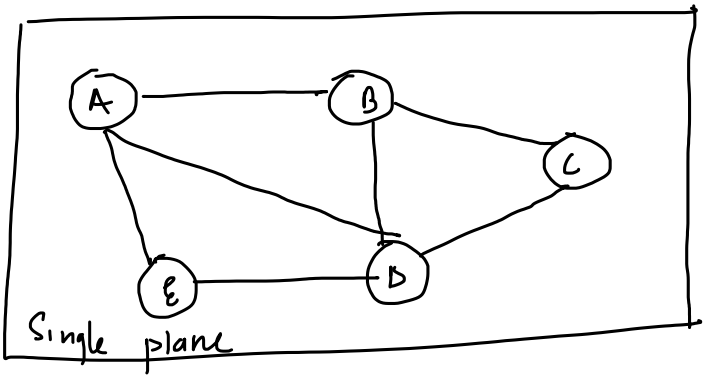
Graph -> A graph is collection of vertex connected through edges and all the vertex belongs in same plane

Consider



A Graph is represented as $G = (V, E)$

Here V = set of vertices = $\{A, B, C, D, E\}$

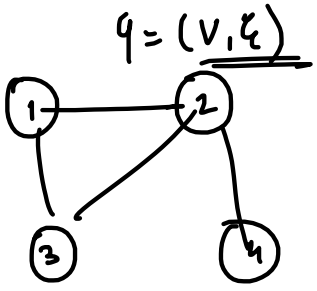
E = set of Edges = $\{(A,B), (B,C), (A,E), (A,D), (B,D), (C,D), (D,E)\}$

Since the edges do not have direction associated with it we call it as (Undirected / Bidirectional) graph

Consider

UnDirected Graph (Bidirectional)

$G = (V, E)$



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

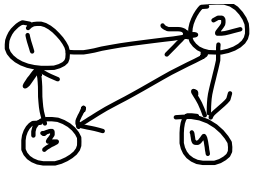
$E = \{(1,2), (1,3), (2,3), (2,4)\}$

↑ parentheses means undirected graph

Directed graph -> Here direction is associated with every edge

$G = \langle V, E \rangle$

} < > Angular Brackets ⇒ Directed graph



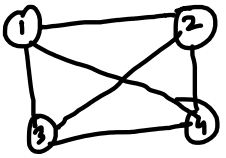
$G = \langle V, E \rangle$; Brackets ⇒ Directed graph

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

$E = \{\langle 1,2\rangle, \langle 2,3\rangle, \langle 2,4\rangle, \langle 3,1\rangle\}$

Note → In UnDirected Graph →

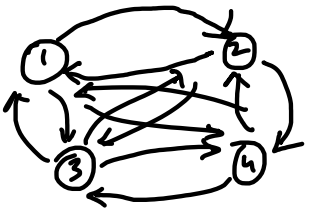
if n = no of vertex



max no of Edge = $\frac{n(n-1)}{2}$ Here $\frac{4\times3}{2} = 6$ //

In Directed Graph

if n = no of vertex

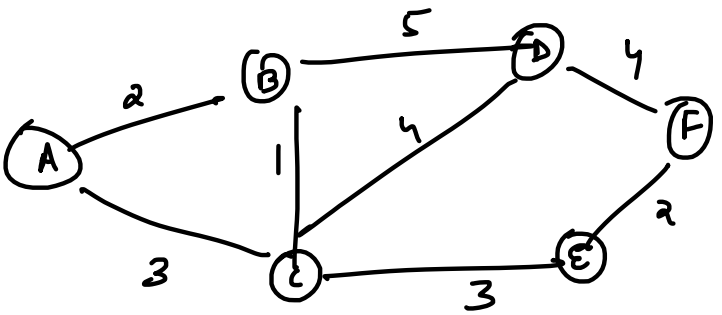


max no Edge = $2 \times \frac{(n)(n-1)}{2} = n(n-1)$

In above Graph there are no weight/cost associated with Edge. We call all the above graph as Unweighted Graph.

Weighted Graph → If the edge of the graph has weight associated with it it is known as Weighted Graph.

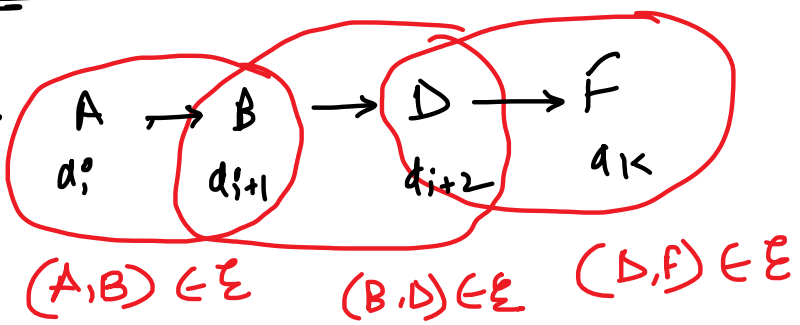
Ex



Path → Path from Vertex a_i^0 to a_k is set of Vertex $a_{i+1}^0, a_{i+2}^0 \ldots a_k$ such that

$(a_i^0, a_{i+1}^0) \in E$

Ex path from A → F →



Connected Node → Two nodes/vertex a_i^0 & a_j are said to be Connected if there is path from a_i^0 to a_j

Connected graph → If all the nodes/vertex of graph are

Connected then the graph is said to be Connected Graph

* In above Graph from any vertex there is path / connection to any other vertex
∴ above graph is Connected Graph

Adjacent Node ⇒ Two nodes / vertex are said to be adjacent if there is an edge bet^n them

In above graph
Adjacent vertex of A → Vertex B & Vertex C

Vimp? Memory Representation of graph

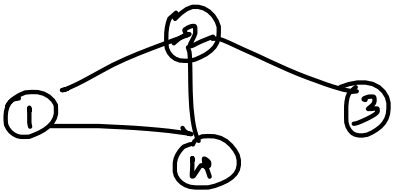
There are 2 ways to represent a graph in Computer Memory

(1) **Adjacency matrix** → It is n×n matrix (n = no of vertex)

Here adj[i][j] = 1 if there is an edge bet^n i & j vertex

= 0 if there is no edge bet^n vertex i & j

ex



} Unweighted, Undirected graph

adj[4][4] =

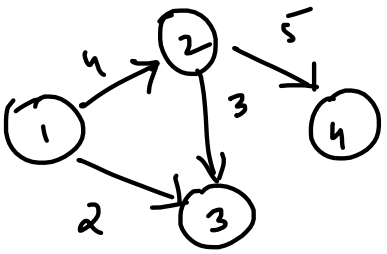
	1	2	3	4
1	0	1	0	1
2	1	0	1	1
3	0	1	0	0
4	1	1	0	0

For Weighted Directed Graph →

adj[i][j] = cost/weight of edge from i to j

= 0 if no edge.

Consider



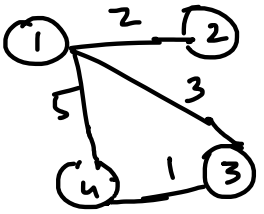
adj[4][4] =

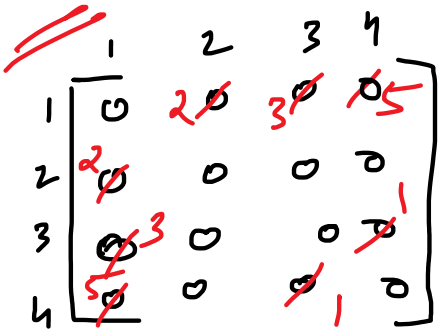
	1	2	3	4
1	0	4	2	0
2	0	0	3	5
3	0	0	0	0
4	0	0	0	0

To read a graph // UnDirected Graph

```
void readgraph()
{ int g[10][10] = {0};
  int e, v, a, b, c;
printf("Enter no of vertex\n");
scanf("%d", &v);
printf("Enter no of edges\n");
scanf("%d", &e)
// Read edge info
for(i=1; i<=e; i++)
{ printf("Enter vertex associated with edge\n");
  scanf("%d %d", &a, &b);
  printf("Enter the cost of edge\n");
  scanf("%d", &c);
  g[a][b] = g[b][a] = c;
} // for
} // fn
```

v=4
e=4





i=1
a=1, b=2, c=2
g[1][2] = g[2][1] = 2

i=2
a=1, b=4, c=5
g[1][4] = g[4][1] = 5

i=3
a=4, b=3 c=1
g[4][3] = g[3][4] = 1

i=4
a=1, b=3, c=3
g[1][3] = g[3][1] = 3

Note if weighted

g[a][b] = g[b][a] = 1

if Undirected

g[a][b] = c OR

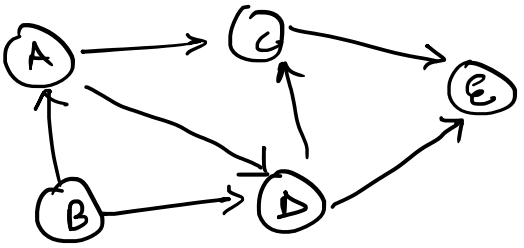
g[a][b] = 1

Topological Sorting

* It is for Directed Acyclic graph (DAG).
* Here the sorting generates linear ordering of vertices such that for every directed edge $\langle u, v \rangle$, vertex u must come before vertex v in the ordering.

B → A → D → C → E

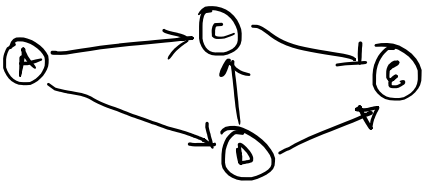
Consider



Find Indegree of Every Vertex →

A	B	C	D	E
1	0	2	2	2

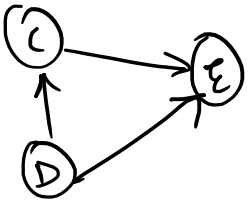
1. As indegree of node B is 0, remove node B from the Graph (B is Visited)



Now Indegree of Every node

A	C	D	E
0	2	1	2

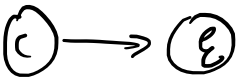
Node A is with Indegree 0 ,So remove node A from graph & Visited.



Now Indegree of Every node

C	D	E
1	0	2

Node D is with Indegree 0, So remove node D and visited.



Indegree of Node

C	E
0	1

Indegree of Node C is 0, So remove C & Visited.

Indegree

E
0

∴ Remove node E from Graph & visited

Order: B → A → D → C → E

* Topological Sorting is not possible if the graph is not DAG.

Application →

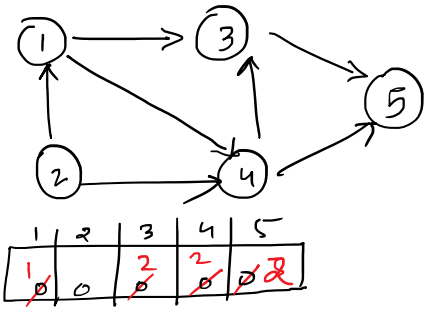
(1) Critical Path Analysis in Graph.

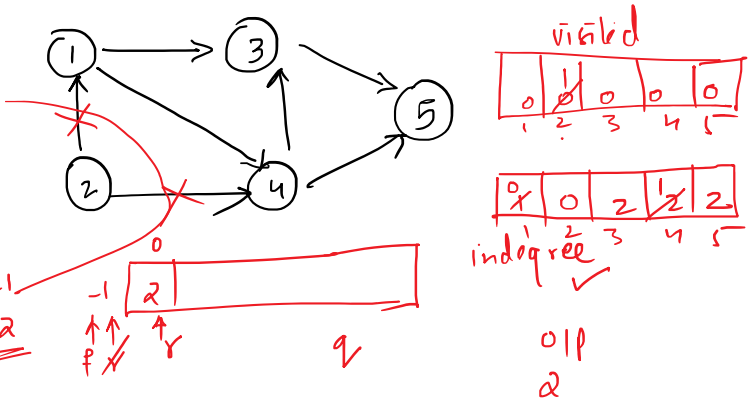
② Used in RAG (Resource Allocation Graph) for detection of deadlock

③ Find cycle in graph.

C Program for Topological Sorting

```
#include<stdio.h>
int adj[10][10]={0},visited[10]={0},n;

void topologicalsort()
{
    int i,j,indeg[10]={0},nd,q[10],f=-1,r=-1;

    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            if(adj[j][i]==1)
            indeg[i]++;
        }
    }

    while(1)
    {
        for(i=1;i<=n;i++)
        {
            if(indeg[i]==0 && visited[i]==0)
            {
                visited[i]=1;
                q[++r]=i;
            }
        }
        if(f==r)
        break;
        nd=q[++f];
        printf("%d ",nd);
        for(i=1;i<=n;i++)
        {
            if(adj[nd][i]==1)
            indeg[i]--;
        }
    }
}

int main()
{
        int e,i,v1,v2;
```





```
    printf("enter nos of nodes\n");
    scanf("%d",&n);
    printf("enter nos of edges\n");
    scanf("%d",&e);
    printf("enter edge details\n");
    for(i=1;i<=e;i++)
    {
        printf(" enter edge\n");
        scanf("%d %d",&v1,&v2);
        adj[v1][v2]=1;
    }
    topologicalsort();
    return 0;
}
```

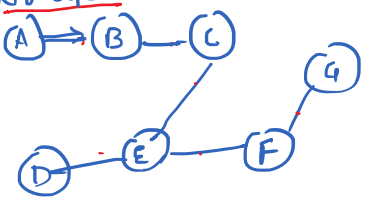
Graph Traversal → Visiting every vertex of graph exactly once.

Imp

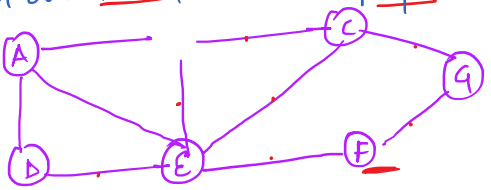
Types
- Depth First Search (DFS) → using Stack data structure → Recursion.
- Breadth First Search (BFS) → Uses Queue data structure.

① Depth First Search → DFS produces Spanning Tree for given graph
- → Uses Stack Data structure
- → max size of stack can be n = no of vertex in graph.
- → No unique DFS
- → more than one DFS possible for given graph. ⇒

Spanning Tree →
Subgraph of Graph →
1. All vertex of original graph
2. Not necessary all Edges
3. No cycle & all are connected



Consider a graph → Undirected



Original graph

Perform DFS Traversal →

A	B	C	D	E	F	G
0	0	0	0	0	0	0

visited. 0 ⇒ Unvisited, 1 ⇒ Visited

(Subgraph) → Spanning (Tree)

Stack

* You may start with any vertex → Let us start with vertex A

① We us start with vertex A
push vertex A in stack & mark it visited

② push unvisited adjacent node of A (anyone) in Stack and mark it visited.
Let say B

③ push any one unvisited adjacent node of B in Stack & mark it visited
Let say C

④ push any one unvisited adjacent vertex of C & mark it visited.
Let say E

⑤ push any one unvisited adjacent vertex of E & mark it visited.
Let say D

⑥ push any one unvisited adjacent vertex of D and mark it visited.
No unvisited adjacent vertex of D is Available
so pop D from Stack. ✓

⑦ Now at stacktop we have E. ✓

DFS above >

DFS above >

A → B → C → E → D → F → G.

(7) Now at stacktop we have E. ✓

(8) Push any one unvisited adjacent vertex of E and mark it visited, it is F

(9) Push any one unvisited adjacent vertex of F and mark it visited, it is G.

(10) At stacktop we have G

(11) no unvisited adjacent of G is available so pop G.

(12) Similarly all the stack elements are popped and at the end stack is Empty.

DFS steps ->

✓ (1) Define a stack of size as total no of vertex.

✓ (2) Create an array visited of size same as no of vertex and initialize with 0.

(3) Select any vertex as starting point, mark it visited and push the vertex in stack.

(4) Visit any one of unvisited adjacent vertex of top of stack, mark it visited and push it in stack.

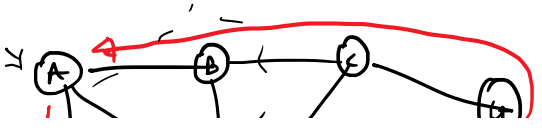
(5) Repeat step 4 until there is no new vertex to be visited for top of stack

(6) When no new vertex to visit for the top of stack then pop the vertex at top of stack.

(6) Repeat step 4, 5 & 6 until stack becomes Empty.

(7) When stack is Empty then produce final spanning Tree by removing unused edges from graph.

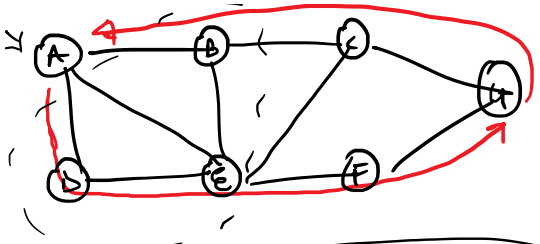
Consider



dfs

A - D - E - F - G - C - B

Consider

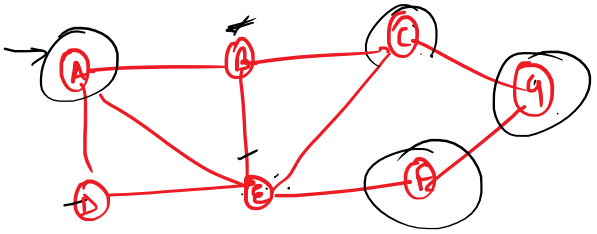


A - D - E - F - G - C - B

depth First → ① Select any one adjacent vertex and visit it

② Repeat step 1

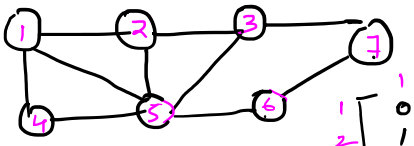
breadth First → ① Visit all the adjacent vertex of a given vertex and then move to other vertex.

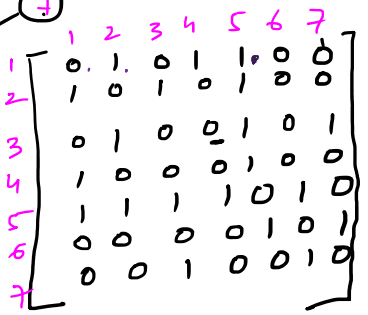


BFS start from A

A B D E C G F

F^n for DFS traversal →

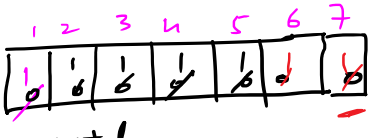




adj[][] ⇒

```
void dfs (int node)
{ int i;
  visited[node]=1;
  printf("%d\t", node);
for (i=1; i<=n ; i++) // for all other vertex
{   // edge bet^n node & i vertex
 if ( adj[node][i] ==1   && visited[i] ==0 )
   // i is adjacent to node   // i^th vertex not visited
                              // i is unvisited
     dfs(i);
}
}// f^n.
```

at node=1, i=5



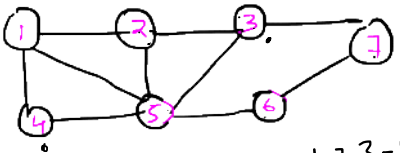
visited

o/p 1_2_3_5_4_6_7

Now for above graph →

Start from vertex 1 → dfs(1) node=1

visited



o/p 1-2-3-5-4-6

visited[1]=1 ✓ Print 1

i=1 ✗ i=2 → node=2, dfs(2), visited[2]=1, Print 2

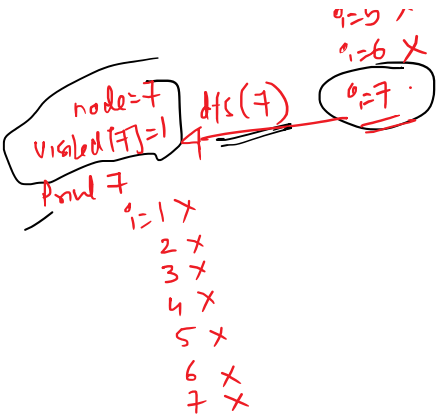
i=1 ✗ i=2 ✗ i=3 → node=3, visited[3]=1, Print 3, dfs(3)

i=1 ✗ i=2 ✗ i=3 ✗ i=4 ✗ i=5 → node=5, visited[5]=1, Print 5, dfs(5)

i=1 ✗ i=2 ✗ i=3 ✗ i=4 → node=4, visited[4]=1 ✓, Print 4, dfs(4)

i=1 ✗ i=2 ✗ i=3 ✗ i=4 ✗ i=5 ✗ i=6 → dfs(6) → node=6, visited[6]=1, Print 6

i=1 ✗ i=2 ✗ i=3 ✗ i=4 ✗ i=5 ✗ i=6 ✗ i=7 ✗

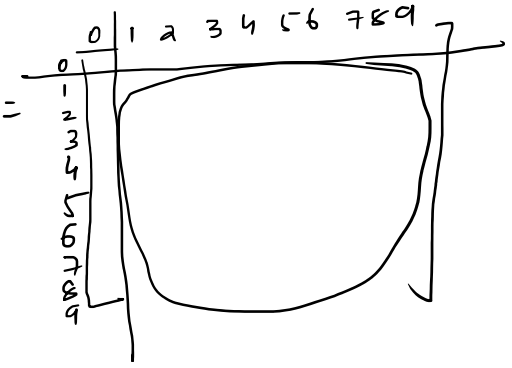

node=7
visited[7]=1
dfs(7)
i=7
Print 7
i=1
2
3
4
5
6
7

C program to Implement DFS for given graph.

```
#include<stdio.h>
int adj[10][10]={0};
int visited[10]={0};
```
Max 9 vertex is allowed.

```
void dfs(int node)
{
  int i;
  visited[node]=1;
  printf("%d\t", node);
  for(i=1; i<=n; i++)
  {
    if( adj[node][i]==1 && visited[i]==0)
      dfs(i);
  }
}
```

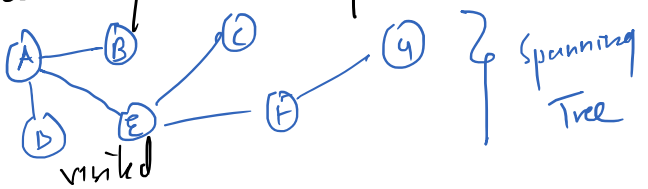
adj[10][10]=



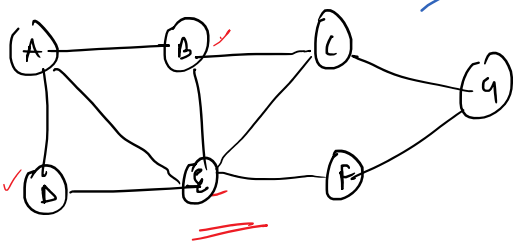

```
int main()
{
  int e, v, v1, v2, node;
  printf("Enter no of vertex\n");
  scanf("%d", &v);
  printf("Enter no of edges\n");
  scanf("%d", &e);
  printf("Enter details of Every edge\n");
  for(i=1; i<=e; i++)
  {
    printf("Enter the vertex infor for edge\n");
    scanf("%d %d", &v1, &v2);
    adj[v1][v2] = adj[v2][v1] = 1;
  }
  printf("Enter start vertex\n");
  scanf("%d", &node);
    dfs(node);
  return 0;
}//main
```

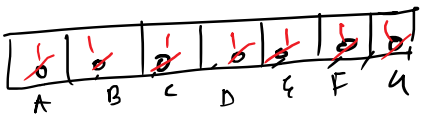
Breadth First Search → It is graph Traversal Technique
→ It visits all the adjacent vertex of node/vertex and then chks for next vertex
→ It uses Queue Data Structure
→ It generates Spanning Tree for the graph.
→ No unique BFS
→ More than one BFS possible for given graph

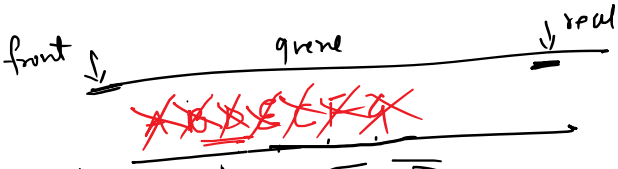
Steps →
1. Define a queue data structure with size equal to no of vertex in graph.
2. Select any vertex as start point, visit it and insert this vertex in queue
3. Visit all unvisited adjacent vertex of the vertex at queue front
4. When there is no new vertex to be visited for queue front vertex then delete the vertex.
5. Repeat step 3 & 4 untill the queue is empty.
6. Produces final Spanning Tree by removing unused edges from the graph.



Ex







(1) Start with Vertex A ✓
visited [A]=1
insert ~~push~~ A in queue

(2) At queue front we have vertex A
insert all unvisited adjacent vertex of A

(9) Now at queue front we have B
Insert all unvisited adjacent vertex of B & mark them visited
visited [C]=1 ✓

(10) Delete B

(11) Now at qufront C

(13) Insert G as unvisited adjacent of C & mark it visited

insert all unvisited adjacent vertex of A in queue & mark them visited.

for A, unvisited adjacent nodes are B, D, E

visited [B] = 1 ✓
visited [D] = 1 ✓
visited [E] = 1 ✓

& Insert B, D, E in queue.

(3) No new unvisited adjacent of A remains so delete A. ✓

visited
visited [C] = 1 ✓
Insert C in queue.

(5) Now delete B

(6) Now at queue front D
No unvisited adjacent of D

(7) Delete D

(8) Now E at queue front

(9) Insert F as unvisited adjacent of E & mark it visited.

& mark it visited

(14) Delete C

(15) Delete F

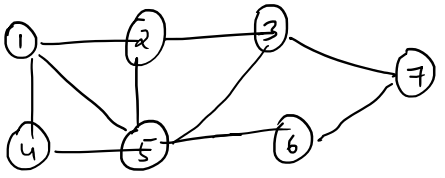
(16) Delete G.

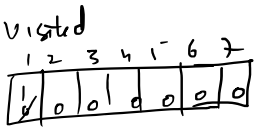
O/P See order of delete ⇒ A → B → D → E - C - F - G ✓

[Since after A we visited B → D - E i.e all adjacent vertex of A hence it is Breadth First order.]

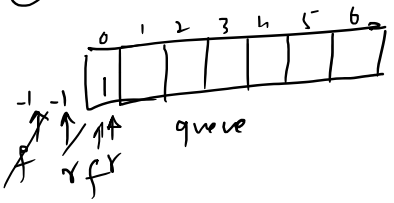
C program to implement BFS for given graph.

```
#include <stdio.h>
int adj[10][10]={0}; ⇒ adjacency matrix
int visited[10]={0}; ⇒ visited array.

void bfs(int node)
{
  int q[10], f=-1, r=-1, i, nd;
  visited[node]=1;
  q[++r]=node;
  while(f!=r)  ← queue is not empty.
  {
    nd=q[++f];
    printf("%d", nd);
    for(i=1; i<=n; i++)
    {
      if(adj[nd][i]==1 && visited[i]==0)
      {
        visited[i]=1;  ✓ mark it visited
        q[++r]=i;  ← insert in queue.
      }
    }
  }//while
}//bfs.

int main
{
  int e, v, v1, v2, node;
  printf("Enter no of vertex\n");
  scanf("%d", &v);
  printf("Enter no of edges\n");
  scanf("%d", &e);
  printf("Enter details of every edge\n");
```





bfs(1)



```
for (i=1 ; i<=e ; i++)
{   printf("Enter the vertex infor for edge\n");
    scanf("%d %d", &v1, &v2);
    adj[v1][v2] = adj[v2][v1] = 1;
}
printf("Enter start vertex\n");
scanf("%d", &node);
    bfs(node);
    return 0;
}//main
```