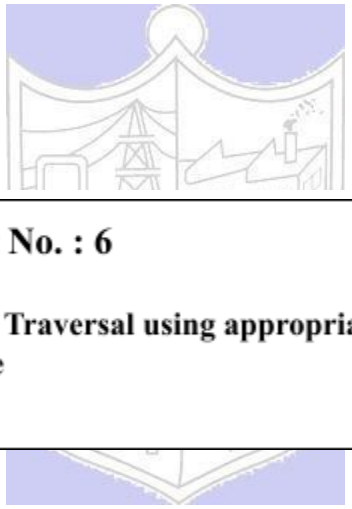


Experiment No. : 6

Title: Graph Traversal using appropriate data structure

Batch:A2 **Roll No.:16010421063** **Experiment No.: 6**

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

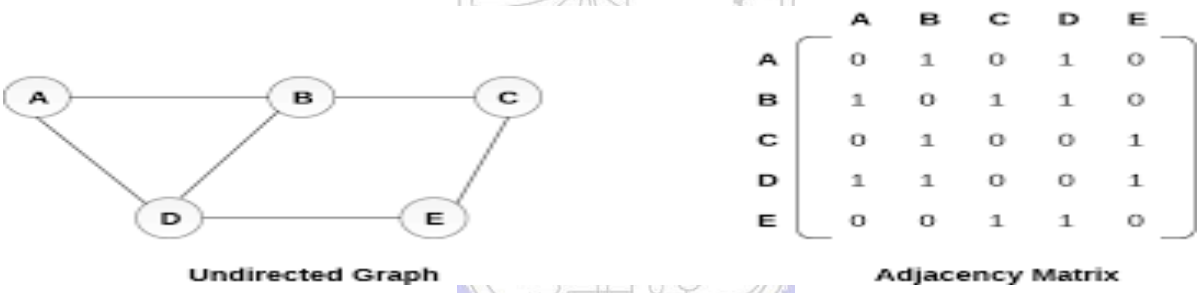
Resources Used: C/ C++ editor and compiler.

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

	A	B	C	D	E
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

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

K. J. SOMAIYA COLLEGE OF ENGG.

Results:

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.

```
#include<stdio.h>
#define SIZE 50


int queue[SIZE], front = -1, rear = -1;
int visited[SIZE];
int vertices;
int matrix[SIZE][SIZE];
```

```
void enQueue(int value){
       if(front == -1)
           front = 0;
       rear++;
       queue[rear] = value;
}

void deQueue(){
   front++;
}

int top(){
   return queue[front];
}

int isNotEmpty(){
   return front<=rear;
}




void bfs(int v)
{
   int i;
   for (i=0;i<vertices;i++)
   {
       if(matrix[v][i] != 0 && visited[i] == 0)
       {
           enQueue(i);
           visited[i]=1;
           printf("%d ",i);
       }
```

```
    }
    deQueue();
    if(isNotEmpty())
    {
        bfs(top());
    }
}



int main(){
    int v,i,j;
    printf("Number of vertices: ");
    scanf("%d",&vertices);
    if(vertices>50){
        printf("Too many vertices");
        return 0;
    }
    printf("Enter the Adjacency matrix: \n");
    for(i=0;i<vertices;i++){
        for(j=0;j<vertices;j++){
            scanf("%d",&matrix[i][j]);
        }
    }
    printf("Source: ");
    scanf("%d",&v);
    enQueue(v);
    printf("\nBFS: \n");
    visited[v]=1;
    printf("%d ",v);
    bfs(v);
    if(rear!=vertices-1){
        printf("Not possible");
    }
```

```
    printf("\n");

}
```

```
→  d:\testing ./a.out
Number of vertices: 4
Enter the Adjacency matrix:
0 1 1 0
1 0 0 1
1 0 0 0
0 1 0 0
Source: 1

BFS:
1 0 3 2
→  d:\testing
```

```
→  d:\testing ./a.out
Number of vertices: 4
Enter the Adjacency matrix:
0 1 1 0
1 0 0 1
1 0 0 0
0 1 0 0
Source: 0

BFS:
0 1 2 3
→  d:\testing ./a.out
```

Outcomes:

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

Conclusion: Successfully understood and 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