1.Graph Traversal techniques DFS (using stack)

#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

{

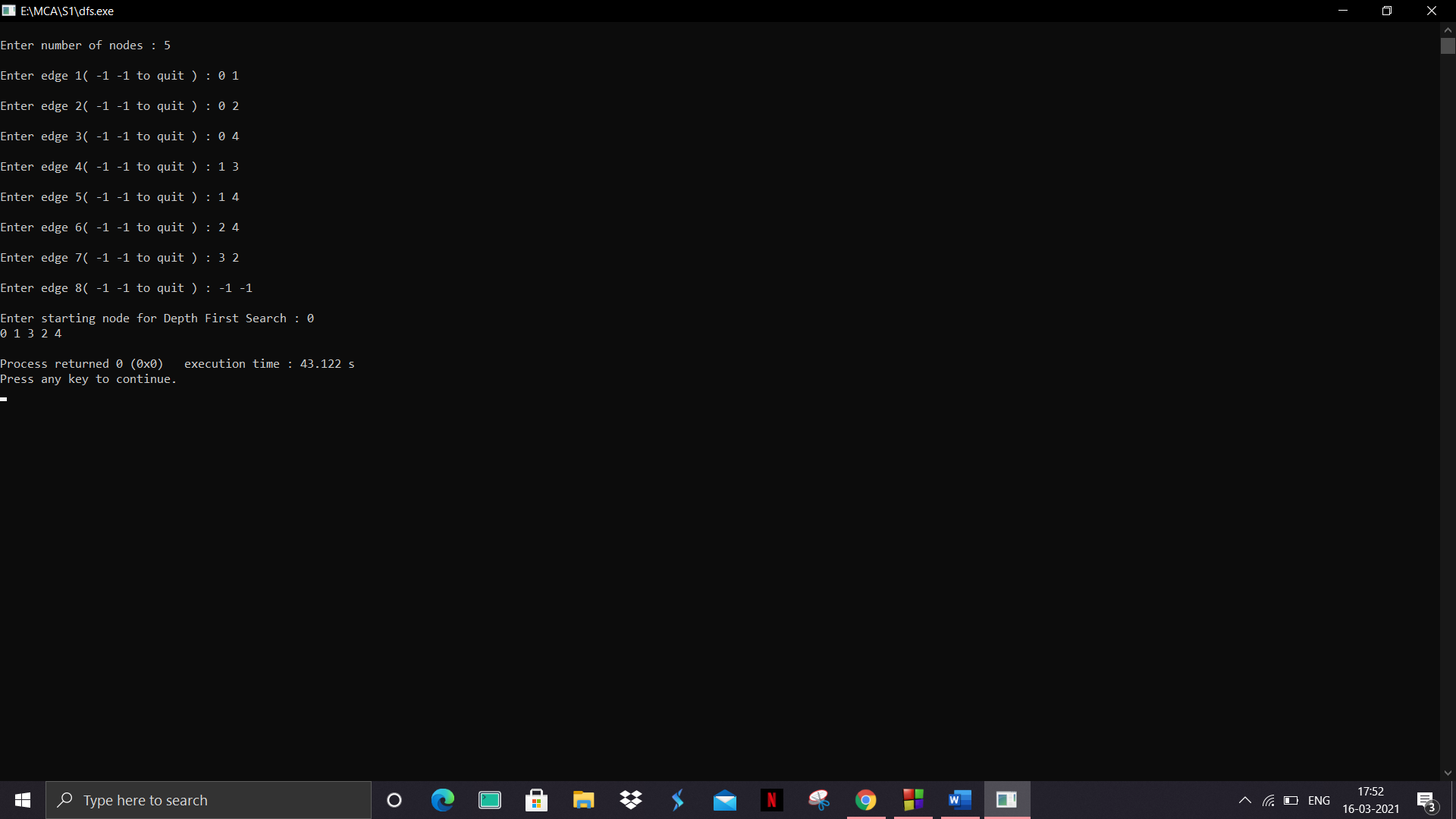
adj[origin][destin] = 1;

}

}

}

OUTPUT



2.Graph Traversal techniques BFS(using queue)

#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);

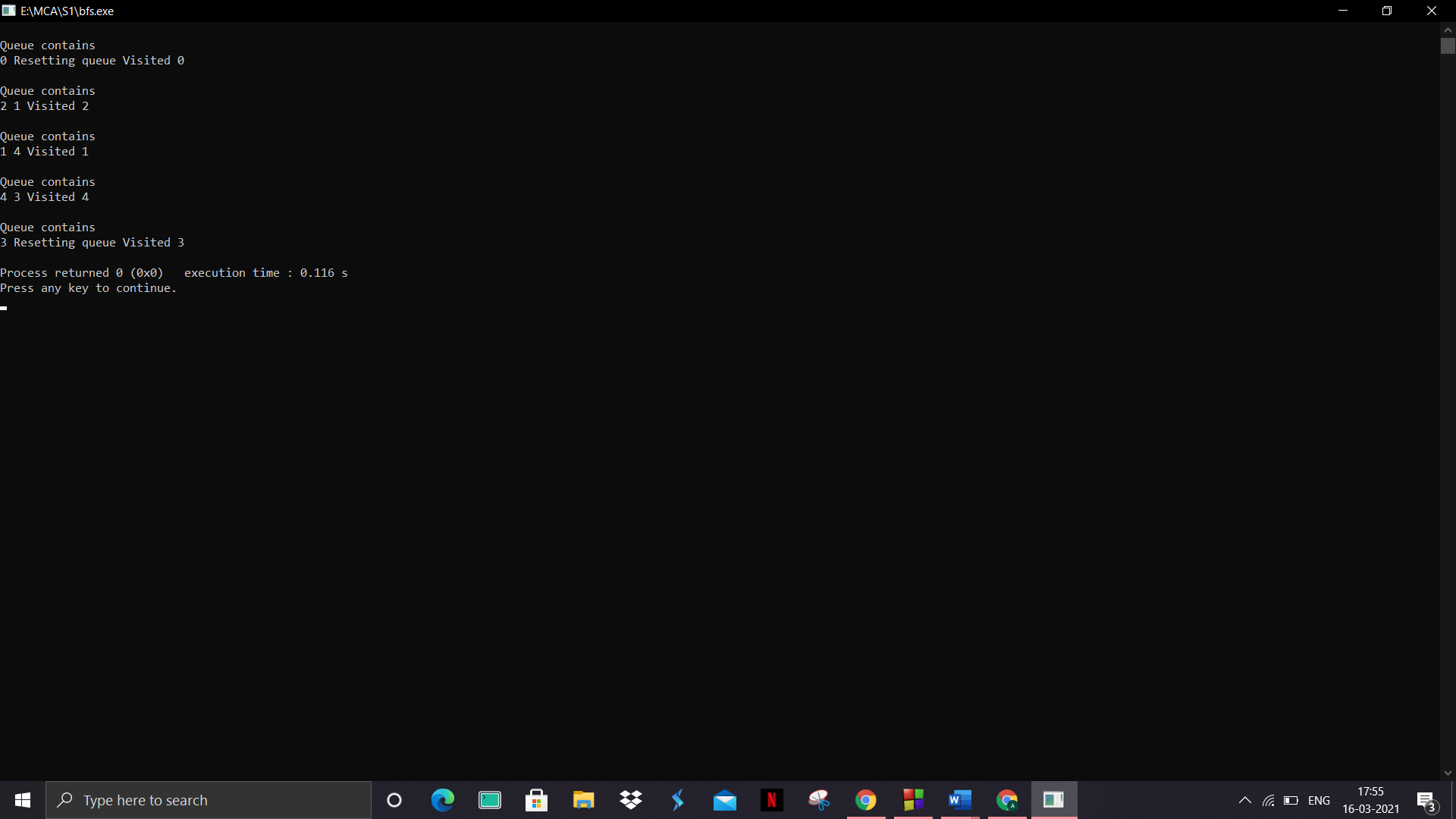
addEdge(graph, 3, 4);

bfs(graph, 0);

return 0;

}

OUTPUT



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

#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

