

**Batch**: A-4 **Roll No.:** 16010422211  **Experiment No.: 6**

**Aim:** Implement a menu driven program to represent a graph and traverse it using BFS technique.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Resources Used:** C/ C++ editor and compiler.

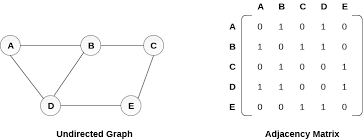
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Theory:**

**Graph**

Given an undirected graph G= (V,E) and a vertex V in V(G), then we are interested in visiting all vertices in G that are reachable from V i.e. all vertices connected to V. There are two techniques of doing it namely Depth First Search (DFS) and Breadth First Search(BFS).

**Graph Representation using Adjacency Matrix**

****

**Depth First Search**

The procedure of performing DFS on an undirected graph can be as follows :

The starting vertex v is visited. Next an unvisited vertex w adjacent to v is selected and a depth first search from w is initiated. When a vertex u is reached such that all its adjacent vertices have been visited, we back up to the last vertex visited which has an unvisited vertex w adjacent to it and initiate a depth first search from w. the search terminates when no unvisited vertex can be reached from any of the visited ones.

Given an undirected graph G=(V,E) with n vertices and an array visited[n] initially set to false, this algorithm, dfs (v) visits all vertices reachable from v. Visited is a global array.

**Breadth First Search**

Starting at vertex v and making it as visited, BFS visits next all unvisited vertices adjacent to v. then unvisited vertices adjacent to there vertices are visited and so on.

A breadth first search of G is carried out beginning at vertex v as bfs (v). All vertices visited are marked as visited [i]=true. The graph G and array visited are global and visited is initialized to false. Initialize, addqueue, emptyqueue, deletequeue are the functions to handle operations on queue.

**Algorithm :**

Implement the static linear queue ADT, Represent the graph using adjacency matrix and implement following pseudo code for BFS.

***Pseudo Code: bfs (v)***

*initialize queue q*

*visited [v] = true*

*addqueue(q,v)*

*while not emptyqueue*

*v=deletequeue(q)*

*add v into bfs sequence*

*for all vertices w adjacent to v do*

*if not visited [w] then*

*addqueue (q,w)*

*visited [w]=true*

**Results:**

#include <stdio.h>

#include <stdbool.h>

#define MAX\_NODES 100

int queue[MAX\_NODES];

int front = -1, rear = -1;

int adjacencyMatrix[MAX\_NODES][MAX\_NODES];

int visited[MAX\_NODES];

int nodes, edges;

void enqueue(int node) {

if (rear == MAX\_NODES - 1) {

printf("Queue is full.\n");

} else {

if (front == -1) {

front = 0;

}

rear++;

queue[rear] = node;

}

}

int dequeue() {

int node;

if (front == -1) {

printf("Queue is empty.\n");

return -1;

} else {

node = queue[front];

front++;

if (front > rear) {

front = rear = -1;

}

return node;

}

}

bool isQueueEmpty() {

return front == -1;

}

void BFS(int startNode) {

int i, current;

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

visited[i] = 0;

}

enqueue(startNode);

visited[startNode] = 1;

while (!isQueueEmpty()) {

current = dequeue();

printf("%d ", current);

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

if (adjacencyMatrix[current][i] && !visited[i]) {

enqueue(i);

visited[i] = 1;

}

}

}

}

int main() {

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

scanf("%d %d", &nodes, &edges);

printf("Enter the adjacency matrix:\n");

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

for (int j = 0; j < nodes; j++) {

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

}

}

int startNode;

printf("Enter the starting node: ");

scanf("%d", &startNode);

if (startNode < 0 || startNode >= nodes) {

printf("Invalid starting node.\n");

return 1;

}

printf("Breadth-First Search starting from node %d: ", startNode);

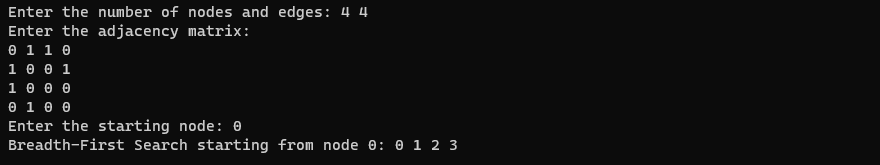
BFS(startNode);

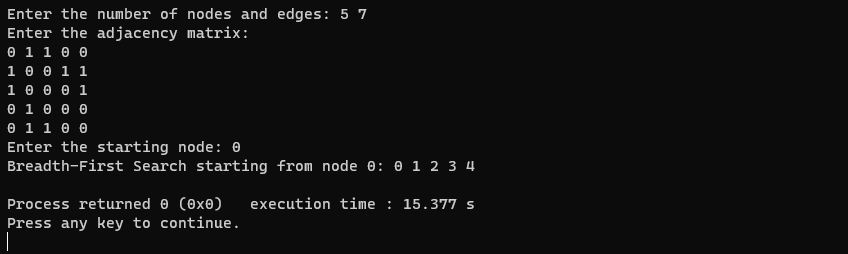
printf("\n");

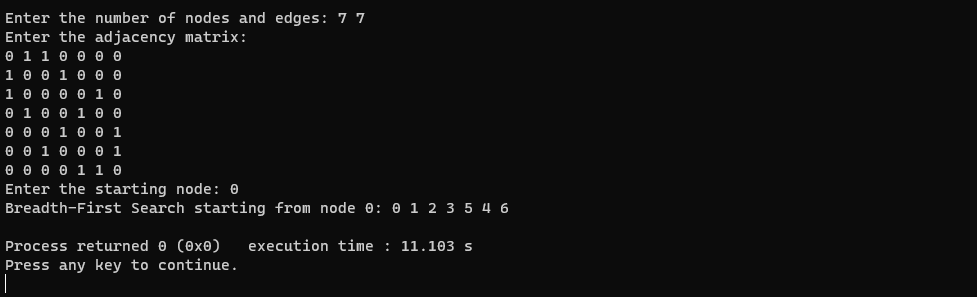
return 0;

}

**Outputs:**







A program depicting the BFS using adjacency matrix and capable of handling all possible boundary conditions and the same is reflected clearly in the output.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Outcomes:** CO2 Apply linear and non-linear data structure in application development.

**Conclusion:** Understood the algorithm & Implemented a menu driven program to represent a graph and traverse it using BFS technique.

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of faculty in-charge with date \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**References:**

**Books/ Journals/ Websites:**

* Y. Langsam, M. Augenstin and A. Tannenbaum, “Data Structures using C”, Pearson Education Asia, 1st Edition, 2002.
* Vlab on BFS