1) Graph traversal technique DFS (using stack)

```
#include<stdio.h>
#include<stdlib.h>

#define MAX 100

#define initial 1
#define visited 2

int n;    /* Number of nodes in the graph */
int adj[MAX][MAX]; /*Adjacency Matrix*/
int state[MAX]; /*Can be initial or visited */

void DF_Traversal();
void DFS(int v);
void create_graph();

int stack[MAX];
int top = -1;
void push(int v);
int pop();
int isEmpty_stack();
```

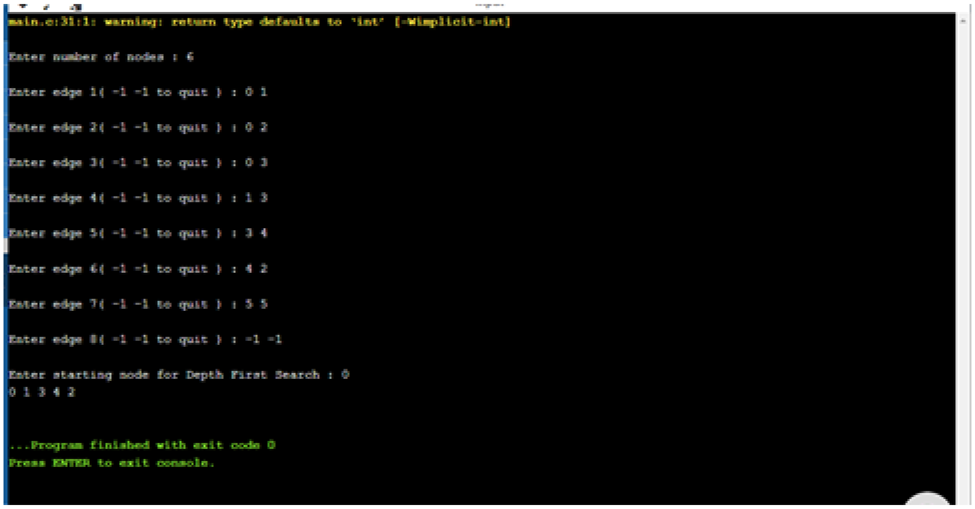
```
main()
{
    create_graph();
    DF_Traversal();
}/*End of main()*/

void DF_Traversal()
{
      int v;

      for(v=0; v<n; v++)
        state[v]=initial;

    printf("\nEnter starting node for Depth First Search : ");
    scanf("%d",&v);
      DFS(v);
    printf("\n");
}/*End of DF_Traversal( )*/

void DFS(int v)
{
      int i;
      push(v);
    while(!isEmpty_stack())
```

```
      {
        v = pop();
        if(state[v]==initial)
        {
            printf("%d ",v);
            state[v]=visited;
        }
        for(i=n-1; i>=0; i--)
        {
            if(adj[v][i]==1 && state[i]==initial)
                  push(i);
        }
      }
}/*End of DFS( )*/


void push(int v)
{
      if(top == (MAX-1))
      {
        printf("\nStack Overflow\n");
        return;
      }
    top=top+1;
    stack[top] = v;
```

```
}/*End of push()*/


int pop()

{

        int v;

        if(top == -1)

        {

                printf("\nStack Underflow\n");

           exit(1);

        }

        else

        {

           v = stack[top];

           top=top-1;

           return v;

        }

}/*End of pop()*/


int isEmpty_stack( )

{

  if(top == -1)

        return 1;

  else
```

```
            return 0;
}/*End if isEmpty_stack()*/


void create_graph()
{
            int i,max_edges,origin,destin;


        printf("\nEnter number of nodes : ");
        scanf("%d",&n);
        max_edges=n*(n-1);


            for(i=1;i<=max_edges;i++)
            {
                printf("\nEnter edge %d( -1 -1 to quit ) : ",i);
                scanf("%d %d",&origin,&destin);


                if( (origin == -1) && (destin == -1) )
                        break;


                if( origin >= n || destin >= n || origin<0 || destin<0)
                {
                        printf("\nInvalid edge!\n");
                        i--;
                }
                else
                {
                        adj[origin][destin] = 1;
                }
            }
}
```

Output:



```
main.c:31:1: warning: return type defaults to 'int' [-Wimplicit-int]

Enter number of nodes : 6

Enter edge 1( -1 -1 to quit ) : 0 1

Enter edge 2( -1 -1 to quit ) : 0 2

Enter edge 3( -1 -1 to quit ) : 0 3

Enter edge 4( -1 -1 to quit ) : 1 3

Enter edge 5( -1 -1 to quit ) : 3 4

Enter edge 6( -1 -1 to quit ) : 4 2

Enter edge 7( -1 -1 to quit ) : 5 5

Enter edge 8( -1 -1 to quit ) : -1 -1

Enter starting node for Depth First Search : 0
0 1 3 4 2


...Program finished with exit code 0
Press ENTER to exit console.
```

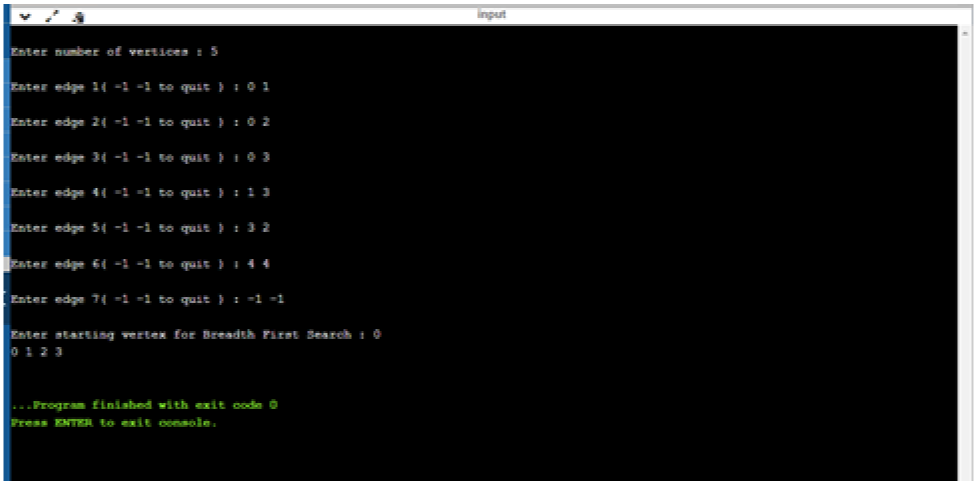
2) Graph traversal technique BFS (using queue)

```
#include<stdio.h>
#include<stdlib.h>

#define MAX 100

#define initial 1
#define waiting 2
#define visited 3

int n; /*Number of vertices in the graph*/
int adj[MAX][MAX]; /*Adjacency Matrix*/
int state[MAX]; /*can be initial, waiting or visited*/

void create_graph();
void BF_Traversal();
void BFS(int v);

int queue[MAX], front = -1,rear = -1;
void insert_queue(int vertex);
int delete_queue();
int isEmpty_queue();

int main()
```

```
{
 create_graph();
 BF_Traversal();


 return 0;


}/*End of main()*/


void BF_Traversal()
{
 int v;


 for(v=0; v<n; v++)
 state[v] = initial;


 printf("\nEnter starting vertex for Breadth First Search : ");
 scanf("%d", &v);
 BFS(v);
}/*End of BF_Traversal()*/


void BFS(int v)
{
 int i;
 insert_queue(v);
 state[v] = waiting;

 while(!isEmpty_queue())
 {
 v = delete_queue( );
 printf("%d ",v);
 state[v] = visited;

 for(i=0; i<n; i++)
 {
 /*Check for adjacent unvisited vertices */
 if(adj[v][i] == 1 && state[i] == initial)
 {
 insert_queue(i);
 state[i] = waiting;
 }
 }
 }
 printf("\n");
}/*End of BFS()*/

void insert_queue(int vertex)
{
 if(rear == MAX-1)

 printf("\nQueue Overflow\n");

 else

 {

 if(front == -1) /*If queue is initially empty */

 front = 0;

 rear = rear+1;

 queue[rear] = vertex ;

 }

}/*End of insert_queue()*/


int isEmpty_queue()

{

 if(front == -1 || front > rear)

 return 1;

 else

 return 0;

}/*End of isEmpty_queue()*/


int delete_queue()

{

 int del_item;

 if(front == -1 || front > rear)

 {
```

```
 printf("\nQueue Underflow\n");
 exit(1);
 }

 del_item = queue[front];
 front = front+1;
 return del_item;
}/*End of delete_queue() */

void create_graph()
{
 int i,max_edges,origin,destin;

 printf("\nEnter number of vertices : ");
 scanf("%d",&n);
 max_edges = n*(n-1);

 for(i=1; i<=max_edges; i++)
 {
 printf("\nEnter edge %d( -1 -1 to quit ) : ",i);
 scanf("%d %d",&origin,&destin);

 if((origin == -1) && (destin == -1))
 break;
 if(origin>=n || destin>=n || origin<0 || destin<0)
 {
 printf("\nInvalid edge!\n");
 i--;
 }
 else
 {
 adj[origin][destin] = 1;
 }
 }
}
```

Output:



```
Enter number of vertices : 5

Enter edge 1( -1 -1 to quit ) : 0 1

Enter edge 2( -1 -1 to quit ) : 0 2

Enter edge 3( -1 -1 to quit ) : 0 3

Enter edge 4( -1 -1 to quit ) : 1 3

Enter edge 5( -1 -1 to quit ) : 3 2

Enter edge 6( -1 -1 to quit ) : 4 4

Enter edge 7( -1 -1 to quit ) : -1 -1

Enter starting vertex for Breadth First Search : 0
0 1 2 3


...Program finished with exit code 0
Press ENTER to exit console.
```

3) Topological Sorting(can be applied only in Directed acyclic graphs)

```
#include<stdio.h>
#include<stdlib.h>

#define MAX 100

int n;    /*Number of vertices in the graph*/
int adj[MAX][MAX]; /*Adjacency Matrix*/
void create_graph();

int queue[MAX], front = -1,rear = -1;
void insert_queue(int v);
int delete_queue();
int isEmpty_queue();

int indegree(int v);

int main()
{
        int i,v,count,topo_order[MAX],indeg[MAX];

      create_graph();
```

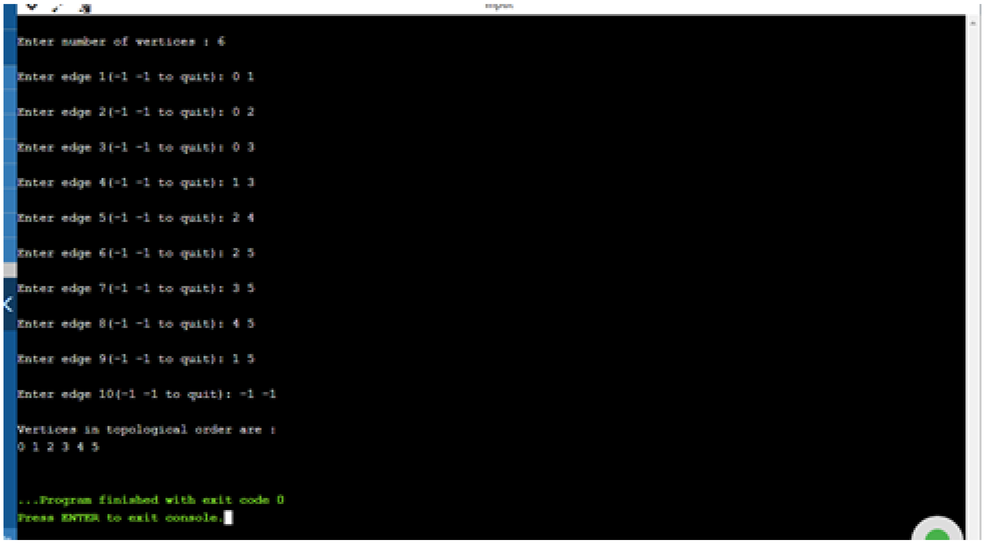
```
   /*Find the indegree of each vertex*/

for(i=0;i<n;i++)

   {

       indeg[i] = indegree(i);

       if( indeg[i] == 0 )

              insert_queue(i);

   }


   count = 0;


while(  !isEmpty_queue( ) && count < n )

   {

       v = delete_queue();

topo_order[++count] = v; /*Add vertex v to topo_order array*/

       /*Delete all edges going fron vertex v */

       for(i=0; i<n; i++)

       {

              if(adj[v][i] == 1)

              {

                     adj[v][i] = 0;

                     indeg[i] = indeg[i]-1;

                     if(indeg[i] == 0)

                         insert_queue(i);

              }
```

```
            }
        }

        if( count < n )
        {
            printf("\nNo topological ordering possible, graph contains cycle\n");
            exit(1);
        }
      printf("\nVertices in topological order are :\n");
        for(i=1; i<=count; i++)
            printf( "%d ",topo_order[i] );
      printf("\n");

        return 0;
}/*End of main()*/

void insert_queue(int vertex)
{
        if (rear == MAX-1)
            printf("\nQueue Overflow\n");
        else
        {
            if (front == -1)  /*If queue is initially empty */
                  front = 0;
            rear = rear+1;
            queue[rear] = vertex ;
        }
}/*End of insert_queue()*/

int isEmpty_queue()
{
        if(front == -1 || front > rear )
            return 1;
        else
            return 0;
}/*End of isEmpty_queue()*/

int delete_queue()
{
        int del_item;
        if (front == -1 || front > rear)
        {
            printf("\nQueue Underflow\n");
            exit(1);
        }
        else
        {
            del_item = queue[front];
```

```
           front = front+1;
           return del_item;
        }
}/*End of delete_queue() */

int indegree(int v)
{
        int i,in_deg = 0;
        for(i=0; i<n; i++)
           if(adj[i][v] == 1)
                in_deg++;
        return in_deg;
}/*End of indegree() */

void create_graph()
{
        int i,max_edges,origin,destin;

     printf("\nEnter number of vertices : ");
     scanf("%d",&n);
     max_edges = n*(n-1);

        for(i=1; i<=max_edges; i++)
        {
```

```
        printf("\nEnter edge %d(-1 -1 to quit): ",i);
        scanf("%d %d",&origin,&destin);

        if((origin == -1) && (destin == -1))
            break;

        if( origin >= n || destin >= n || origin<0 || destin<0)
        {
            printf("\nInvalid edge!\n");
            i--;
        }
        else
            adj[origin][destin] = 1;
    }
}
```

Output:



```
Enter number of vertices : 6

Enter edge 1(-1 -1 to quit): 0 1

Enter edge 2(-1 -1 to quit): 0 2

Enter edge 3(-1 -1 to quit): 0 3

Enter edge 4(-1 -1 to quit): 1 3

Enter edge 5(-1 -1 to quit): 2 4

Enter edge 6(-1 -1 to quit): 2 5

Enter edge 7(-1 -1 to quit): 3 5

Enter edge 8(-1 -1 to quit): 4 5

Enter edge 9(-1 -1 to quit): 1 5

Enter edge 10(-1 -1 to quit): -1 -1

Vertices in topological order are :
0 1 2 3 4 5


...Program finished with exit code 0
Press ENTER to exit console.
```