**BFS connectivity**

#include<stdio.h>

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

void bfs(int v)

{

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

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

q[++r]=i;

visited[q[r]]=1;

}

}

if(f<=r)

{

bfs(q[f++]);

}

}

void main(){

int v,c=1;

printf("Enter the no of vertices\n");

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);

visited[v]=1;

bfs(v);

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

if(visited[i]){

c++;

//printf("%d ",q[i]);

}

}

if(c!=n){

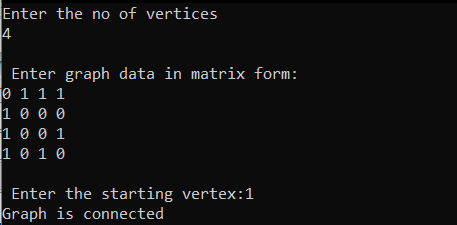
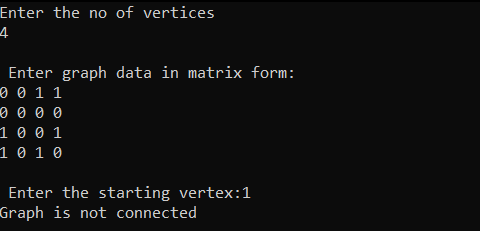
printf("Graph is not connected\n");

}

else{

printf("Graph is connected");

}

}

**BFS acyclicity**

#include<stdio.h>

#include<stdlib.h>

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

void bfs(int v)

{

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

if(a[v][i]==1 && visited[i]!=1){

q[++r]=i;

visited[q[r]]++;

if(visited[i]==1){

printf("Graph is cyclic");

exit(0);

}

}

}

visited[v]=1;

while(f<=r)

{

bfs(q[++f]);

}

}

void main(){

int v;

printf("Enter the no of vertices\n");

scanf("%d",&n);

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

{

q[i]=0;

visited[i]=-1;

}

printf("\n Enter undirected 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);

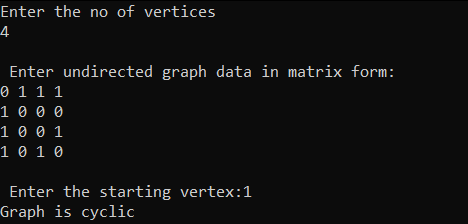
visited[v]++;

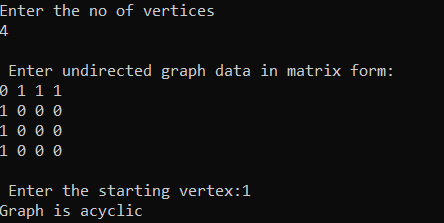
q[++r]=v;

bfs(v);

printf("Graph is acyclic");

}





**Minimum Path using BFS**

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

int n, m = 0;

int adj[MAX][MAX];

int visited[MAX];

int track[MAX];

void create\_graph();

void BF\_Traversal();

void BFS();

int queue[MAX], stack[MAX], track[MAX], front = -1, rear = -1, top = -1;

int track\_size = 0, stack\_size = 0;

void push\_queue(int vertex);

int pop\_queue();

int isEmpty\_queue();

void push\_track(int data);

void shortest\_path();

void push(int p);

int pop();

int main()

{

create\_graph();

BFS();

shortest\_path();

return 0;

}

void BFS()

{

int v, src, des;

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

visited[v] = 0;

printf("Enter the source and destination nodes: ");

scanf("%d%d", &src, &des);

v = src;

int i;

push\_queue(v);

push\_track(9);

while (!isEmpty\_queue())

{

v = pop\_queue();

if(visited[v])

continue;

push\_track(v);

visited[v] = 1;

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

{

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

push\_track(i);

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

return;

if (adj[v][i] == 1 && visited[i] == 0)

{

push\_queue(i);

}

}

push\_track(9);

}

printf("\n");

}

void push\_queue(int vertex)

{

if (rear == MAX - 1)

printf("Queue Overflow\n");

else

{

if (front == -1)

front = 0;

rear = rear + 1;

queue[rear] = vertex;

}

}

int isEmpty\_queue()

{

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

return 1;

else

return 0;

}

int pop\_queue()

{

int delete\_item;

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

{

printf("Queue Underflow\n");

exit(1);

}

delete\_item = queue[front];

front = front + 1;

return delete\_item;

}

void push(int p)

{

top++;

stack[top] = p;

stack\_size++;

}

int pop()

{

return stack[top--];

}

void push\_track(int data)

{

track[m] = data;

m++;

track\_size++;

}

void shortest\_path()

{

push(9);

push(track[track\_size - 1]);

int a, i, j, first = 1;

for (i = track\_size - 1; i >= 0; i--)

{

if (track[i] == 9 && first == 1)

{

a = i + 1;

push(track[a]);

first = 0;

continue;

}

if (track[i] == track[a])

{

for (j = i; j >= 0; j--)

{

if (track[j] == 9)

{

a = j + 1;

push(track[a]);

i = i - (i - j);

break;

}

}

}

}

for (i = 0; i < stack\_size - 1; i++)

{

int q = pop();

if (q == 9)

break;

printf("%d ", q);

}

}

void create\_graph()

{

int count, max\_edge, origin, destin;

printf("Enter number of vertices : ");

scanf("%d", &n);

max\_edge = n \* (n - 1);

for (count = 1; count <= max\_edge; count++)

{

printf("Enter edge %d( -1 -1 to quit ) : ", count);

scanf("%d %d", &origin, &destin);

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

break;

if (origin >= n || destin >= n || origin < 0 || destin < 0)

{

printf("Invalid edge!\n");

count--;

}

else

{

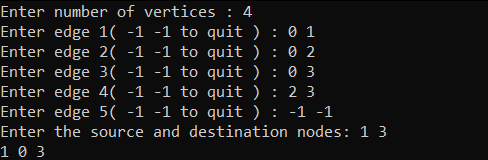
adj[origin][destin] = 1;

adj[destin][origin] = 1;

}

}

}



**BFS order of visit**

#include<stdio.h>

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

void bfs(int v)

{

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

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

q[++r]=i;

visited[q[r]]=1;

}

if(f<=r)

{

bfs(q[f++]);

}

}

void main(){

int v;

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

scanf("%d",&n);

for(i=1;i<=n;i++) 0

{

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);

visited[v]=1;

bfs(v);

printf("\n The order in which nodes reached are:\n");

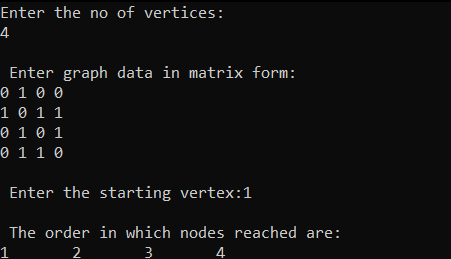
printf("%d\t",v);

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

if(visited[i])

printf("%d\t",q[i]);

}



**Topological sorting using DFS**

#include<stdio.h>

int a[20][20],reach[20],n,order[20],deadend[20],x=0,w=0;

void dfs(int v) {

int i;

reach[v]=1;

int flag=0;

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

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

order[x]=i;

x++;

dfs(i);

}

}

if(flag==0){

deadend[w]=v;

w++;

}

}

void dead\_end(){

int flag;

int i,j;

for(i=0;i<n;i++){flag=1;

for(j=0;j<w;){

if(order[n-i-1]==deadend[j]){flag=0; break; }

else{j++;}

}if(flag==1){

deadend[w]=order[n-i-1];w++;

}

} printf("the top\_sort order is\n");

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

printf("%d ",deadend[n-i-1]);

}

}

void 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++)

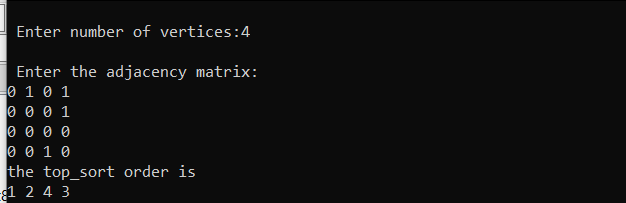
{for (j=1;j<=n;j++){

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

dfs(1);

dead\_end();

}



**Topological sorting using source removal**

#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++){

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;

}

