Ashique P Raj

Mca A Batch

Roll no :27

1.DFS USING STACK

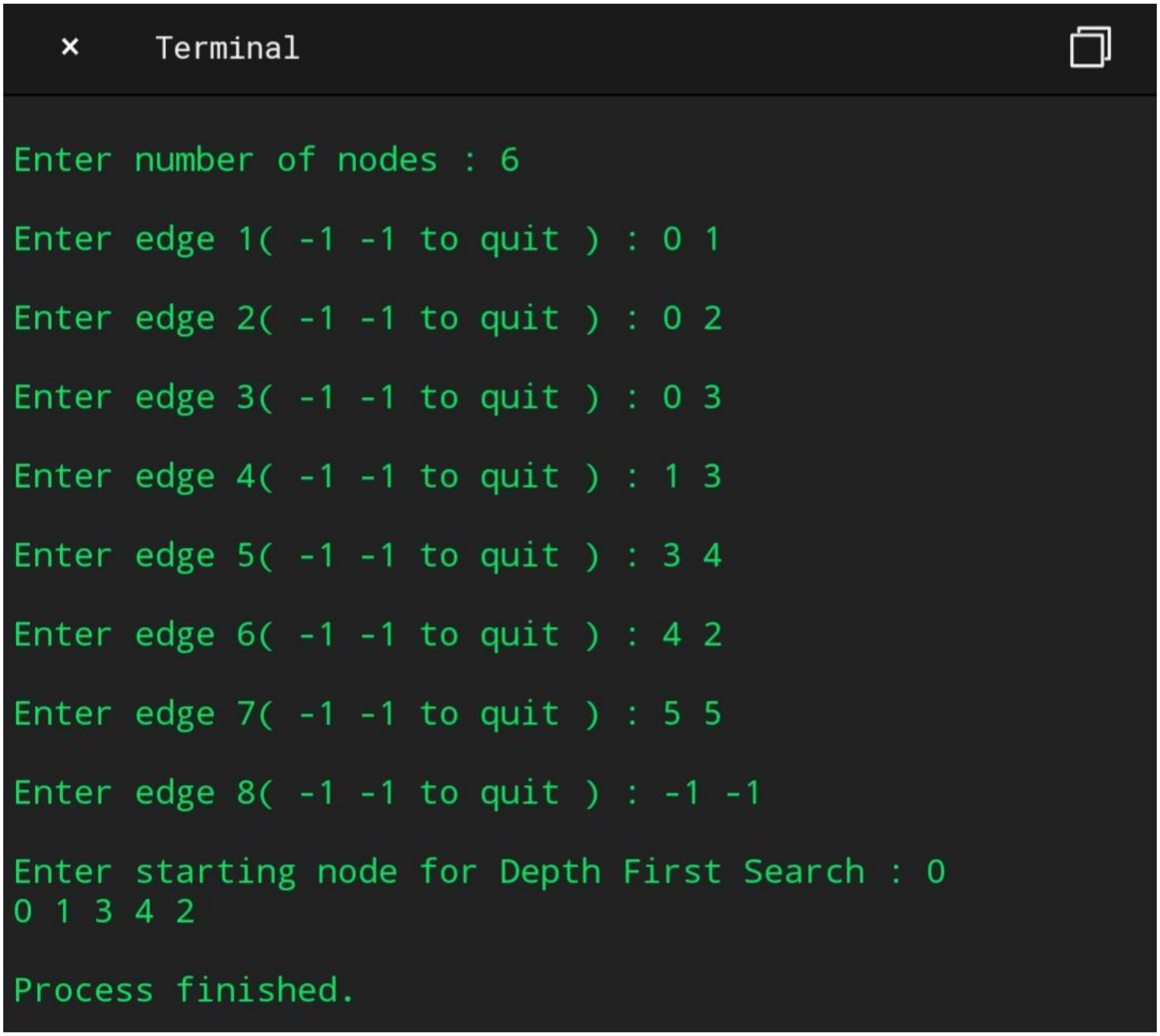
PROGRAM

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

#define MAX 100

#define initial 1
#define visited 2

Int n;
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



```
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

Process finished.
```

2.BFS USING QUEUE

PROGRAM

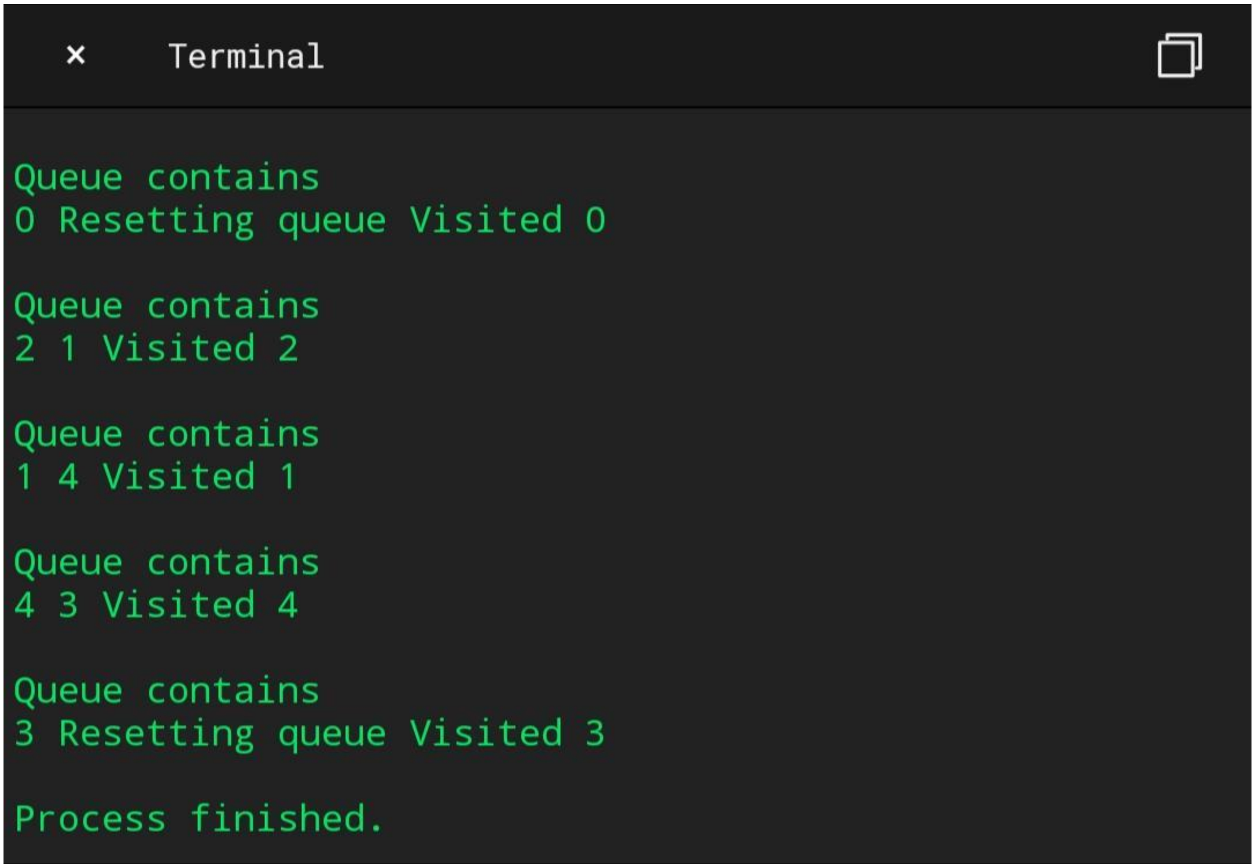
```
#include <stdio.h>

#include <stdlib.h>

#define SIZE 40
```

```
Struct queue {
  Int items[SIZE];
  Int front;
  Int rear;
};

Struct queue* createQueue();
Void enqueue(struct queue* q, int);
Int dequeue(struct queue* q);
Void display(struct queue* q);
Int isEmpty(struct queue* q);
Void printQueue(struct queue* q);

Struct node {
  Int vertex;
  Struct node* next;
};

Struct node* createNode(int);

Struct Graph {
  Int numVertices;
  Struct node** adjLists;
  Int* visited;
};

Void bfs(struct Graph* graph, int startVertex) {
  Struct queue* q = createQueue();
```

```
 Graph->visited[startVertex] = 1;
 Enqueue(q, startVertex);

 While (!isEmpty(q)) {
  printQueue(q);
  int currentVertex = dequeue(q);
  printf(“Visited %d\n”, currentVertex);

  struct node* temp = graph->adjLists[currentVertex];

  while (temp) {
   int adjVertex = temp->vertex;

   if (graph->visited[adjVertex] == 0) {
    graph->visited[adjVertex] = 1;
    enqueue(q, adjVertex);
   }
   Temp = temp->next;
  }
 }
}

Struct node* createNode(int v) {
 Struct node* newNode = malloc(sizeof(struct node));
 newNode->vertex = v;
 newNode->next = NULL;
 return newNode;
}
```

```
Struct Graph* createGraph(int vertices) {
  Struct Graph* graph = malloc(sizeof(struct Graph));
  Graph->numVertices = vertices;

  Graph->adjLists = malloc(vertices * sizeof(struct node*));
  Graph->visited = malloc(vertices * sizeof(int));

  Int I;
  For (I = 0; I < vertices; i++) {
    Graph->adjLists[i] = NULL;
    Graph->visited[i] = 0;
  }

  Return graph;
}

Void addEdge(struct Graph* graph, int src, int dest) {
  Struct node* newNode = createNode(dest);
  newNode->next = graph->adjLists[src];
  graph->adjLists[src] = newNode;

  newNode = createNode(src);
  newNode->next = graph->adjLists[dest];
  graph->adjLists[dest] = newNode;
}


Struct queue* createQueue() {
  Struct queue* q = malloc(sizeof(struct queue));
  q->front = -1;
  q->rear = -1;
  return q;
}


Int isEmpty(struct queue* q) {
  If (q->rear == -1)
    Return 1;
  Else
    Return 0;
}

Void enqueue(struct queue* q, int value) {
  If (q->rear == SIZE – 1)
    Printf(“\nQueue is Full!!”);
  Else {
    If (q->front == -1)
      q->front = 0;
    q->rear++;
    q->items[q->rear] = value;
  }
}

Int dequeue(struct queue* q) {
  Int item;
  If (isEmpty(q)) {
    Printf(“Queue is empty”);
    Item = -1;
  } else {
    Item = q->items[q->front];
    q->front++;
    if (q->front > q->rear) {
      printf(“Resetting queue “);
      q->front = q->rear = -1;
    }
  }
  Return item;
}

// Print the queue
Void printQueue(struct queue* q) {
  Int I = q->front;

  If (isEmpty(q)) {
    Printf(“Queue is empty”);
  } else {
    Printf(“\nQueue contains \n”);
    For (I = q->front; I < q->rear + 1; i++) {
      Printf(“%d “, q->items[i]);
    }
  }
}

Int main() {
  Struct Graph* graph = createGraph(6);
  addEdge(graph, 0, 1);
  addEdge(graph, 0, 2);
  addEdge(graph, 1, 2);
  addEdge(graph, 1, 4);
  addEdge(graph, 1, 3);
  addEdge(graph, 2, 4);
  addEdge(graph, 3, 4);

  bfs(graph, 0);

  return 0;
}
```

OUTPUT



```
Queue contains
0 Resetting queue Visited 0

Queue contains
2 1 Visited 2

Queue contains
1 4 Visited 1

Queue contains
4 3 Visited 4

Queue contains
3 Resetting queue Visited 3

Process finished.
```

3.PROGRAM FOR TOPOLOGICAL SORTING CAN BE APPLIED ONLY DIRECTED SORTING

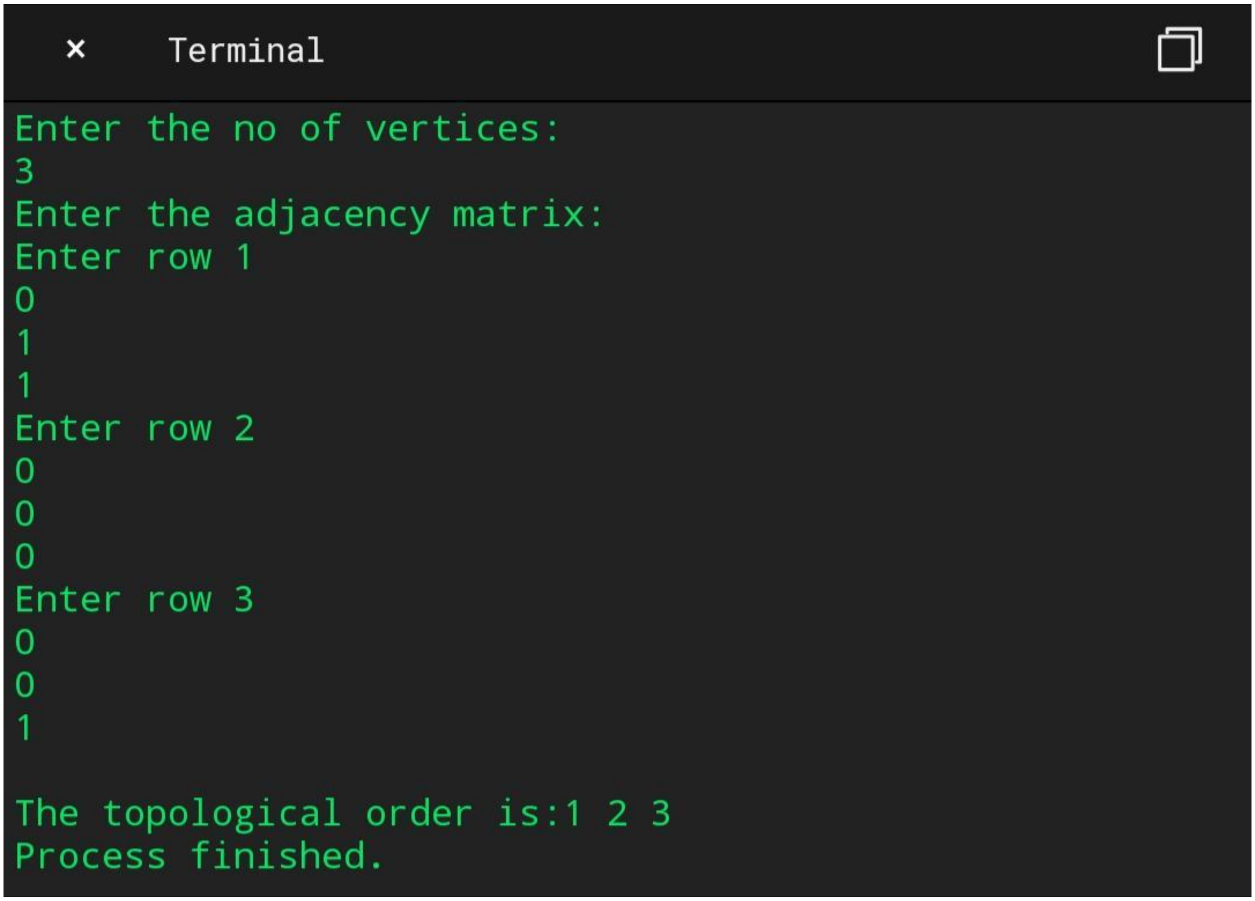
PROGRAM

```
#include <stdio.h>

Int main(){
  Int I,j,k,n,a[10][10],indeg[10],flag[10],count=0;

  Printf("Enter the no of vertices:\n");
```

```
Scanf(“%d”,&n);


Printf(“Enter the adjacency matrix:\n”);
For(i=0;i<n;i++){
 Printf(“Enter row %d\n”,i+1);
 For(j=0;j<n;j++)
  Scanf(“%d”,&a[i][j]);
}


For(i=0;i<n;i++){
   Indeg[i]=0;
   Flag[i]=0;
 }


 For(i=0;i<n;i++)
   For(j=0;j<n;j++)
     Indeg[i]=indeg[i]+a[j][i];


 Printf(“\nThe topological order is:”);


 While(count<n){
   For(k=0;k<n;k++){
     If((indeg[k]==0) && (flag[k]==0)){
       Printf(“%d “,(k+1));
       Flag [k]=1;
     }


     For(i=0;i<n;i++){
       If(a[i][k]==1)
```

```
          Indeg[k]--;
      }
    }

    Count++;
  }

  Return 0;
}
```

OUTPUT



```
Enter the no of vertices:
3
Enter the adjacency matrix:
Enter row 1
0
1
1
Enter row 2
0
0
0
Enter row 3
0
0
1

The topological order is:1 2 3
Process finished.
```