**Student Name:- CHAUDHARY HAMDAN**

**Student Roll No.:- 1905387**

**Algorithm Lab. Class Assignment-12**

**CSE Group 1**

**Date: - 22nd October 2021**

1. Write a C program to implement Depth First Search.

**Program**

#include <stdio.h>

#include <stdlib.h>

#define sf(x) scanf("%d", &x)

#define pf printf

#define pfs(x) printf("%d ", x)

#define pfn(x) printf("%d\n", x)

#define pfc(x) printf("%d, ", x)

#define FI(i,x,y,inc) for(int i = x; i < y; i += inc)

#define F(i,x,y) FI(i, x, y, 1)

#define F0(i,n) FI(i, 0, n, 1)

#define RF(i,x,y) for(int i = x; i >= y; i--)

#define pfarr(i,a,n) for(int i = 0; i < n-1; i++) pfs(a[i]); pfn(a[n-1]);

void i\_o\_from\_file() {

#ifndef ONLINE\_JUDGE

freopen("C:\\Users\\KIIT\\input", "r", stdin);

freopen("C:\\Users\\KIIT\\output", "w", stdout);

#endif

}

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;

}

}

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;

}

// 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;

}

int main() {

i\_o\_from\_file();

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

int v;

sf(v);

struct Graph\* graph = createGraph(v);

while (1) {

int a, b;

sf(a);

if (a == -1) {

break;

}

sf(b);

addEdge(graph, a, b);

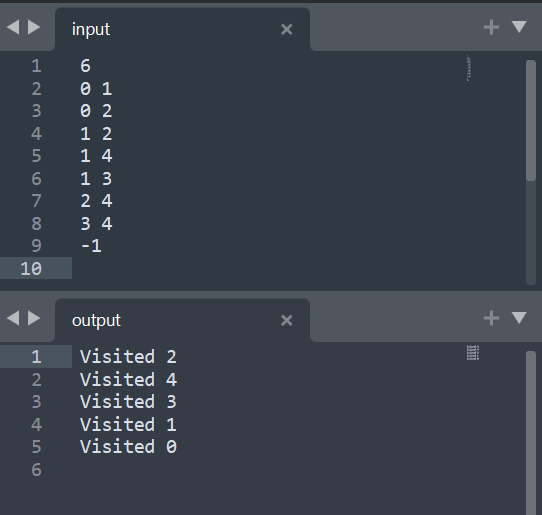
}

DFS(graph, 2);

return 0;

}

**Output**

****

1. Write a C program to check whether a given graph is connected or not using DFS method.

**Program**

#include <stdio.h>

#include <stdlib.h>

#define sf(x) scanf("%d", &x)

int n = 0;

void i\_o\_from\_file() {

#ifndef ONLINE\_JUDGE

freopen("C:\\Users\\KIIT\\input", "r", stdin);

freopen("C:\\Users\\KIIT\\output", "w", stdout);

#endif

}

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;

};

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

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

struct node\* temp = adjList;

graph->visited[vertex] = 1;

n++;

while (temp != NULL) {

int connectedVertex = temp->vertex;

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

DFS(graph, connectedVertex);

}

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;

}

// 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;

}

int main() {

i\_o\_from\_file();

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

int v;

sf(v);

n = 0;

struct Graph\* graph = createGraph(v);

while (1) {

int a, b;

sf(a);

if (a == -1) {

break;

}

sf(b);

addEdge(graph, a, b);

}

DFS(graph, 2);

if (n == v) {

printf("Connected\n");

}

else {

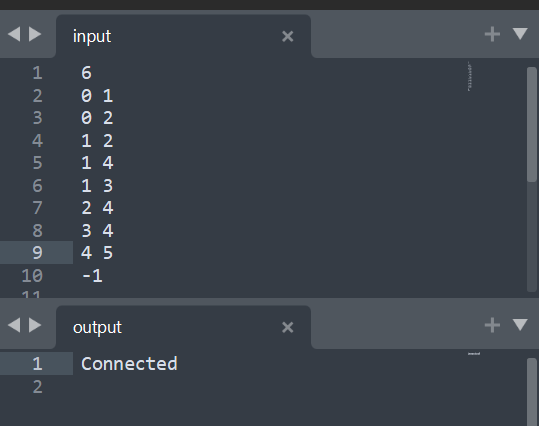
printf("Non Connected\n");

}

return 0;

}

**Output**

****

1. Write a C program to check whether a given graph is connected or not using BFS method.

**Program**

// Author: Chaudhary Hamdan

// Generated at: Fri Oct 8 14:06:41 2021

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#define sf(x) scanf("%d", &x)

#define SIZE 40

int n = 0;

void i\_o\_from\_file();

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);

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)) {

int currentVertex = dequeue(q);

n++;

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) {

// 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;

}

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) {

q->front = q->rear = -1;

}

}

return item;

}

int main() {

i\_o\_from\_file();

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

int v;

sf(v);

n = 0;

struct Graph\* graph = createGraph(v);

while (1) {

int a, b;

sf(a);

if (a == -1) {

break;

}

sf(b);

addEdge(graph, a, b);

}

bfs(graph, 0);

if (n == v) {

printf("Connected\n");

}

else {

printf("Non Connected\n");

}

return 0;

}

void i\_o\_from\_file() {

#ifndef ONLINE\_JUDGE

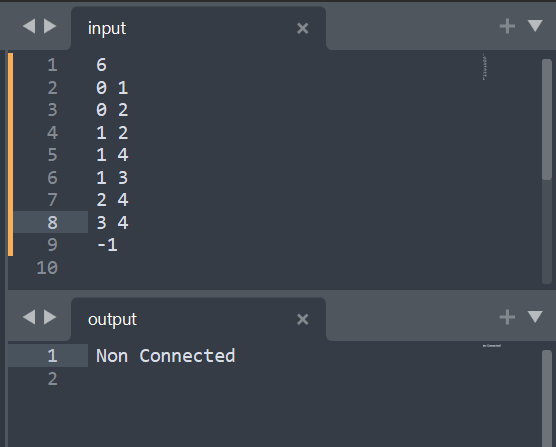
freopen("C:\\Users\\KIIT\\input", "r", stdin);

freopen("C:\\Users\\KIIT\\output", "w", stdout);

#endif

}

**Output**

****

1. Write a C program to print all the nodes reachable from a given starting node in a given digraph using Depth First Search method.

**Program**

// DFS algorithm in C

#include <stdio.h>

#include <stdlib.h>

#define sf(x) scanf("%d", &x)

void i\_o\_from\_file() {

#ifndef ONLINE\_JUDGE

freopen("C:\\Users\\KIIT\\input", "r", stdin);

freopen("C:\\Users\\KIIT\\output", "w", stdout);

#endif

}

struct node {

int vertex;

struct node\* next;

};

struct node\* createNode(int v);

struct Graph {

int numVertices;

int\* visited;

struct node\*\* adjLists;

};

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;

}

int main() {

i\_o\_from\_file();

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

int v;

sf(v);

struct Graph\* graph = createGraph(v);

while (1) {

int a, b;

sf(a);

if (a == -1) {

break;

}

sf(b);

addEdge(graph, a, b);

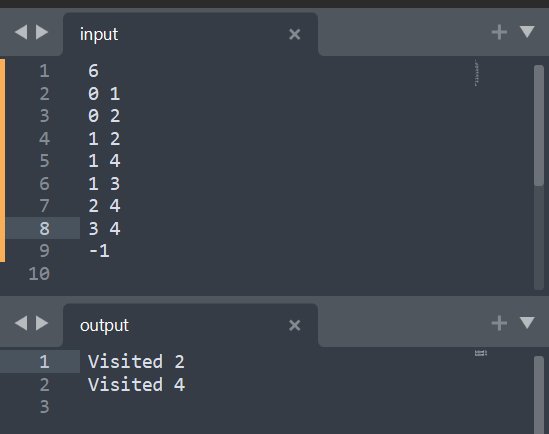
}

DFS(graph, 2);

return 0;

}

**Output**

****

1. Write an algorithm to print all the nodes reachable from a given starting node in a digraph using BFS method.

**Program**

// Author: Chaudhary Hamdan

// Generated at: Fri Oct 8 14:06:41 2021

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#define sf(x) scanf("%d", &x)

#define SIZE 40

void i\_o\_from\_file();

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);

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)) {

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) {

// Add edge from src to dest

struct node\* newNode = createNode(dest);

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

graph->adjLists[src] = 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) {

q->front = q->rear = -1;

}

}

return item;

}

int main() {

i\_o\_from\_file();

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

int v;

sf(v);

struct Graph\* graph = createGraph(v);

while (1) {

int a, b;

sf(a);

if (a == -1) {

break;

}

sf(b);

addEdge(graph, a, b);

}

bfs(graph, 0);

return 0;

}

void i\_o\_from\_file() {

#ifndef ONLINE\_JUDGE

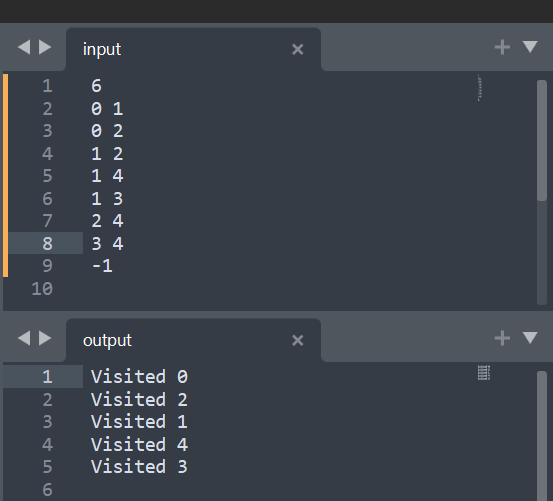
freopen("C:\\Users\\KIIT\\input", "r", stdin);

freopen("C:\\Users\\KIIT\\output", "w", stdout);

#endif

}

**Output**

****