Experiment No. : 6

Title: Graph Traversal using appropriate data structure

Batch: SY-IT(B3) **Roll No.:16010423076** **Experiment No.: 6**

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

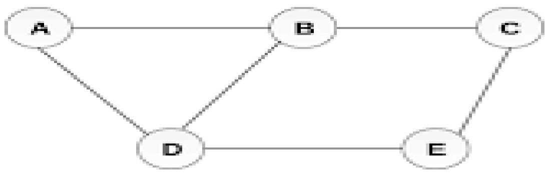
Resources Used: C/ C++ editor and compiler.

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



Undirected Graph

$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 1 & 0 & 1 & 0 \\ B & 1 & 0 & 1 & 1 & 0 \\ C & 0 & 1 & 0 & 0 & 1 \\ D & 1 & 1 & 0 & 0 & 1 \\ E & 0 & 0 & 1 & 1 & 0 \end{array}$$

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)

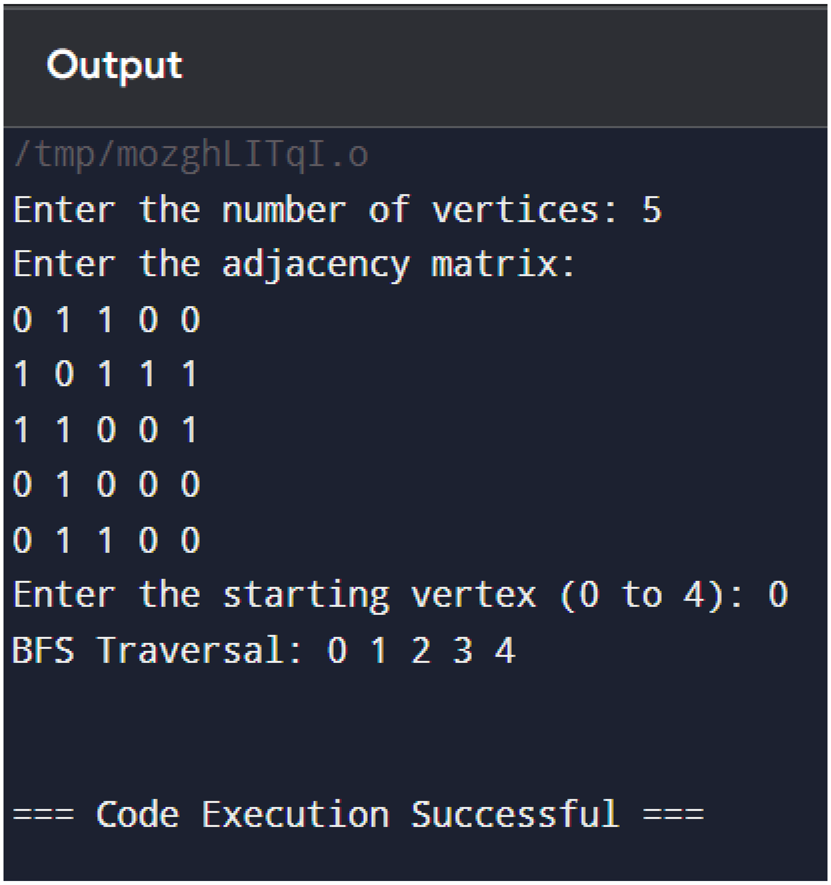
```
initialize queue q
visited [v] = true
addqueue(q,v)
while not emptyqueue
    v=deletequeue(q)
        add v into bfs sequence
        for all vertices w adjacent to v do
            if not visited [w] then
                addqueue (q,w)
                visited [w]=true
```

Results:

```
#include <stdio.h>

#include <stdbool.h>



#define MAX 10



// queue implementation for BFS

int queue[MAX];

int front = -1, rear = -1;
```

```
// functions for Queue Operations

void initializeQueue() {

    front = rear = -1;

}


bool isQueueEmpty() {

    return front == -1;

}


bool isQueueFull() {

    return rear == MAX - 1;

}


void enqueue(int v) {

    if (!isQueueFull()) {

        if (front == -1)

            front = 0;

        queue[++rear] = v;

    } else {

        printf("Queue is Full!\n");

    }

}


int dequeue() {

    if (!isQueueEmpty()) {
        int data = queue[front];
        if (front == rear)
            front = rear = -1; // Reset queue if it's empty
        else
            front++;
        return data;
    } else {
        printf("Queue is Empty!\n");
        return -1;
    }
}

// BFS function
void bfs(int graph[MAX][MAX], int n, int start) {
    bool visited[MAX] = { false }; // Keep track of visited nodes
    initializeQueue();

    visited[start] = true;
    enqueue(start);

    printf("BFS Traversal: ");
    while (!isQueueEmpty()) {
        int v = dequeue();
        printf("%d ", v);

        // Visit all adjacent vertices of v
        for (int w = 0; w < n; w++) {
            if (graph[v][w] == 1 && !visited[w]) {
                visited[w] = true;
                enqueue(w);
            }
        }
    }
    printf("\n");
}

// function to input graph as adjacency matrix
void inputGraph(int graph[MAX][MAX], int *n) {
    printf("Enter the number of vertices: ");
    scanf("%d", n);

    printf("Enter the adjacency matrix:\n");
    for (int i = 0; i < *n; i++) {
        for (int j = 0; j < *n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }
}

// menu-driven program
int main() {
    int graph[MAX][MAX];
    int n, start;


    inputGraph(graph, &n);

    printf("Enter the starting vertex (0 to %d): ", n - 1);

    scanf("%d", &start);

    bfs(graph, n, start);

    return 0;

}
```

Output :



```
/tmp/mozghLITqI.o
Enter the number of vertices: 5
Enter the adjacency matrix:
0 1 1 0 0
1 0 1 1 1
1 1 0 0 1
0 1 0 0 0
0 1 1 0 0
Enter the starting vertex (0 to 4): 0
BFS Traversal: 0 1 2 3 4


=== Code Execution Successful ===
```

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 structures in application development.

Conclusion:

From this experiment, I learned how to use linear and non-linear data structures like queues and graphs to solve problems efficiently. I applied the BFS technique to traverse a graph, which helped me understand how breadth-first traversal works step-by-step.

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