BFS:

#include <stdio.h>

#include <stdlib.h>

// Structure to represent a node in the adjacency list

struct AdjListNode {

int dest;

struct AdjListNode\* next;

};

// Structure to represent an adjacency list

struct AdjList {

struct AdjListNode \*head;

};

// Structure to represent a graph

struct Graph {

int V;

struct AdjList\* array;

};

// Function to create a new adjacency list node

struct AdjListNode\* newAdjListNode(int dest) {

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

newNode->dest = dest;

newNode->next = NULL;

return newNode;

}

// Function to create a graph with V vertices

struct Graph\* createGraph(int V) {

struct Graph\* graph = (struct Graph\*)malloc(sizeof(struct Graph));

graph->V = V;

graph->array = (struct AdjList\*)malloc(V \* sizeof(struct AdjList));

for (int i = 0; i < V; ++i)

graph->array[i].head = NULL;

return graph;

}

// Function to add an edge to the graph

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

struct AdjListNode\* newNode = newAdjListNode(dest);

newNode->next = graph->array[src].head;

graph->array[src].head = newNode;

// Since the graph is undirected, add an edge from dest to src as well

newNode = newAdjListNode(src);

newNode->next = graph->array[dest].head;

graph->array[dest].head = newNode;

}

// Function to perform Breadth First Search traversal of the graph

void BFS(struct Graph\* graph, int start) {

// Mark all the vertices as not visited

int \*visited = (int \*)malloc(graph->V \* sizeof(int));

for (int i = 0; i < graph->V; ++i)

visited[i] = 0;

// Create a queue for BFS

int \*queue = (int \*)malloc(graph->V \* sizeof(int));

int front = 0, rear = 0;

// Mark the current node as visited and enqueue it

visited[start] = 1;

queue[rear++] = start;

while (front < rear) {

// Dequeue a vertex from the queue and print it

int currentVertex = queue[front++];

printf("%d ", currentVertex);

// Get all adjacent vertices of the dequeued vertex currentVertex

// If an adjacent vertex has not been visited, mark it visited and enqueue it

struct AdjListNode\* temp = graph->array[currentVertex].head;

while (temp) {

int adjVertex = temp->dest;

if (!visited[adjVertex]) {

visited[adjVertex] = 1;

queue[rear++] = adjVertex;

}

temp = temp->next;

}

}

free(visited);

free(queue);

}

int main() {

int V, E; // Number of vertices and edges

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

scanf("%d", &V);

// Create a graph with V vertices

struct Graph\* graph = createGraph(V);

printf("Enter the number of edges: ");

scanf("%d", &E);

// Add edges to the graph

printf("Enter edges (src dest):\n");

for (int i = 0; i < E; ++i) {

int src, dest;

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

addEdge(graph, src, dest);

}

int startVertex;

printf("Enter the starting vertex for BFS traversal: ");

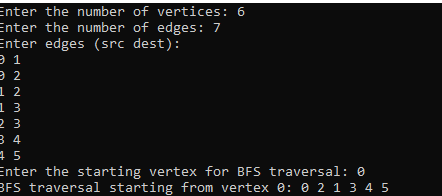
scanf("%d", &startVertex);

printf("BFS traversal starting from vertex %d: ", startVertex);

BFS(graph, startVertex);

return 0;

}



DFS:

#include <stdio.h>

#include <stdlib.h>

// Structure to represent a node in the adjacency list

struct AdjListNode {

int dest;

struct AdjListNode\* next;

};

// Structure to represent an adjacency list

struct AdjList {

struct AdjListNode \*head;

};

// Structure to represent a graph

struct Graph {

int V;

struct AdjList\* array;

};

// Function to create a new adjacency list node

struct AdjListNode\* newAdjListNode(int dest) {

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

newNode->dest = dest;

newNode->next = NULL;

return newNode;

}

// Function to create a graph with V vertices

struct Graph\* createGraph(int V) {

struct Graph\* graph = (struct Graph\*)malloc(sizeof(struct Graph));

graph->V = V;

graph->array = (struct AdjList\*)malloc(V \* sizeof(struct AdjList));

for (int i = 0; i < V; ++i)

graph->array[i].head = NULL;

return graph;

}

// Function to add an edge to the graph

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

struct AdjListNode\* newNode = newAdjListNode(dest);

newNode->next = graph->array[src].head;

graph->array[src].head = newNode;

// Since the graph is undirected, add an edge from dest to src as well

newNode = newAdjListNode(src);

newNode->next = graph->array[dest].head;

graph->array[dest].head = newNode;

}

// Recursive function for DFS traversal

void DFSUtil(struct Graph\* graph, int v, int\* visited) {

visited[v] = 1; // Mark the current vertex as visited

struct AdjListNode\* temp = graph->array[v].head;

while (temp != NULL) {

int adjVertex = temp->dest;

if (!visited[adjVertex])

DFSUtil(graph, adjVertex, visited);

temp = temp->next;

}

}

// Function to check if the graph is connected using DFS

int isConnected(struct Graph\* graph, int V) {

int\* visited = (int\*)malloc(V \* sizeof(int));

int i;

// Mark all vertices as not visited

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

visited[i] = 0;

// Find a vertex with non-zero degree and do DFS traversal starting from it

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

if (graph->array[i].head != NULL) {

DFSUtil(graph, i, visited);

break;

}

// If DFS traversal doesn't visit all vertices, the graph is not connected

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

if (visited[i] == 0)

return 0;

return 1;

}

int main() {

int V, E; // Number of vertices and edges

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

scanf("%d", &V);

// Create a graph with V vertices

struct Graph\* graph = createGraph(V);

printf("Enter the number of edges: ");

scanf("%d", &E);

// Add edges to the graph

printf("Enter edges (src dest):\n");

for (int i = 0; i < E; ++i) {

int src, dest;

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

addEdge(graph, src, dest);

}

if (isConnected(graph, V))

printf("The graph is connected.\n");

else

printf("The graph is not connected.\n");

return 0;

}

