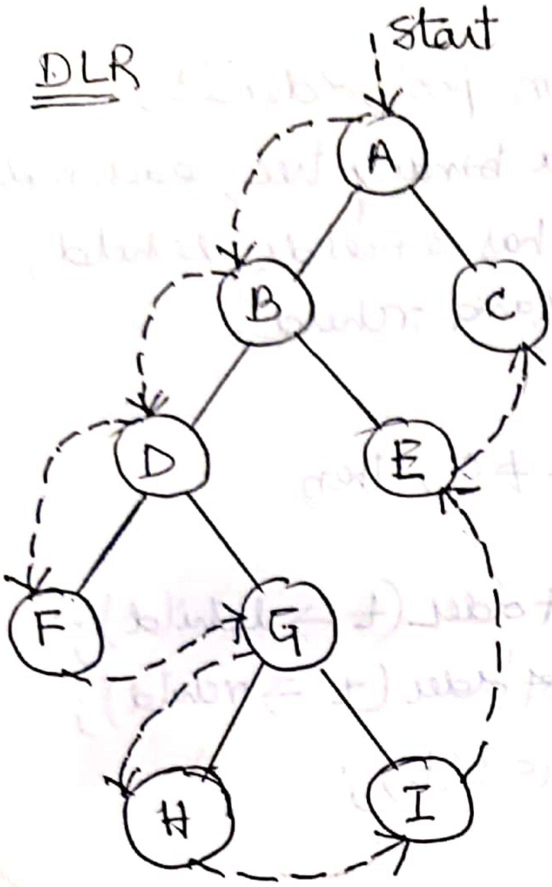
(2). Preorder Traversal (DLR) :-

In preorder traversal, first root node is visited. If the root node has a left subtree then it is visited, and then preceeds to traverse the right subtree. This process is continued till all the nodes of a binary tree are visited.

Example :

consider the Binary of the following

DLR



preorder traversal order is

ABDFGHIEC.

Algorithm preorder(t)
// 't' is a binary tree, each node
// of 't' has 3 fields lchild,
// data and rchild.
{
 if (t ≠ 0) then
 {
  visit(t);
  preorder(t → lchild);
  preorder(t → rchild);
 }
}

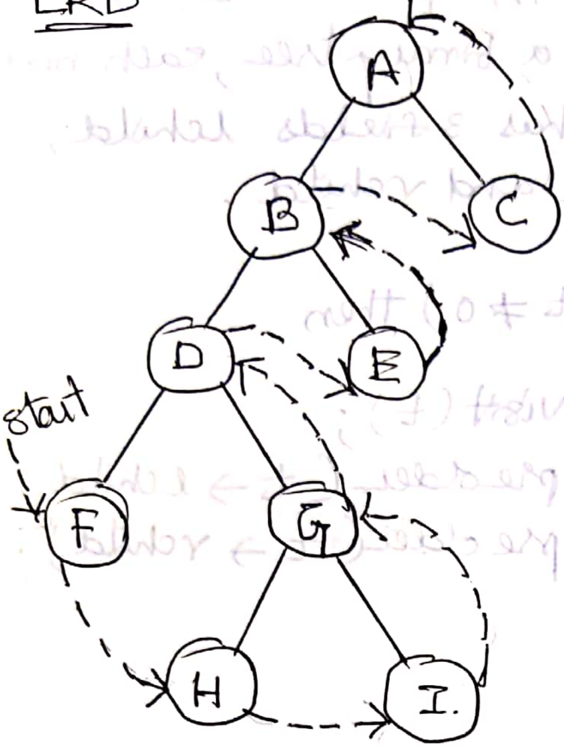
(3). post order Traversal : (LRD) :-

In postorder traversal, first traverse the left subtree. If the left subtree has left child then it is visited. After that the right subtree is traversed and finally the root node is visited. This process is continued till all the nodes of a binary tree are visited.

Example:

consider the followings binary tree.

LRD



post order traversal order is

FHIGDEBCA

Algorithm post order(t)

// 't' is a binary tree, each node
// of 't' has 3 fields lchild,
// data and rchild.

```
{
    if(t ≠ 0) then
    {
        post order(t → lchild);
        post order(t → rchild);
        visit(t);
    }
}
```

Techniques for graphs :-

There are two standard ways to traversal of a graphs

(1). Breadth First search (BFS) uses an auxilary structure to hold the nodes for processing.

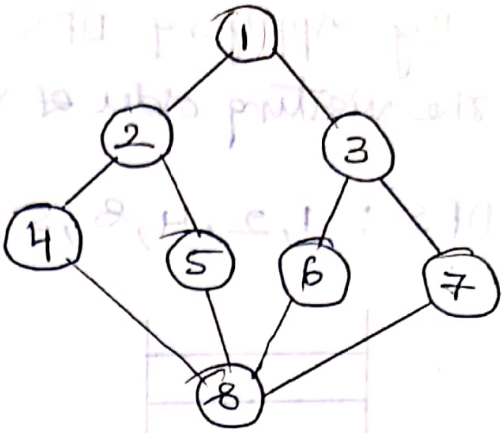
(2) Depth First search (DFS) uses a stack as an auxilary structure to hold the nodes.

(1). Breadth First search (BFS) :-

In BFS first examine the starting node 'v', then examine all neighbour nodes of 'v', then examine all the neighbour nodes of the neighbour nodes of 'v' and so on. This algori method uses the queue data structure.

For example

consider the following graph 'G'



By applying BFS algorithm, the visiting order of vertices is

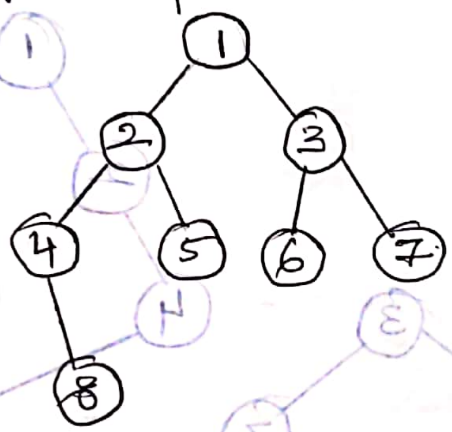
BFS : 1, 2, 3, 4, 5, 6, 7, 8

Queue

~~1~~	~~2~~	~~3~~	~~4~~	5	6	7	8

~~Queue~~

The BFS spanning tree of the above graph is as shown below



BFS order is

1, 2, 3, 4, 5, 6, 7, 8

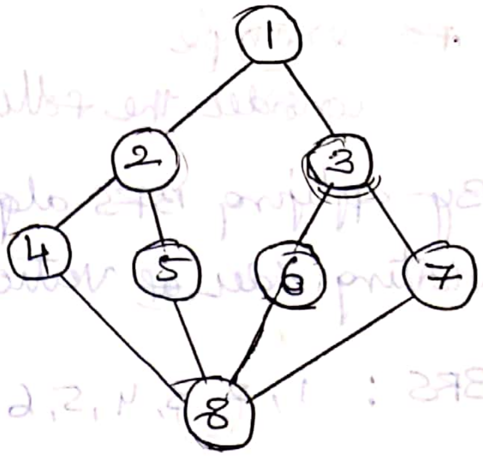
(2). **Depth First search (DFS) :**

In DFS search method, first examine the starting vertex 'v' then examine each vertex along a path 'p' which begins at 'v'. i.e process the neighbours of 'v' then neighbours of neighbours of 'v' and so on. After the end of the path 'p' backtracks on 'p' and continue along the path 'p' and so on. This method uses the stack data structure.

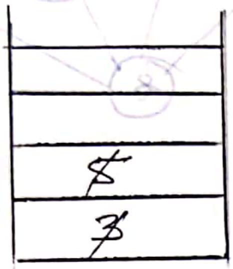
For example

consider the following graph 'G'

By Applying DFS method
The visiting order of vertices is

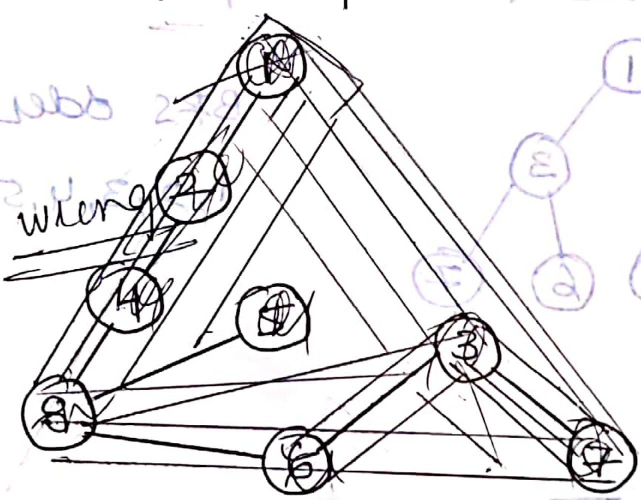


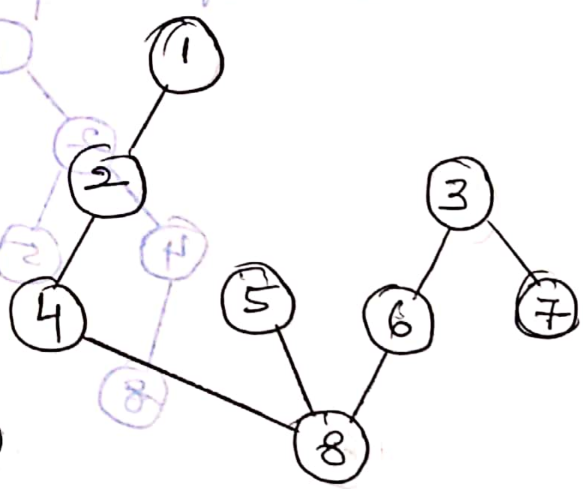
DFS : 1, 2, 4, 8, 5, 6, 3, 7



stack

The DFS spanning tree of the above graph is as shown below.





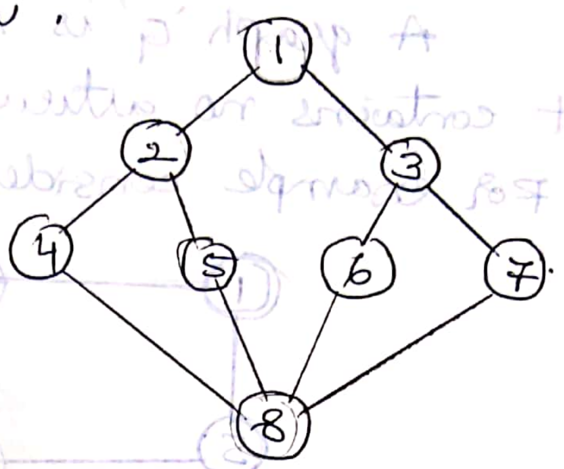
In DFS of a graph 'G' from starting vertex-1 results in the order
1, 2, 4, 8, 5, 6, 3, 7.

Connected components :

A graph is said to be connected if there exists a path between any two vertices. If the graph 'G' is connected graph then we can visit all the ~~vertic~~ vertices ~~of~~ the graph.

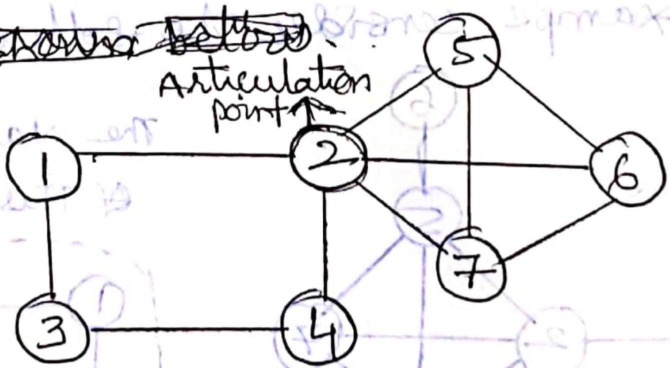
The sub graph obtained after traversing a graph using BFS and DFS represents the connected components of the graph.

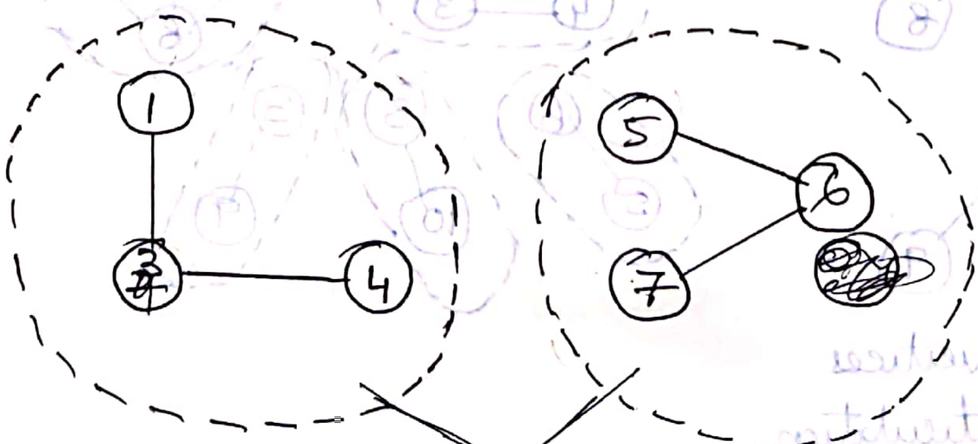
The BFS & DFS spanning trees of the this graph are ~~sep~~, represented in BFS and DFS search methods.



Bi-connected components :-

Articulation point : Let $G = (V, E)$ be a connected undirected graph then the articulation point of a graph 'G' is a vertex, whose removal disconnects the graph into two or more components. This articulation point is a kind of 'cut' vertex. This is illustrated as ~~shown below~~ shown below.



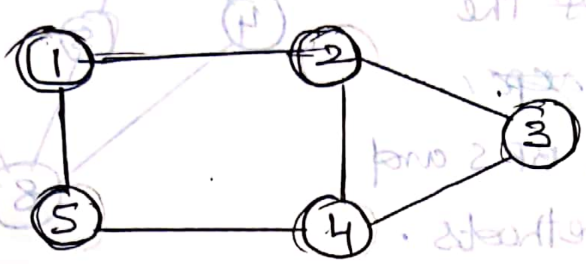


Two disjoint graphs

Bi-connected Components :-

A graph 'G' is said to be Bi-connected if it contains no articulation point.

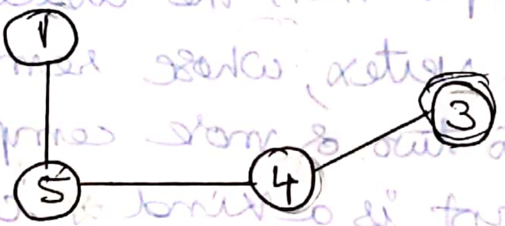
For example consider the graph of the following



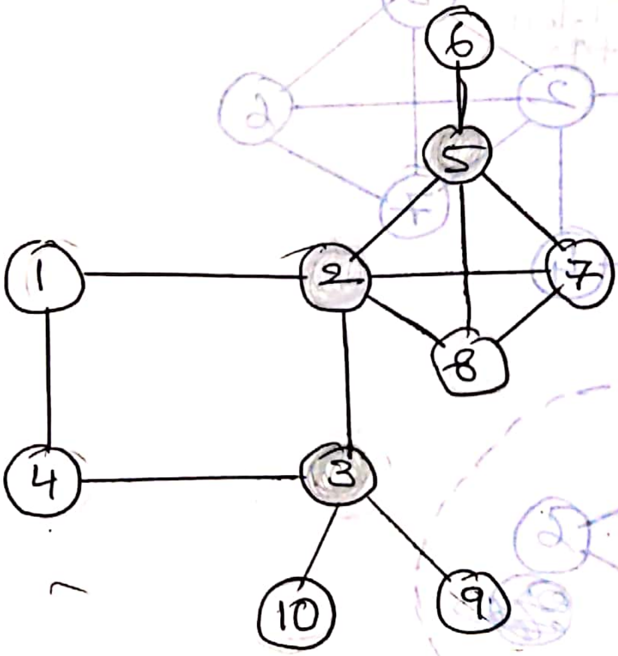
Bi-connected graph

Here ~~we~~ even though remove any single vertex we do not get disjoint graphs.

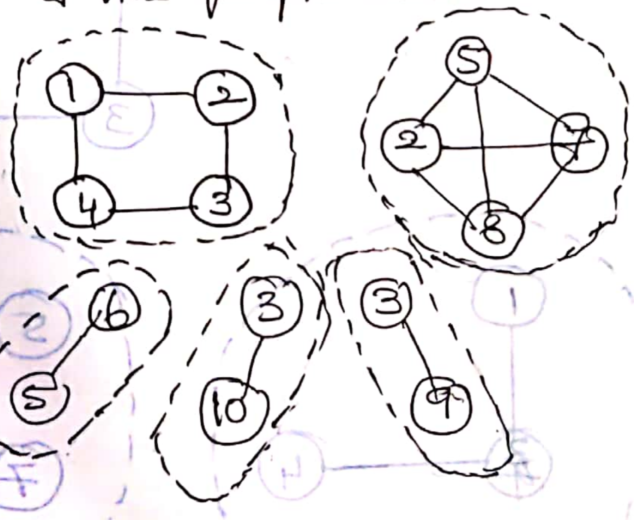
Let us remove the vertex 2 we get.



For example consider the following graph.



The various bi-connected components of this graph are



In this graph the vertices 2, 3, and 5 are articulation points.