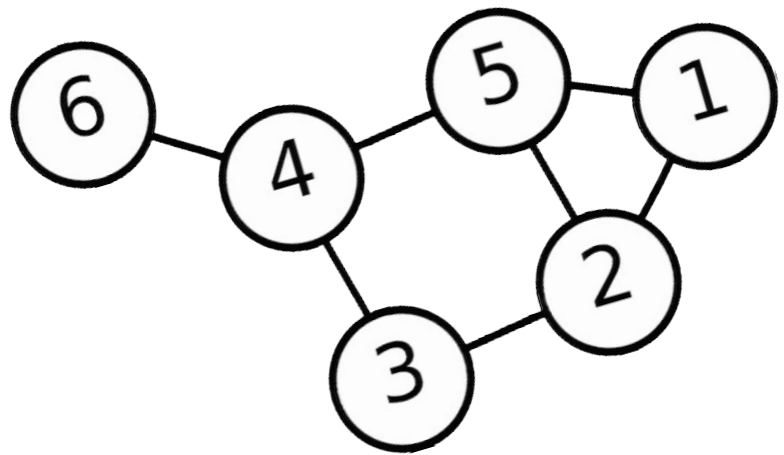
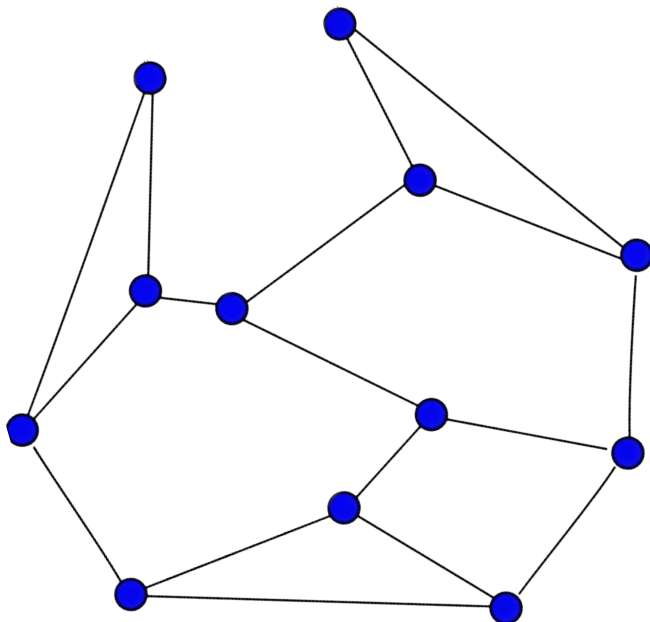




Graphs (Part-1)

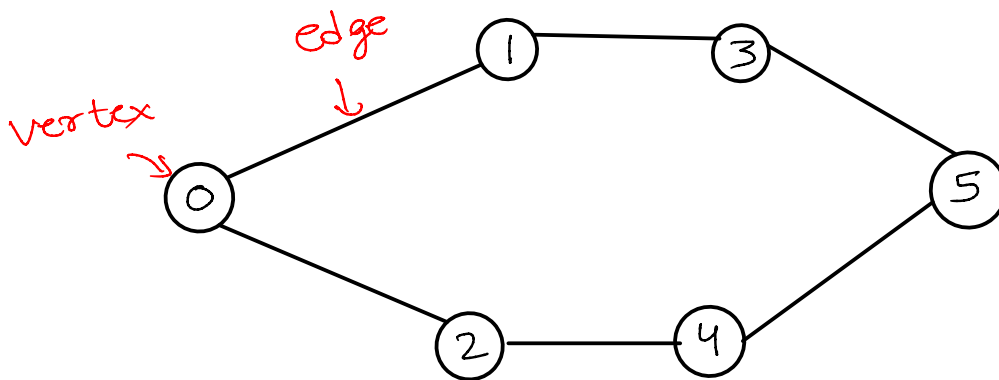
- By Kapil Yadav



1. BFS using adjacency matrix
2. BFS using adjacency list
3. DFS using adjacency matrix
4. DFS using adjacency list

Graph

A Graph $G(V,E)$ is a set of Vertices and Edges.



$$V = \{0, 1, 2, 3, 4, 5\}$$

$$E = \{\{0,1\}, \{0,2\}, \{1,3\}, \{2,4\}, \{3,5\}, \{4,5\}\}$$

→ Two vertices are said to be adjacent, if there is an edge between them.

Order of Graph:-

→ The number of vertices in a graph represents order of graph.

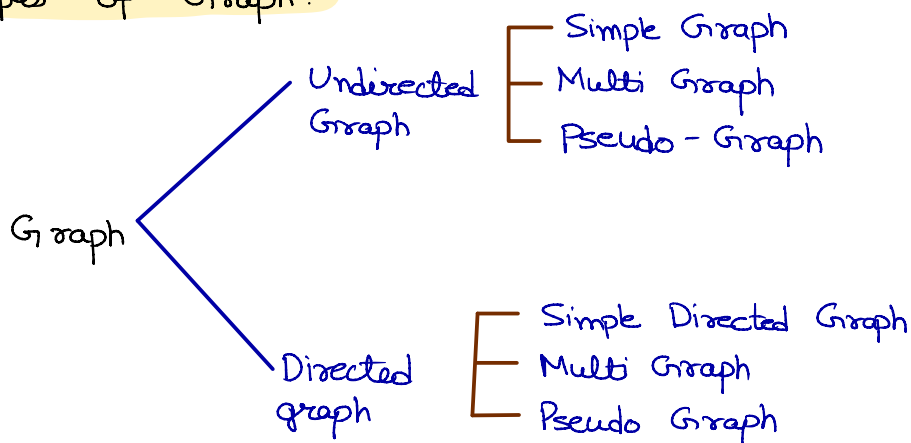
$$O(G) = 6$$

Size of Graph:-

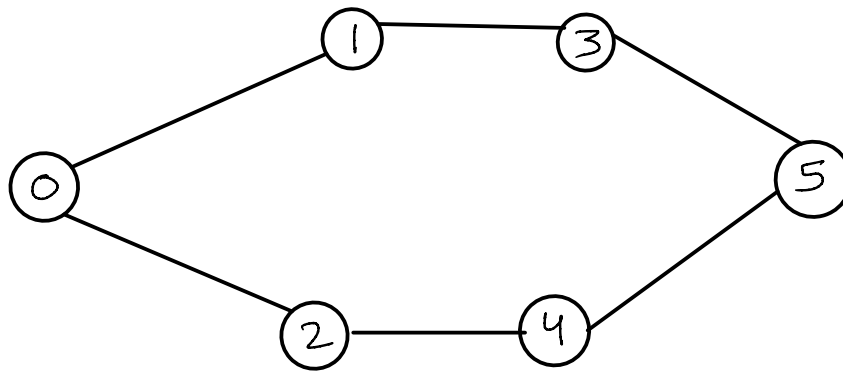
→ No. of edges in a graph

$$\text{Size}(G) = e = 6$$

Types of Graph:-

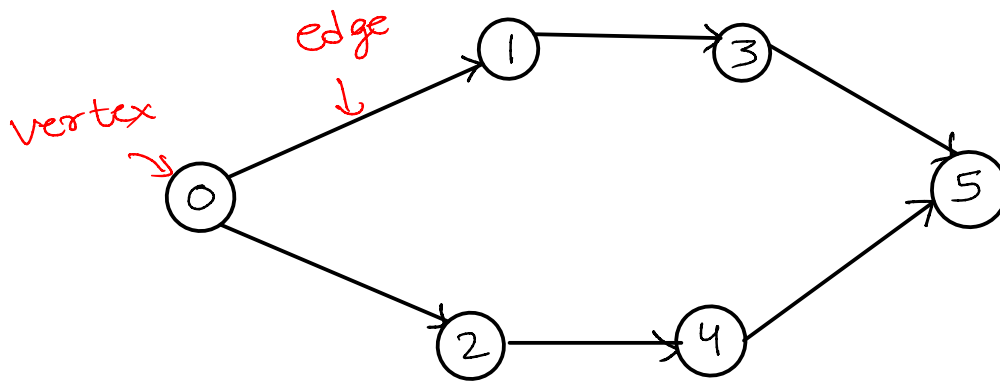


1.) Undirected Graph:- Edges in Undirected graph are Bidirectional.



Undirected graph

2. Directed Graph:- Graph having directed edges.



Simple Graph:-

- No self loops.
- No Multiple edges.

Multi Graph:-

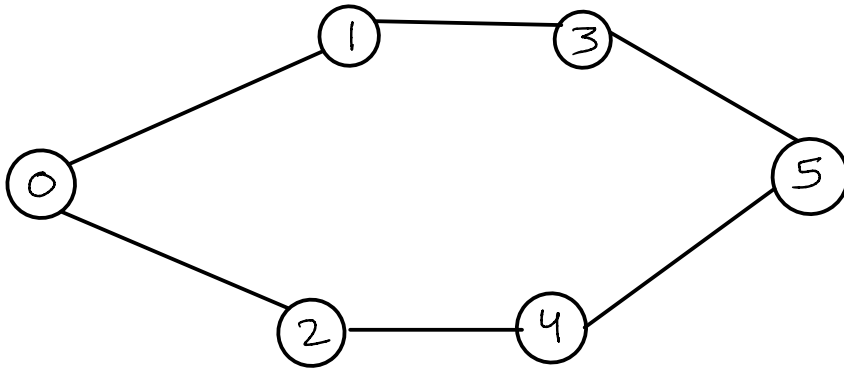
- Multiple edges are allowed.
- Self loops are not allowed.

pseudo Graph:-

- Self loops are allowed
- Multiple edges are allowed.

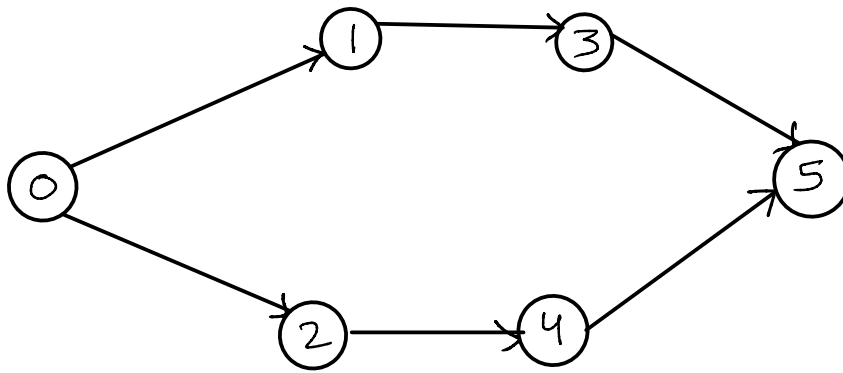
Degree of a vertex:-

- Number of vertices adjacent to a vertex V .



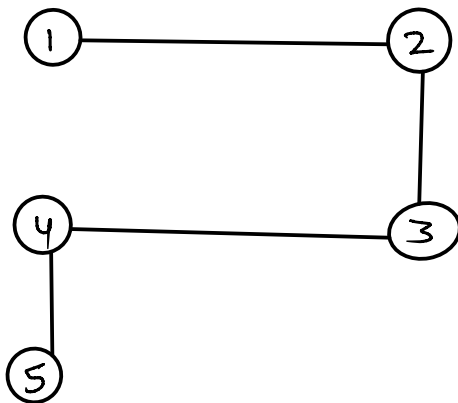
$$\deg(2) = 2$$

→ In case of Directed graph, each vertex has an indegree and an outdegree.



$$\begin{aligned} \text{Indegree}(2) &= 1 \\ \text{Outdegree}(2) &= 1 \end{aligned}$$

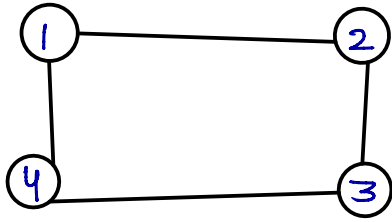
Path :- Path is a sequence of alternating vertices and edges such that the edge connects each successive vertex.



Path = 1 2 3 4 5

→ Edge and Vertex should not be repeat in a path.

Cycle :- Cycle is a path that starts and ends at the same vertex.



1 2 3 4 1
3 4 1 2 3
⋮
4 1 2 3 4

→ Tree is a connected graph with no cycles.

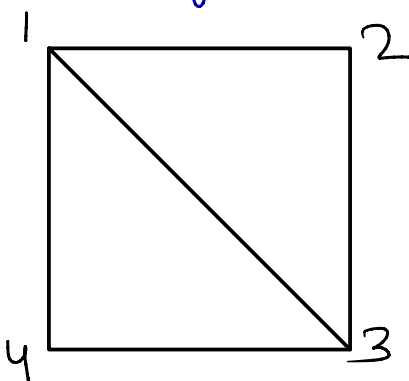
Graph Representation:-

1. Adjacency Matrix
2. Adjacency list
3. Incidence Matrix (when edges are labelled)

1. Adjacency Matrix:- A graph G with V vertices, can be represent using $V \times V$ matrix

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$$

→ In Undirected graph, the value $a_{ij} = a_{ji}$, and the adjacency matrix is symmetric.



$$\begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} & \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{matrix}$$

$$SC = O(V \times V)$$

```

1 //Adjacency Matrix
2 int main() {
3     int V,E;
4     cin>>V;
5     cin>>E;
6     //graph having Vertices from 0 to n-1
7     //if vertices are from 1 to n,take n+1,n+1 size adjamtrix
8     vector<vector<int>> adjmat(V, vector<int> (V,0));
9     int i=0;
10
11     while(i<E){
12         int from,to;
13         cin>>from>>to;
14         adjmat[from][to] = 1;
15         adjmat[to][from] = 1;
16         i++;
17     }

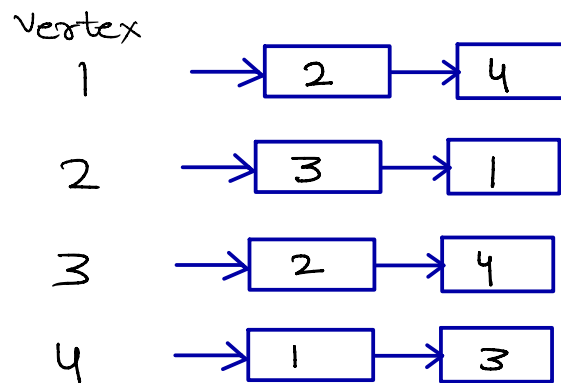
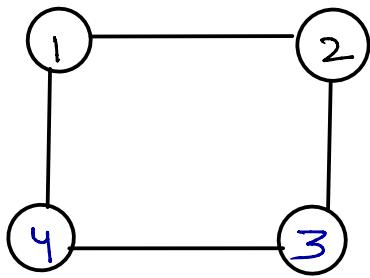
```

Adjacency Matrix of Undirected Graph

→ for Directed graph, remove line 15.

2. Adjacency list:-

→ we use an array of vector to represent the graph.



Adjacency list

$$\text{Space} = O(2E)$$

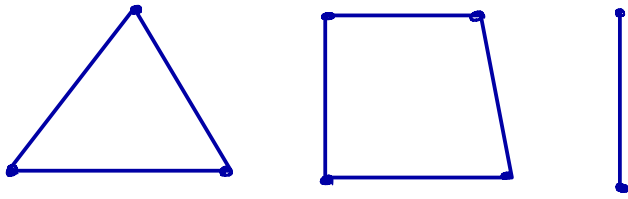
```
1 //Adjacency List
2 int main() {
3     int V,E;
4     cin>>V;
5     cin>>E;
6     //graph having Vertices from 0 to n-1
7     vector<int> adjlist[V];
8     int i=0;
9     while(i<E){
10         int from,to;
11         cin>>from>>to;
12         adjlist[from].push_back(to);
13         adjlist[to].push_back(from);
14         i++;
15     }
```

Adjacency list of Undirected Graph

→ for Directed graph, remove line 13.

→ Space Complexity for Directed graph = $O(E)$.

Connected Components :- No. of connected subgraph in a given graph.



→ Number of components in connected graph = 1

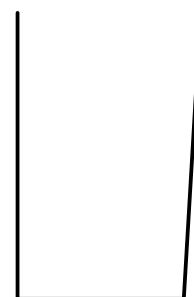
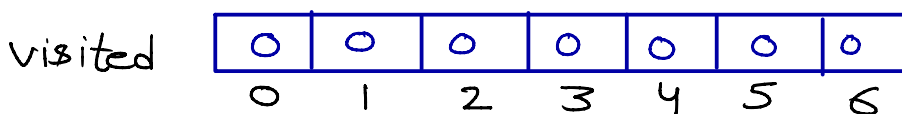
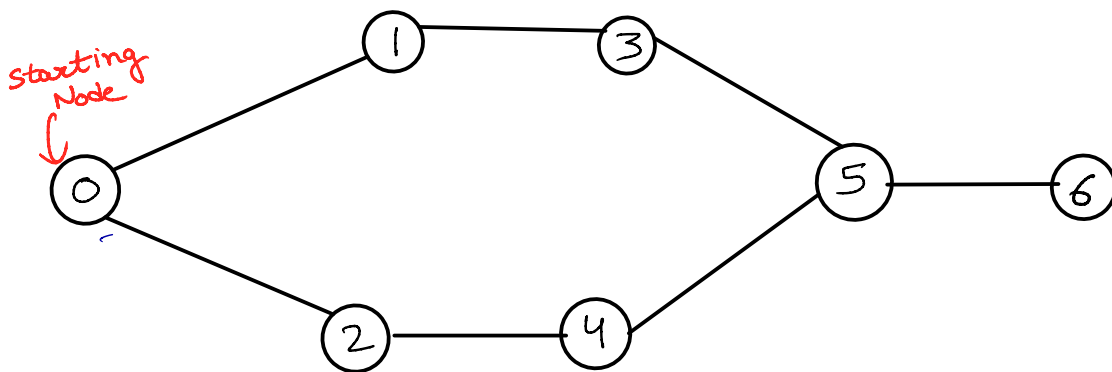
Breadth-First Search: BFS is one of the traversal techniques.

- There can be multiple connected components, so we need to have visited array.
- Each node will be either visited or not visited.
- BFS traverse the graph layerwise, so we use Queue data structure for traversal.

Algorithm:-

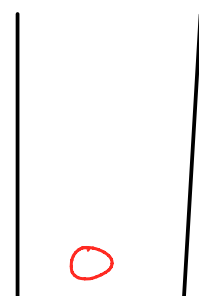
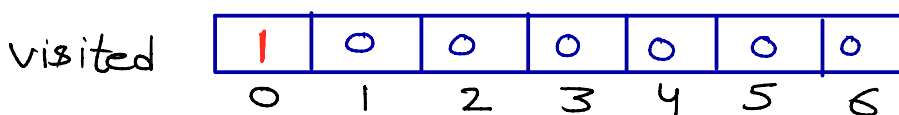
- Take a queue and insert the starting node.
- Declare a visited array and initialize it with 0.
- Mark the starting vertex as visited.
- Follow the steps, until the queue become empty.
 - Remove front vertex from queue.
 - Insert all the unvisited neighbours of the vertex into the queue and mark them visited.

Example 1:



queue

1. Push 0 in queue



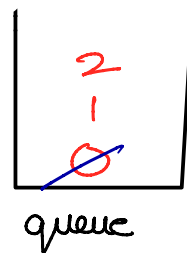
queue

2. While queue does not get empty,

Pop 0, Push 1, Push 2

visited

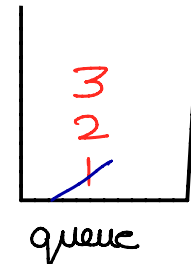
1	1	1	0	0	0	0
0	1	2	3	4	5	6



3.) Pop 1, Push 3

visited

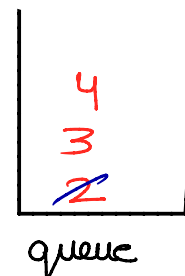
1	1	1	1	0	0	0
0	1	2	3	4	5	6



4.) Pop 2, Push 4

visited

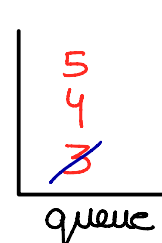
1	1	1	1	1	0	0
0	1	2	3	4	5	6



5.) Pop 3, Push 5

visited

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



6.) Pop 4

visited

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



7.) Pop 5, Push 6

visited

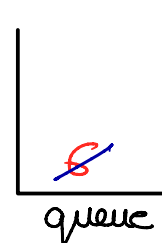
1	1	1	1	1	1	1
0	1	2	3	4	5	6



8.) Pop 6

visited

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



ans
↓
{0,1,2,3,
4,5,6}

BFS using Adjacency Matrix (Undirected Graph)

```
1 #include <iostream>
2 #include<vector>
3 #include<queue>
4 using namespace std;
5
6 //BFS using Adjacency Matrix
7 vector<int> bfs(vector<vector<int>>& adjmat){
8     int V = adjmat.size();
9     vector<int> vis(adjmat.size(),0);
10    vector<int> ans;
11    queue<int> q;
12    q.push(0);
13    vis[0] = 1;
14    while(!q.empty()){
15        int currele = q.front();
16        q.pop();
17        ans.push_back(currele);
18        for(int i=0;i<V;i++){
19            if(vis[i]==0 && adjmat[currele][i]==1){
20                vis[i]=1;
21                q.push(i);
22            }
23        }
24    }
25    return ans;
26 }
27
28 void printmatrix(vector<vector<int>>& adjmat){
29     int V = adjmat.size();
30     for(int i=0;i<V;i++){
31         for(int j=0;j<V;j++){
32             cout<<adjmat[i][j]<<" ";
33         }
34         cout<<endl;
35     }
36 }
```

$$TC = O(V \times V)$$

$$SC = O(V \times V) + O(V)$$

Adj matrix

to store answer

```

1 int main() {
2     int V,E;
3     cin>>V;
4     cin>>E;
5     //graph having Vertices from 0 to n-1
6     //if vertices are from 1 to n,take n+1,n+1 size adjamtrix
7     vector<vector<int>> adjmat(V, vector<int> (V,0));
8     int i=0;
9
10    while(i<E){
11        int from,to;
12        cin>>from>>to;
13        adjmat[from][to] = 1;
14        adjmat[to][from] = 1;
15        i++;
16    }
17    //printmatrix(adjmat);
18    vector<int> ans = bfs(adjmat);
19
20    for(int i=0;i<ans.size();i++){
21        cout<<ans[i]<<" ";
22    }
23
24    return 0;
25 }

```

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

BFS Using Adjacency List (Undirected Graph)

```
1 #include <iostream>
2 #include<vector>
3 using namespace std;
4
5 vector<int> bfs(int V, vector<int> adj[]) {
6     vector<int> ans;
7     vector<bool> vis(V,0);
8     queue<int> q;
9     q.push(0);
10    vis[0] = 1;
11
12    while(!q.empty()){
13        int curtele = q.front();
14        ans.push_back(curtele);
15        q.pop();
16        for(int i=0;i<adj[curtele].size();i++){
17            if(!vis[adj[curtele][i]]){
18                vis[adj[curtele][i]] = 1;
19                q.push(adj[curtele][i]);
20            }
21        }
22    }
23    return ans;
24 }
25
26 void printlist(vector<int> adjlist[],int V){
27     for(int i=0;i<V;i++){
28         cout<<i<<"-> ";
29         for(int j=0;j<adjlist[i].size();j++){
30             cout<<adjlist[i][j]<<" ";
31         }
32         cout<<endl;
33     }
34 }
```

$$TC = O(V) + O(2E)$$

↑
No. of
vertices

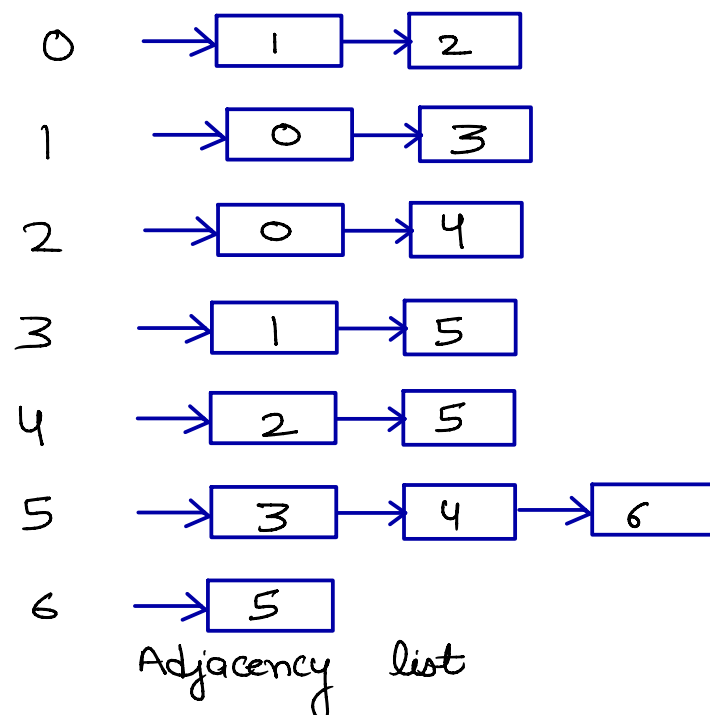
↑
total degrees

$$\begin{aligned} SC &= O(V + E) \\ &= O(V^2) \quad \{ \text{In worst case} \} \end{aligned}$$

```

1 int main() {
2     int V,E;
3     cin>>V;
4     cin>>E;
5     //graph having Vertices from 0 to n-1
6     vector<int> adjlist[V];
7     int i=0;
8     while(i<E){
9         int from,to;
10        cin>>from>>to;
11        adjlist[from].push_back(to);
12        adjlist[to].push_back(from);
13        i++;
14    }
15    printlist(adjlist,V);
16    vector<int> ans = bfs(V,adjlist);
17
18    cout<<"BFS USING ADJACENCY LIST : ";
19    for(int i=0;i<ans.size();i++){
20        cout<<ans[i]<<" ";
21    }
22
23    return 0;
24 }

```



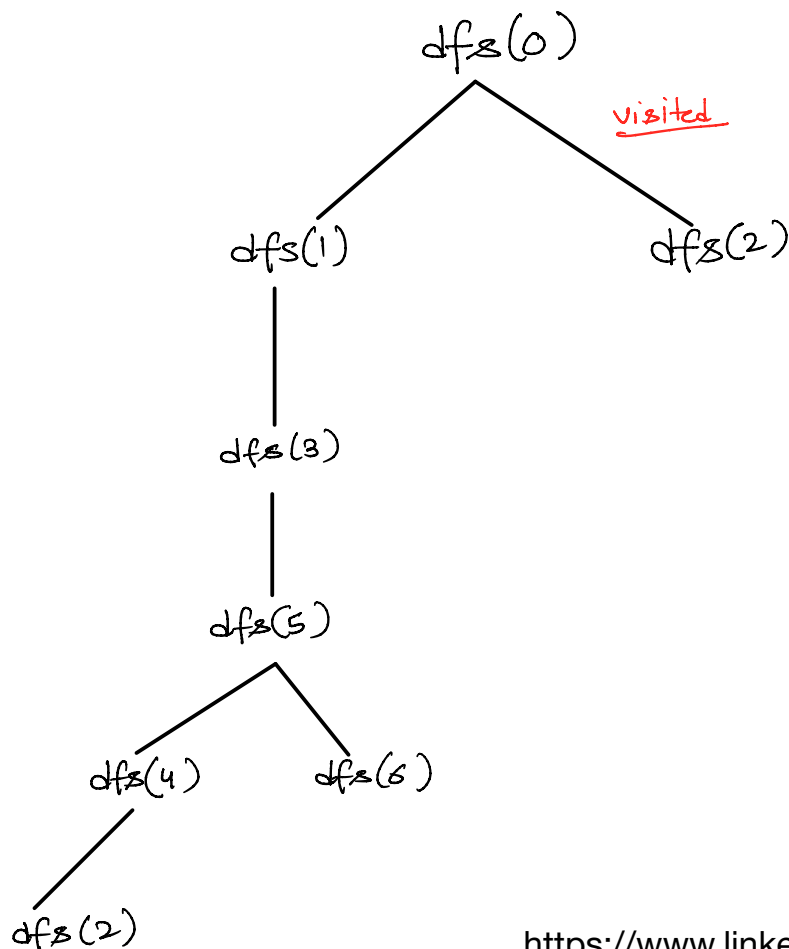
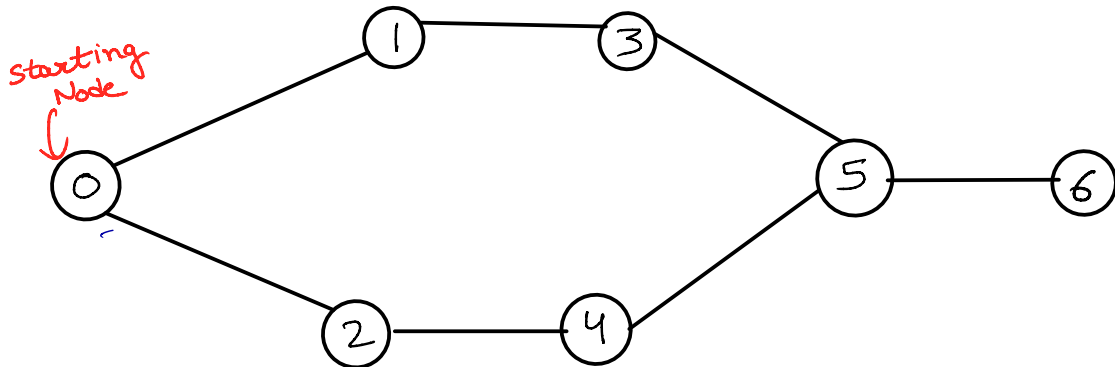
Depth First Search (DFS):

Depth first search is a recursive algorithm.
→ In DFS, we select a starting node, and explore it as far as possible.

Algorithm :-

- Choose a starting node and do recursion.
- Make current node as visited and add current node in our output list (vector/array).
- Traverse all the adjacent and unvisited nodes.

Ex -



<https://www.linkedin.com/in/kapilyadav22/>

→ We can choose neighbour nodes in any order. But each node will traverse only one.

→ There can be Multiple DFS traversals for a graph.

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

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

```

1 //DFS Using Adjacency Matrix
2 void dfs(int currver,vector<vector<int>>& adjmat, vector<bool>&
  vis,vector<int>& ans){
3     ans.push_back(currver);
4     vis[currver]=1;
5     int V = adjmat.size();
6     for(int i=0;i<V;i++){
7         if(!vis[i] && adjmat[currver][i]==1){
8             dfs(i,adjmat,vis,ans);
9         }
10    }
11 }
12
13 int main() {
14     int V,E;
15     cin>>V;
16     cin>>E;
17     //graph having Vertices from 0 to n-1
18     //if vertices are from 1 to n,take n+1,n+1 size adjamtrix
19     vector<vector<int>> adjmat(V, vector<int> (V,0));
20     int i=0;
21
22     while(i<E){
23         int from,to;
24         cin>>from>>to;
25         adjmat[from][to] = 1;
26         adjmat[to][from] = 1;
27         i++;
28     }
29
30     vector<int> ans;
31     vector<bool> vis(V,0);
32     dfs(0,adjmat,vis,ans);
33     cout<<" DFS USING ADJACENCY MATRIX : ";
34     for(int i=0;i<ans.size();i++){
35         cout<<ans[i]<<" ";
36     }
37     return 0;
38 }

```



```
1 //DFS Using Adjacency list
2
3 void dfs(int currver,vector<int> adjlist[], vector<bool>&
  vis,vector<int>& ans){
4     ans.push_back(currver);
5     vis[currver]=1;
6     for(auto i : adjlist[currver]){
7         if(!vis[i]){
8             dfs(i,adjlist,vis,ans);
9         }
10    }
11 }
12
13 int main() {
14     int V,E;
15     cin>>V;
16     cin>>E;
17     //graph having Vertices from 0 to n-1
18     vector<int> adjlist[V];
19     int i=0;
20     while(i<E){
21         int from,to;
22         cin>>from>>to;
23         adjlist[from].push_back(to);
24         adjlist[to].push_back(from);
25         i++;
26     }
27     vector<bool> vis(V,0);
28     vector<int> ans;
29     dfs(0,adjlist,vis,ans);
30
31     for(int i=0;i<ans.size();i++){
32         cout<<ans[i]<<" ";
33     }
34     return 0;
35 }
```