**LAB CYCLE-4**

**SUBMITTED BY**

**ANUSHREE**

**MCA139**

**PROGRAM-1**

**Q1:** Implement Graph Traversal Technique : BFS

**PROGRAM CODE:**

#include<stdio.h>

int a[20][20], q[20], visited[20], n, i, j, f = 0, r = -1;

void bfs(int v)

{

for(i = 1; i <= n; i++)

if(a[v][i] && !visited[i])

q[++r] = i;

if(f <= r) {

visited[q[f]] = 1;

bfs(q[f++]);

}

}

int main()

{

int v;

printf("\n Enter the number of vertices:");

scanf("%d", &n);

for(i=1; i <= n; i++)

{

q[i] = 0;

visited[i] = 0;

}

printf("\n Enter graph data in matrix form:\n");

for(i=1; i<=n; i++)

{

for(j=1;j<=n;j++)

{

scanf("%d", &a[i][j]);

}

}

printf("\n Enter the starting vertex:");

scanf("%d", &v);

bfs(v);

printf("\n The node which are reachable are:\n");

for(i=1; i <= n; i++)

{

if(visited[i])

printf("%d\t", i);

else

{

printf("\n Bfs is not possible. Not all nodes are reachable");

break;

}

}

}

**PROGRAM-2**

**Q2:** Implement Graph Traversal Technique : DFS

**PROGRAM CODE:**

#include<stdio.h>

int a[20][20],reach[20],n;

int dfs(int v)

{

int i;

reach[v]=1;

for (i=1;i<=n;i++)

if(a[v][i] && !reach[i])

{

printf("\n %d->%d",v,i);

dfs(i);

}

}

int main()

{

int i,j,count=0;

printf("\n Enter number of vertices:");

scanf("%d",&n);

for (i=1;i<=n;i++)

{

reach[i]=0;

for (j=1;j<=n;j++)

a[i][j]=0;

}

printf("\n Enter the adjacency matrix:\n");

for (i=1;i<=n;i++)

for (j=1;j<=n;j++)

scanf("%d",&a[i][j]);

dfs(1);

printf("\n");

for (i=1;i<=n;i++)

{

if(reach[i])

count++;

}

if(count==n)

printf("\n Graph is connected"); else

printf("\n Graph is not connected");

return 0;

}

**PROGRAM-3**

**Q3:** Implement Graph Traversal Technique : Topological Sorting

**PROGRAM CODE:**

#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

}

**PROGRAM NO-4**

**Q4**: Finding the Strongly connected Components in a directed graph.

## PROGRAM CODE:

#include <stdio.h>

#include <stdlib.h>

#define MAX\_DEGREE 5

#define MAX\_NUM\_VERTICES 20

struct vertices\_s {

int visited;

int deg;

int adj[MAX\_DEGREE]; /\* < 0 if incoming edge \*/

} vertices[] = {

{0, 3, {2, -3, 4}},

{0, 2, {-1, 3}},

{0, 3, {1, -2, 7}},

{0, 3, {-1, -5, 6}},

{0, 2, {4, -7}},

{0, 3, {-4, 7, -8}},

{0, 4, {-3, 5, -6, -12}},

{0, 3, {6, -9, 11}},

{0, 2, {8, -10}},

{0, 3, {9, -11, -12}},

{0, 3, {-8, 10, 12}},

{0, 3, {7, 10, -11}}

};

int num\_vertices = sizeof(vertices) / sizeof(vertices[0]);

struct stack\_s {

int top;

int items[MAX\_NUM\_VERTICES];

} stack = {-1, {}};

void stack\_push(int v) {

stack.top++;

if (stack.top < MAX\_NUM\_VERTICES)

stack.items[stack.top] = v;

else {

printf("Stack is full!\n");

exit(1);

}

}

int stack\_pop() {

return stack.top < 0 ? -1 : stack.items[stack.top--];

}

void dfs(int v, int transpose) {

int i, c, n;

vertices[v].visited = 1;

for (i = 0, c = vertices[v].deg; i < c; ++i) {

n = vertices[v].adj[i] \* transpose;

if (n > 0)

/\* n - 1 because vertex indexing begins at 0 \*/

if (!vertices[n - 1].visited)

dfs(n - 1, transpose);

}

if (transpose < 0)

stack\_push(v);

else

printf("%d ", v + 1);

}

void reset\_visited() {

int i;

for (i = 0; i < num\_vertices; ++i)

vertices[i].visited = 0;

}

void order\_pass() {

int i;

for (i = 0; i < num\_vertices; ++i)

if (!vertices[i].visited)

dfs(i, -1);

}

void scc\_pass() {

int i = 0, v;

while((v = stack\_pop()) != -1) {

if (!vertices[v].visited) {

printf("scc %d: ", ++i);

dfs(v, 1);

printf("\n");

}

}

}

int main(void) {

order\_pass();

reset\_visited();

scc\_pass();

return 0;

}