**AIM :**

To create a graph and perform a Breadth First Search (BFS) and Depth First Search (DFS).

**ALGORITHM:**

**Step 1:** Start.

**Step 2**: Create a node which contains vertex and next as their members

**Step 3:** Allocates memory dynamically for nodes and the graph structure using malloc().

**Step 4**: Create a graph with a specified number of vertices and initialize adjacency lists.

**Step 5:** Create a function to add the edges between the vertices .

**Step 6:** Performs BFS traversal starting from a given vertex using a queue data structure to maintain

order.

**Step 7:** Conducts DFS traversal starting from a specified vertex, employing recursion to explore graph

branches.

**Step 8:** Display the adjacency list representation of the graph and the traversal sequences for both BFS

and DFS.

**Step 9:** End

**PROGRAM :**

\* createNode(int v);

struct Graph

#include<stdio.h>

#include<stdlib.h>

#define MAX 100

struct Node {

int vertex;

struct Node\* next;

};

struct Node{

int numVertices;

struct Node\*\* adjLists;

int\* visited;

};

struct Graph\* createGraph(int vertices);

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

void printGraph(struct Graph\* graph);

void BFS(struct Graph\* graph, int startVertex);

void DFS(struct Graph\* graph, int startVertex);

int main() {

struct Graph\* graph = createGraph(4);

addEdge(graph, 0, 1);

addEdge(graph, 0, 2);

addEdge(graph, 1, 2);

addEdge(graph, 2, 0);

addEdge(graph, 2, 3);

addEdge(graph, 3, 3);

printf("Graph:\n");

printGraph(graph);

printf("\nBFS Traversal:\n");

BFS(graph, 2);

printf("\nDFS Traversal:\n");

DFS(graph, 2);

return 0;

}

struct Node\* createNode(int v)

{

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

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

struct Graph\* createGraph(int vertices) {

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

graph->numVertices = vertices;

graph->adjLists = (struct Node\*\*)malloc(vertices \* sizeof(struct Node\*));

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

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

graph->adjLists[i] = NULL;

graph->visited[i] = 0;

}

return graph;

}

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

struct Node\* newNode = createNode(dest);

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

graph->adjLists[src] = newNode;

newNode = createNode(src);

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

graph->adjLists[dest] = newNode;

}

void printGraph(struct Graph\* graph) {

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

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

printf("Vertex %d: ", v);

while (temp) {

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

temp = temp->next;

}

printf("NULL\n");

}

}

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

struct Node\* queue[MAX];

int front = 0, rear = 0;

queue[rear] = createNode(startVertex);

graph->visited[startVertex] = 1;

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

while (front <= rear) {

struct Node\* currentNode = queue[front];

front++;

while (currentNode) {

int adjVertex = currentNode->vertex;

if (!graph->visited[adjVertex]) {

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

queue[++rear] = createNode(adjVertex);

graph->visited[adjVertex] = 1;

}

currentNode = currentNode->next;

}

}

}

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

struct Node\* temp = graph->adjLists[vertex];

graph->visited[vertex] = 1;

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

while (temp) {

int adjVertex = temp->vertex;

if (!graph->visited[adjVertex]) {

DFSUtil(graph, adjVertex);

}

temp = temp->next;

}

}

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

graph->visited[startVertex] = 1;

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

struct Node\* temp = graph->adjLists[startVertex];

while (temp) {

int adjVertex = temp->vertex;

if (!graph->visited[adjVertex]) {

DFSUtil(graph, adjVertex);

}

temp = temp->next;

}

}

**OUTPUT:**

Graph:

Vertex 0: 2 -> 2 -> 1 -> NULL

Vertex 1: 2 -> 0 -> NULL

Vertex 2: 3 -> 0 -> 1 -> 0 -> NULL

Vertex 3: 3 -> 3 -> 2 -> NULL

BFS Traversal:

Visited 2

DFS Traversal:

Visited 2

Visited 3

Visited 0

Visited 1

**RESULT:**

Hence the program to Breadth First Search and Depth First Search is implemented.