**// BFS algorithm in C**

#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;

};

// BFS algorithm

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;

}

}

}

// Creating a node

struct node\* createNode(int v) {

struct node\* newNode = malloc(sizeof(struct node));

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

// Creating a graph

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;

}

// Add edge

void addEdge(struct Graph\* graph, int src, int dest) {

// Add edge from src to dest

struct node\* newNode = createNode(dest);

newNode->next = graph->adjLists[src];

graph->adjLists[src] = newNode;

// Add edge from dest to src

newNode = createNode(src);

newNode->next = graph->adjLists[dest];

graph->adjLists[dest] = newNode;

}

// Create a queue

struct queue\* createQueue() {

struct queue\* q = malloc(sizeof(struct queue));

q->front = -1;

q->rear = -1;

return q;

}

// Check if the queue is empty

int isEmpty(struct queue\* q) {

if (q->rear == -1)

return 1;

else

return 0;

}

// Adding elements into queue

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;

}

}

// Removing elements from queue

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;

}

// DFS algorithm in C

#include <stdio.h>

#include <stdlib.h>

struct node {

int vertex;

struct node\* next;

};

struct node\* createNode(int v);

struct Graph {

int numVertices;

int\* visited;

// We need int\*\* to store a two dimensional array.

// Similary, we need struct node\*\* to store an array of Linked lists

struct node\*\* adjLists;

};

**// DFS algo**

void DFS(struct Graph\* graph, int vertex) {

struct node\* adjList = graph->adjLists[vertex];

struct node\* temp = adjList;

graph->visited[vertex] = 1;

printf("Visited %d \n", vertex);

while (temp != NULL) {

int connectedVertex = temp->vertex;

if (graph->visited[connectedVertex] == 0) {

DFS(graph, connectedVertex);

}

temp = temp->next;

}

}

// Create a node

struct node\* createNode(int v) {

struct node\* newNode = malloc(sizeof(struct node));

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

// Create graph

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;

}

// Add edge

void addEdge(struct Graph\* graph, int src, int dest) {

// Add edge from src to dest

struct node\* newNode = createNode(dest);

newNode->next = graph->adjLists[src];

graph->adjLists[src] = newNode;

// Add edge from dest to src

newNode = createNode(src);

newNode->next = graph->adjLists[dest];

graph->adjLists[dest] = newNode;

}

// Print the graph

void printGraph(struct Graph\* graph) {

int v;

for (v = 0; v < graph->numVertices; v++) {

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

printf("\n Adjacency list of vertex %d\n ", v);

while (temp) {

printf("%d -> ", temp->vertex);

temp = temp->next;

}

printf("\n");

}

}

int main() {

struct Graph\* graph = createGraph(4);

addEdge(graph, 0, 1);

addEdge(graph, 0, 2);

addEdge(graph, 1, 2);

addEdge(graph, 2, 3);

printGraph(graph);

DFS(graph, 2);

return 0;

}

<https://www.coders-hub.com/2013/04/bfs-and-dfs-program-in-c.html>

**C code to implement BFS and DFS**

/\* C program to implement BFS(breadth-first search) and DFS(depth-first search) algorithm \*/  
  
#include<stdio.h>  
  
int q[20],top=-1,front=-1,rear=-1,a[20][20],vis[20],stack[20];  
int delete();  
void add(int item);  
void bfs(int s,int n);  
void dfs(int s,int n);  
void push(int item);  
int pop();  
  
void main()  
{  
int n,i,s,ch,j;  
char c,dummy;  
printf("ENTER THE NUMBER VERTICES ");  
scanf("%d",&n);  
for(i=1;i<=n;i++)  
{  
for(j=1;j<=n;j++)  
{  
printf("ENTER 1 IF %d HAS A NODE WITH %d ELSE 0 ",i,j);  
scanf("%d",&a[i][j]);  
}  
}  
printf("THE ADJACENCY MATRIX IS\n");  
for(i=1;i<=n;i++)  
{  
for(j=1;j<=n;j++)  
{  
printf(" %d",a[i][j]);  
}  
printf("\n");  
}  
  
do  
{  
for(i=1;i<=n;i++)  
vis[i]=0;  
printf("\nMENU");  
printf("\n1.B.F.S");  
printf("\n2.D.F.S");  
printf("\nENTER YOUR CHOICE");  
scanf("%d",&ch);  
printf("ENTER THE SOURCE VERTEX :");  
scanf("%d",&s);  
  
switch(ch)  
{  
case 1:bfs(s,n);  
break;  
case 2:  
dfs(s,n);  
break;  
}  
printf("DO U WANT TO CONTINUE(Y/N) ? ");  
scanf("%c",&dummy);  
scanf("%c",&c);  
}while((c=='y')||(c=='Y'));  
}  
  
  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*BFS(breadth-first search) code\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  
void bfs(int s,int n)  
{  
int p,i;  
add(s);  
vis[s]=1;  
p=delete();  
if(p!=0)  
printf(" %d",p);  
while(p!=0)  
{  
for(i=1;i<=n;i++)  
if((a[p][i]!=0)&&(vis[i]==0))  
{  
add(i);  
vis[i]=1;  
}  
p=delete();  
if(p!=0)  
printf(" %d ",p);  
}  
for(i=1;i<=n;i++)  
if(vis[i]==0)  
bfs(i,n);  
}  
  
  
void add(int item)  
{  
if(rear==19)  
printf("QUEUE FULL");  
else  
{  
if(rear==-1)  
{  
q[++rear]=item;  
front++;  
}  
else  
q[++rear]=item;  
}  
}  
int delete()  
{  
int k;  
if((front>rear)||(front==-1))  
return(0);  
else  
{  
k=q[front++];  
return(k);  
}  
}  
  
  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DFS(depth-first search) code\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  
void dfs(int s,int n)  
{  
int i,k;  
push(s);  
vis[s]=1;  
k=pop();  
if(k!=0)  
printf(" %d ",k);  
while(k!=0)  
{  
for(i=1;i<=n;i++)  
if((a[k][i]!=0)&&(vis[i]==0))  
{  
push(i);  
vis[i]=1;  
}  
k=pop();  
if(k!=0)  
printf(" %d ",k);  
}  
for(i=1;i<=n;i++)  
if(vis[i]==0)  
dfs(i,n);  
}  
void push(int item)  
{  
if(top==19)  
printf("Stack overflow ");  
else  
stack[++top]=item;  
}  
int pop()  
{  
int k;  
if(top==-1)  
return(0);  
else  
{  
k=stack[top--];  
return(k);  
}  
}