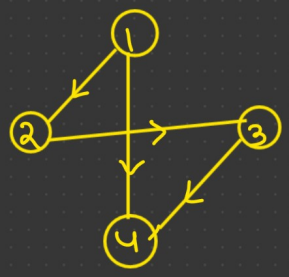
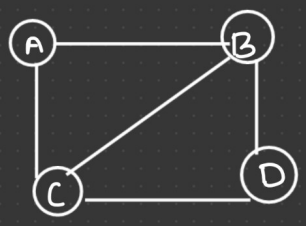
Graph

A graph G can be defined as an ordered set G(V, E), where V is set of vertices & E is collection of Edges which are used to connect vertices.





Ex :- G(V, E)

$V = \{1, 2, 3, 4\}$

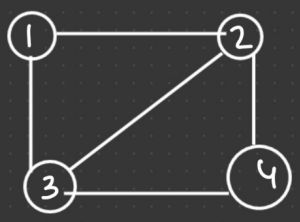
$E = \{(1,2), (3,4), (2,3), (1,4)\}$

Ordered pair :- (a,b) is said to be ordered pair when $(a,b) \neq (b,a)$.

Unordered pair :- (a,b) is said to be unordered pair when $(a,b) = (b,a)$

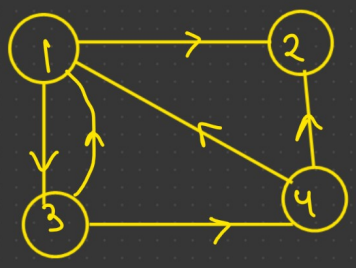
Directed & Undirected Graph

Undirected Graph :- In this graph edges are not associated with the direction with them. It mean if their is an edge between 1 to 2 then we can say that (1, 2) & (2, 1) both exist.



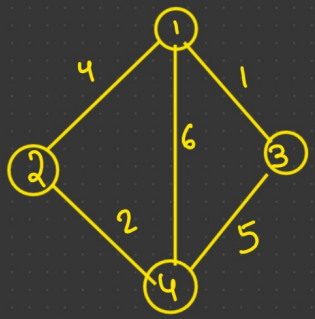
Directed Graph :- In this graph edges are associated with the direction with them. It means if there is an edge between (1, 2) then we can say only path exists from 1 to 2 . (1, 2) ≠ (2, 1) . It is also known as "Diagraph".

Ex:-



Weighted Graph :- In this graph every edge is associated with a weight/length in it. It must be +ve . It is also known as cost of an edge.

Ex:-

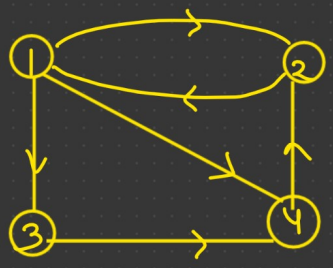


w(1, 2) = 4

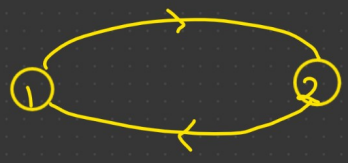
w(3, 4) = 5

Sub - Graph :- A Graph H is said to be Subgraph of G. iff for all edges & vertices of H belongs to graph G.

G =



H =



H is subgraph of G.

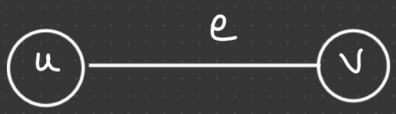
Graph Terminology :-

1) **Loop** :- An edge that is associated with similar end points can be called as loop.

loop

self loop

Adjacent Nodes :- If two nodes u & v are connected via an edge e, then u & v are known as adjecent Nodes / neighbours.

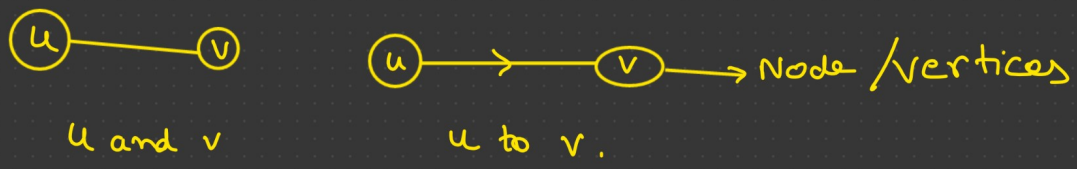


Note :- For directed Graph, (u, v) is an edge in G.
⇒ "v is adjacent to u". but reverse is not true.

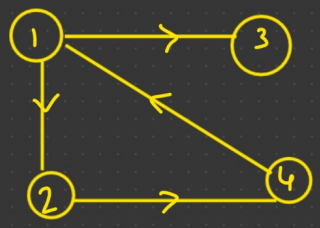
Incident :- It is a relation between an edge & vertices.

In an undirected graph (u, v) is incident from u and v.

In an directed graph (u, v) is incident from u to v.



Degree of Nodes :- A degree of a node is the number of edges that connected to that node.



degree (1) = 3
degree (4) = 2

Indegree (1) = 1
Outdegree (1) = 2

path (1, 4) = 1 → 2 → 4

Indegree of Node :- Number of edges pointing "to" Node.

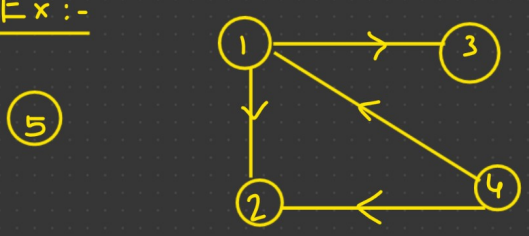
Outdegree of Node :- Number of edges going "from" the Node.

Note :- For undirected grap Indegree == outdegree.

Degree of Node = Indegree + Outdegree.

Source :- A vertex, that has no incoming edge, but has some outgoing edges is called source vertex (Indegree = 0)

Ex :-



Source = 4

Sink = 2, 3

Pendant vertex = 3

Isolated vertex = 5

Sink :- A vertex, that has no outgoing edge, but has some incoming edges is called sink vertex. (outdegree = 0)

Pendant Vertex :- A vertex, is said to be pendant vertex if Indegree is equal to 1 and outdegree is equal to 0.

Isolated Vertex :- A vertex, which has zero degree is Called isolated vertex.

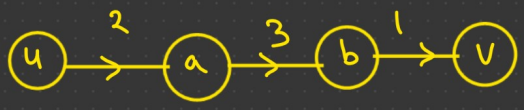
Path :- A path can be defined as sequence of Nodes that followed in the order to reach some terminal node.

Closed path :- A path which same same intial & final vertex is called closed path.

Length of the path :- Number of edges in the path is called length of the path.

Length (u, v) = 4

For weighted graph :-

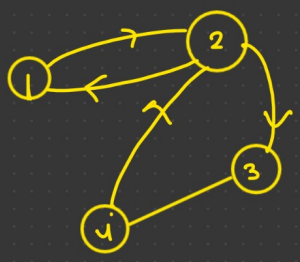


length(u, v) = 6

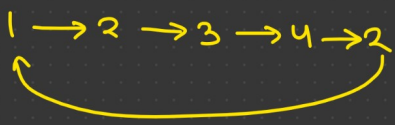
Reachable :- If there exist a path from u to v then we can say that v is reachable from u.

Cycle :- A path that has same intial & final vertex.

Simple Closed path :- If all the nodes of the graph are distinct in the given path except intial vertex.

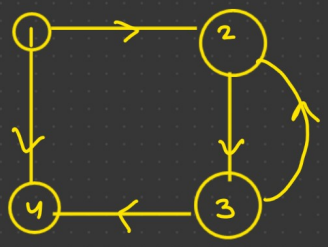


simple path / simple cycle

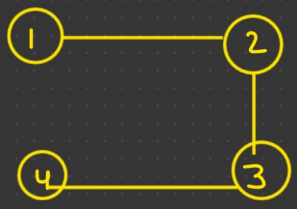


closed path. / Cycle.

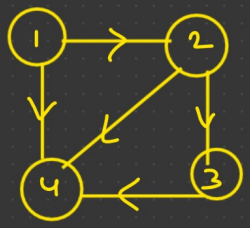
Cyclic Graph :- A graph that contain cycle.



Acyclic Graph :- A graph that does not contain any cycle.



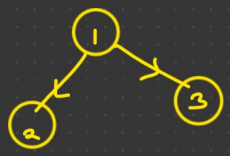
If a directed graph has no cycle, then it is known as DAG.
(Directed Acyclic Graph).



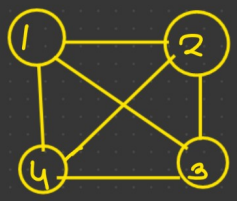
Null Graph :- A graph has only isolated vertex.

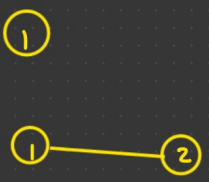
⊙

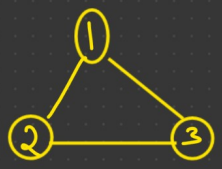
Connected Graph:- A graph which has no isolated vertex is called connected graph.



Complete Graph:- A graph in which each node is connected to every other nodes in the graph.







Maximum no. of edges :- If n is the no. of vertices.

For undirected graph = $(n-1)+(n-2)+(n-3)+\ldots+1$

$$= \frac{(n-1)(n-1+1)}{2} = \boxed{\frac{n(n-1)}{2}}$$

For directed graph :- $\boxed{n \times (n-1)}$

Formula:- $1+2+3+\ldots+n = \frac{n(n+1)}{2}$

$(n-1) + (n-1) + (n-1) \ldots\ldots$ n times

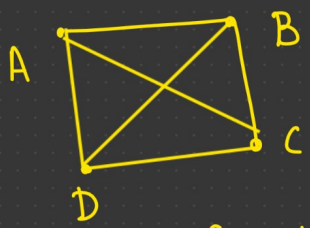
$n \times (n-1)$

$(n-1)$ — 1
$(n-1)$ — 2
$(n-1)$ — 3
⋮
$(n-1)$ — n

Self loop :- An edge having the same initial & final vertex.

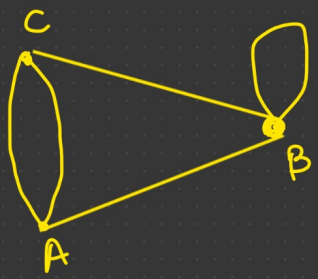
Multiple / parallel Edge :- All the edges having same pair of vertex.

Simple Graph :- A graph that does not have any loop or parallel edge.

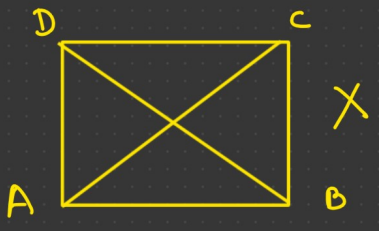


Simple graph

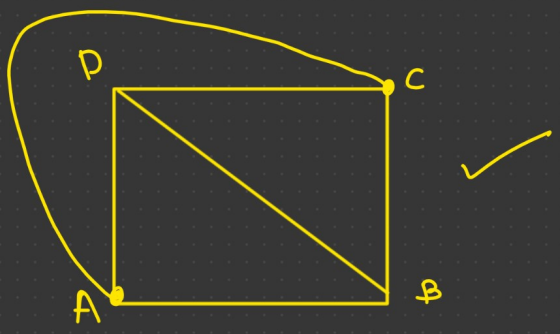
Multi Graph :- A graph having multiple Edges.



Planer Graph :- It can be drawn in a plane without any 2 intersecting edges.



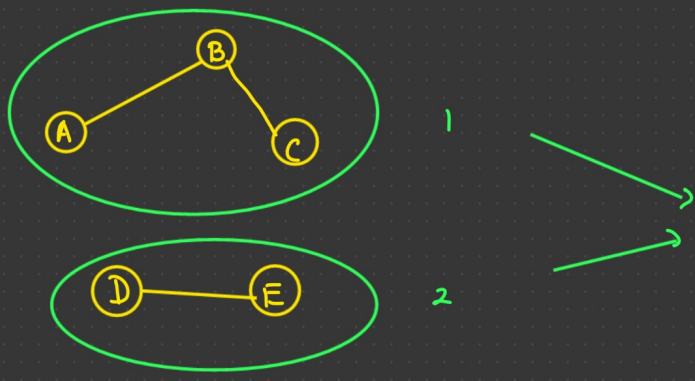
X



✓

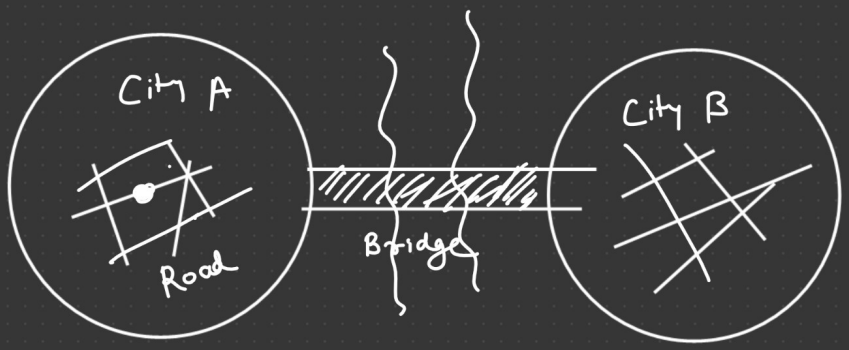
Connected Component :-

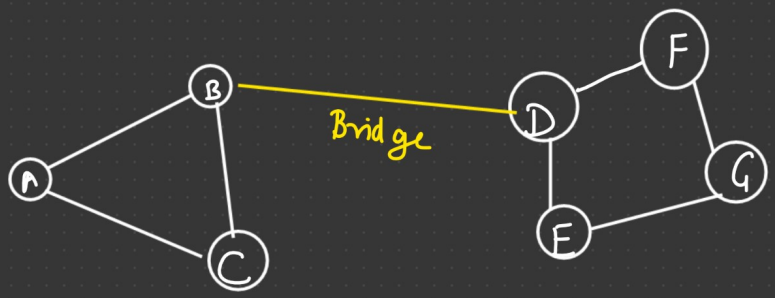
It is maximal connected subgraph.



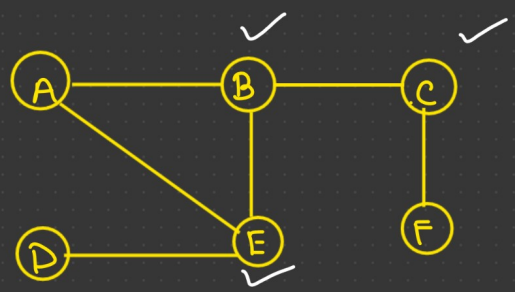
Bridge:-

If on removing an edge from a connected graph, the graph becomes disconnected then this edge is called Bridge.

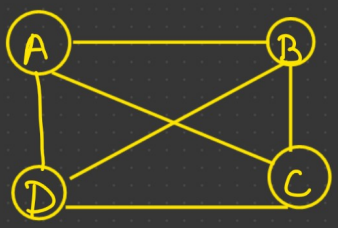




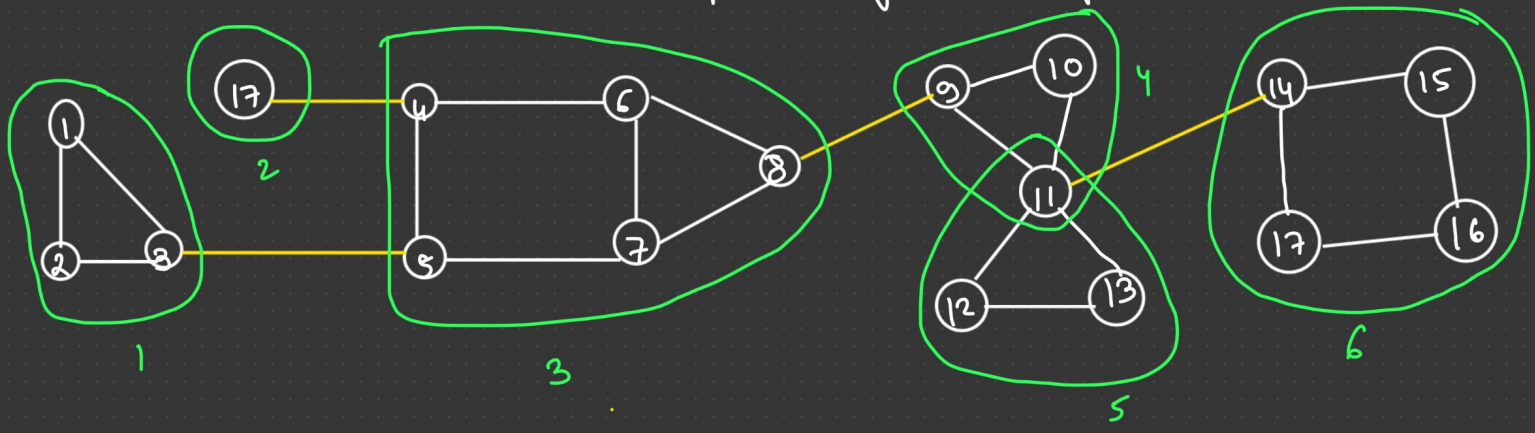
Articulation Point :- If on removing a vertex from a connected graph, graph becomes disconnected then that vertex is called Articulation point / Cut vertex.



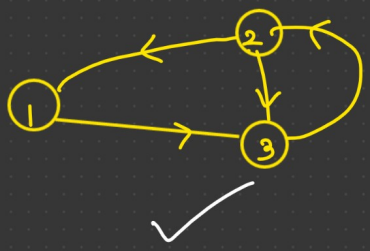
Biconnected Graph :- A graph which has no articulation point.

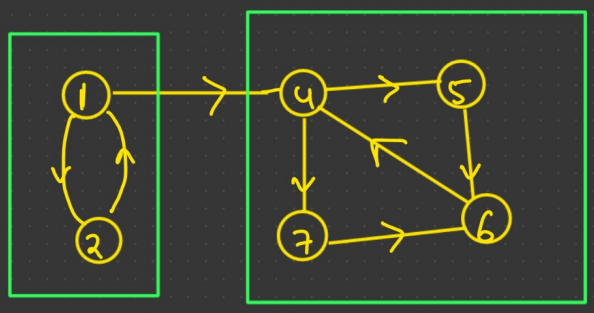


Biconnected Components :- A biconnected component is the maximal connected component of Subgraph.



Strongly Connected Components :- A diagraph is said to strongly connected if for any pair of vertices, if there is a path from u to v & also from v to u.





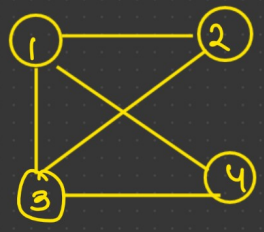
Weakly Connected Component :- A digraph is said to be weakly connected Component if their is a path from u to v "or" v to u "or" both.

Tree :- An "undirected graph" T is called tree if it has "no cycle". There is exactly one simple path from u to v.

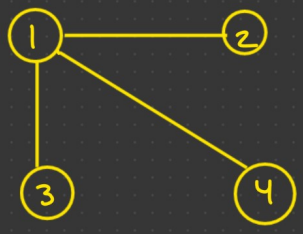
- n vertices ⇒ exactly "n-1" edges.
- All the edges in graph are bridge then it is a tree.
- If any edge is added to the tree then a simple cycle is formed.

Spanning Tree :- A subgraph T of a connected graph G, which contains all the vertices of G is called spanning tree of G.

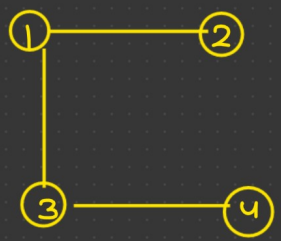
- Spanning Tree of a graph is not unique.



Graph

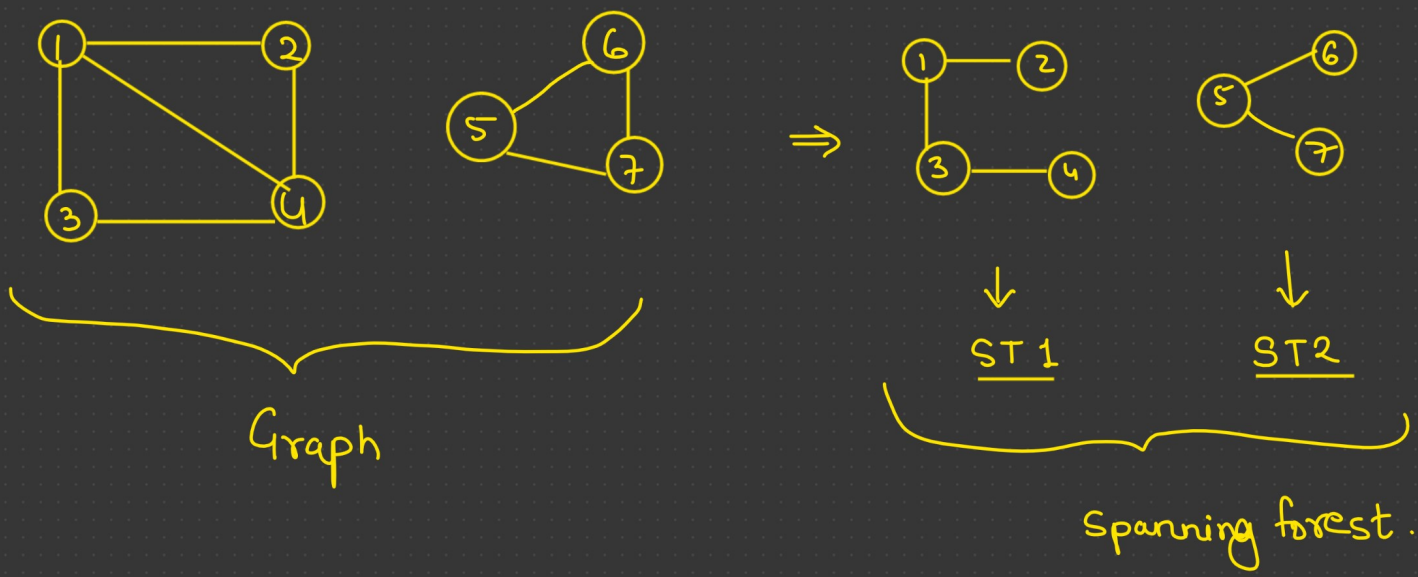


Spanning Tree ✓



Spanning Tree ✓

Spanning Forest :- A spanning forest is a subgraph that consists of spanning Tree for each connected component of a graph.



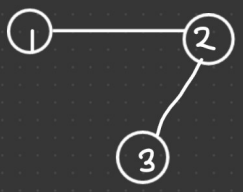
Graph Representation

1) Adjacency Matrix :-

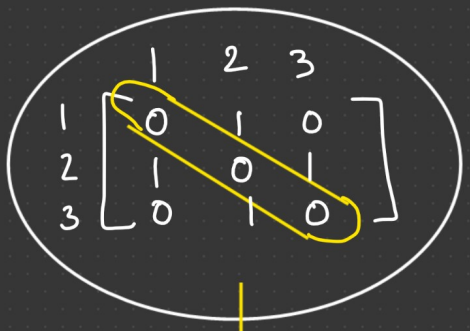
An Adjacency Matrix is a 2D array of V × V vertices. Each row & column represents a vertex.

If a[i][j] = 1 ⇒ There is an edge between i → j

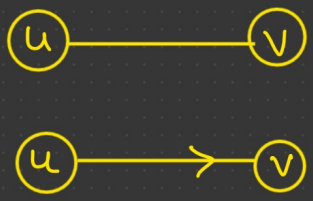
If a[i][j] = 0 ⇒ There is no edge between i → j



undirected



Symmetric matrix



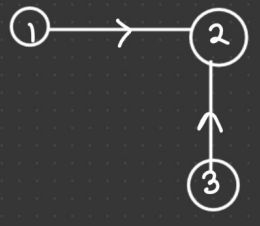
Yes = 1
No = 0

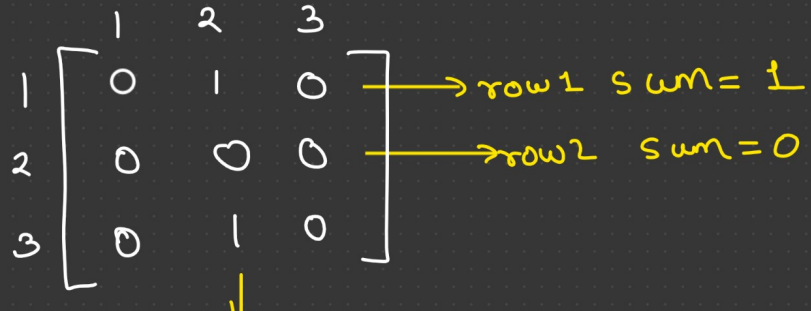
$aR_b \Rightarrow bRa$

a[2][1] = a[1][2]

a[3][1] = a[1][3]

☆ Undirected graph always have Symmetric Matrix.





Column 2 sum = 2

Advantages & Disadvantages of Adjacency Matrix:-

- Easy to use.
- Edge lookup is extremely fast.
- In a Diagraph, Row of sum represents outdegree and column sum represents indegree.
- Good for dense graph.

```
if(a[1][2] == 1)
{
  cout<<"Edge Found";
}
```

[number of non-zero elements > number of zero element, then it is called Dense Graph. otherwise it is a sparse graph.]

- Not good for sparse graph.
- It is not easy to add/remove any vertex/edge in graph at runtime using adjacency Matrix.

Code implementation :-

```
int mat[20][20];
int V, E, u, v;
cout<<" Enter no. of vertex and Edge";
cin>>V>>E;
for(int i=0 ; i<E ; i++)
{
    cin>>u>>v;
    mat[u][v] = 1;
    mat[v][u] = 1;
}
```

Undirected graph

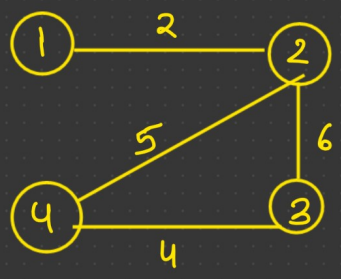
```
mat[u][v] = 1;
mat[v][u] = 1;
```

Directed graph

```
mat[u][v] = 1;
```

For weighted graph :-

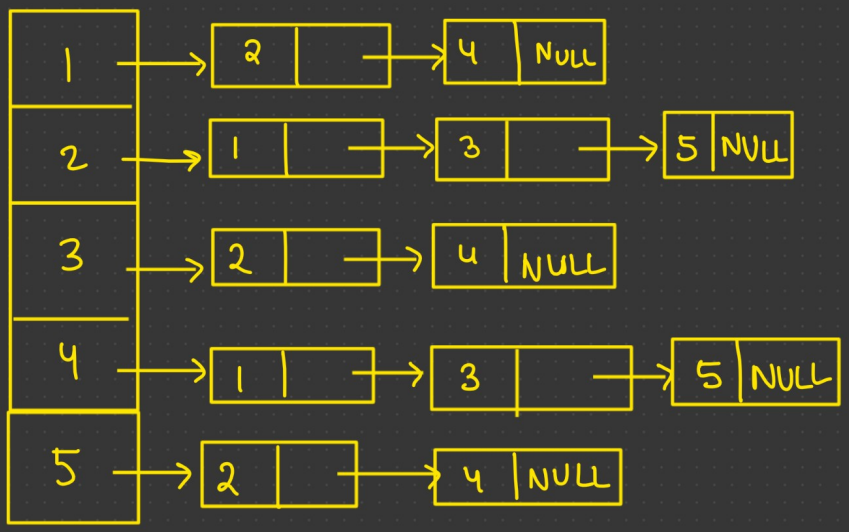
	1	2	3	4
1	0	2	0	0
2	2	0	6	5
3	0	6	0	4
4	0	5	4	0

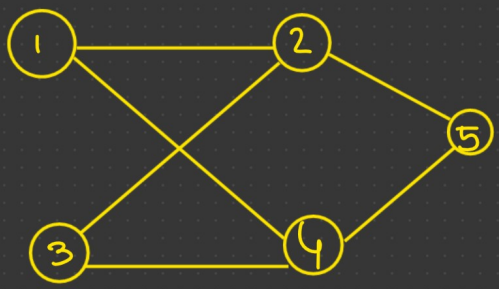


Adjacency List

It is represented as an array of Linked List.

The index of the array represents the vertex number and each linked list represents the other vertices that form an edge with vertex.





1) `vector<int> a[];`

2) `map< int , list<int> > adj;`

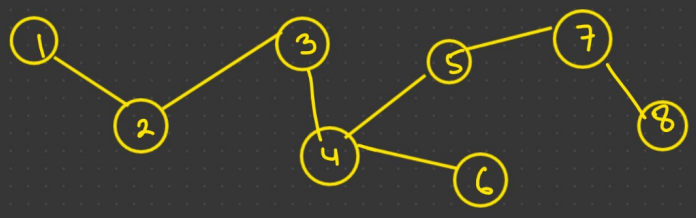
3) `vector< list<int> > adj;`

Depth First Search (DFS)

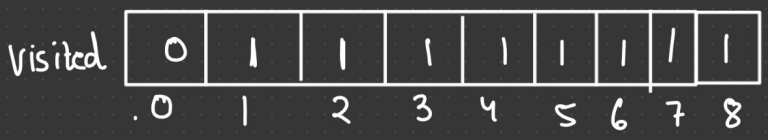
DFS is an algorithm for traversing / searching for graph.

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

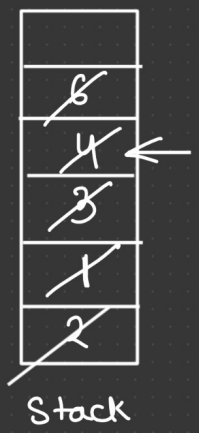
1, 2, 3, 4, 6, 5, 7, 8



DFS = Preorder Traversal



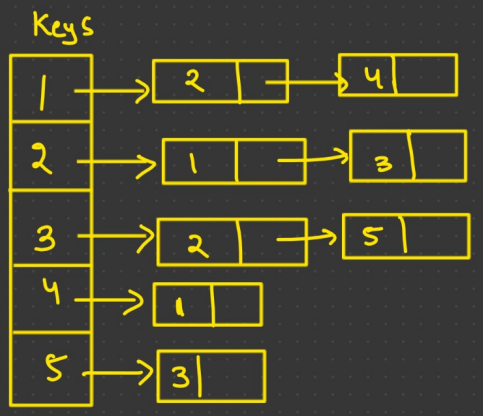
Source = 2

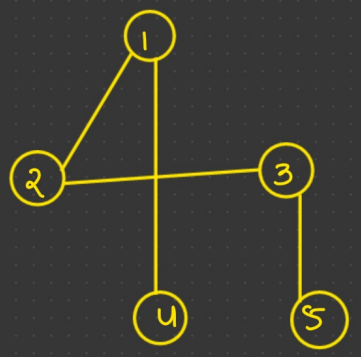


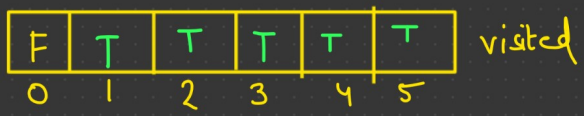
2, 1, 3, 4, 5, 7, 8, 6,

```
map<int, bool> visited;
map<int, list<int> > adj;

void DFS(int v)
 {
    visited[v] = true;
    cout<<v<<" ";

    list<int>:: iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); ++i)
   if (!visited[*i])
      DFS(*i);
 }
```

2, 1, 4, 3, 5

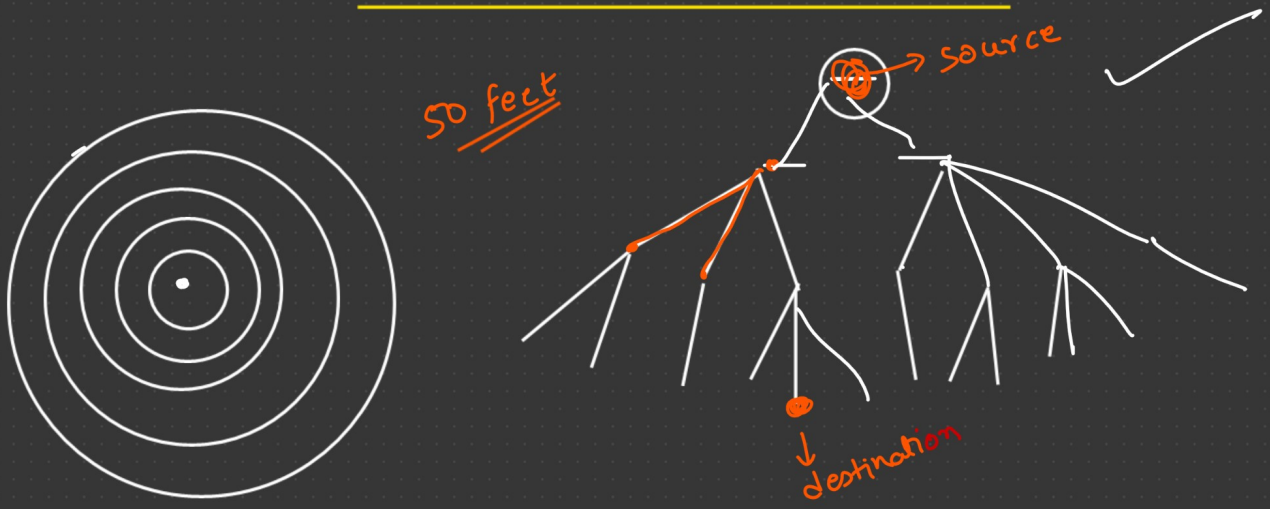






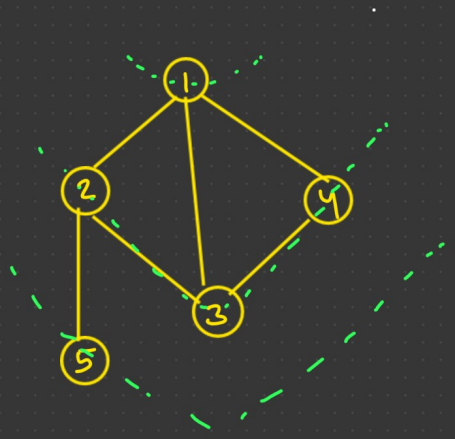
Time Complexity:- $O(V \times E)$

Breadth first Search (BFS)



	0	1	2	3	4	5	6
visited	0	1	1	1	1	1	0

Queue ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~



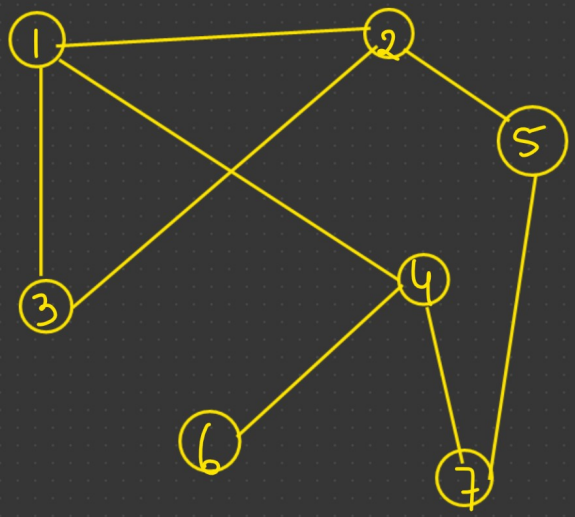
1, 2, 3, 4, 5

Find its BFS traversal :-

visited

0	1	2	3	4	5	6	7
0	1	1	1	1	1	1	1

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



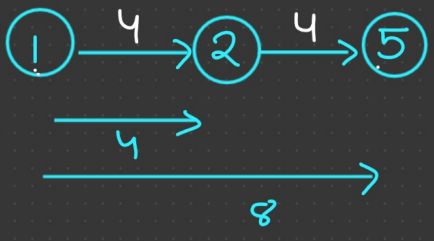
queue

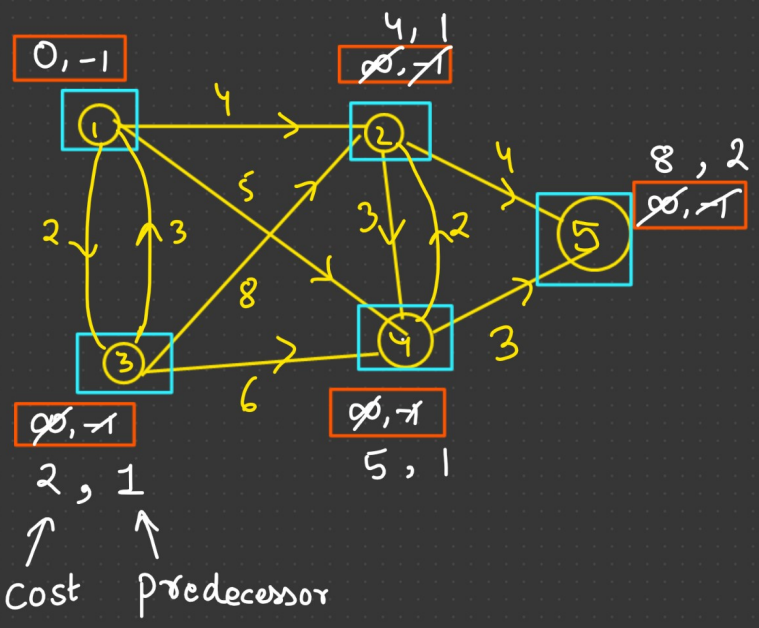
1, 2, 3, 4, 5, 6, 7

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

Dijkstra's Algorithm

- Single Source
- Shortest Path / Distance





	Cost	Pred
2	4	1
3	2	1
4	5	1
5	8	2

Table of (1)