1.BFS

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

#define MAX 100

#define initial 1
#define visited 2

int n;
int adj[MAX][MAX];
int state[MAX];

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();
}

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");
}

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);
            }
      }
}

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

}

int pop()
{
      int v;
      if(top == -1)
      {
            printf("\nStack Underflow\n");
            exit(1);
      }
      else
      {
```

```
        v = stack[top];
        top=top-1;
        return v;
    }
}

int isEmpty_stack( )
{
  if(top == -1)
      return 1;
  else
      return 0;
}

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

```
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 4
```

2.DFS

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

#define MAX 100

#define initial 1
#define visited 2

int n;
int adj[MAX][MAX];
int state[MAX];

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();
}
```

```
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");
}

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);
		}
	}
```

```
}

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

}

int pop()
{
    int v;
    if(top == -1)
    {
        printf("\nStack Underflow\n");
        exit(1);
    }
    else
    {
        v = stack[top];
        top=top-1;
        return v;
    }
}

int isEmpty_stack( )
```

```
{
  if(top == -1)
        return 1;
  else
        return 0;
}

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

```
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 ) : 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

Enter starting node for Depth First Search : 0
0 1 3 5 2 4
```

3.topological sort

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

#define MAX 100

#define initial 1
#define visited 2

int n;
int adj[MAX][MAX];
int state[MAX];

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();
}
```

```
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");
}

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);
		}
	}
```

```
}

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

}

int pop()
{
    int v;
    if(top == -1)
    {
        printf("\nStack Underflow\n");
        exit(1);
    }
    else
    {
        v = stack[top];
        top=top-1;
        return v;
    }
}

int isEmpty_stack( )
```

```
{
  if(top == -1)
        return 1;
  else
        return 0;
}

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

```
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
```