Experiment No. : 6

Title: Graph Traversal using appropriate data structure

Batch: B1 **Roll No.: 16010420133** **Experiment No.: 6**

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

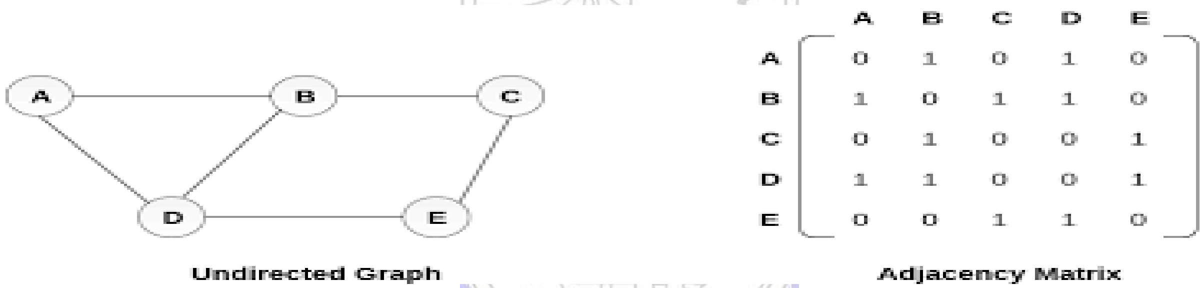
Resources Used: C/ C++ editor and compiler.

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:

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.

Program:

```
#include<stdio.h>

#include<stdlib.h>

#define N 50


int queue[N], front, rear, count, n, adj_mat[50][50], visited[N], stack[N], top;


void menu()
```

```
{
    int choice, start, j;
    printf("\n1. BFS");
    printf("\n2. DFS");
    printf("\n3. Re-enter");
    printf("\n4. Exit");
    printf("\n\nEnter choice (1, 2, 3, 4): ");
    scanf("%d",&choice);
    switch(choice)
    {
        case 1:
            printf("\nEnter starting node: ");
            scanf("%d", &start);
            BFS(start);
            menu();
            break;
        case 2:
            printf("\nEnter starting node: ");
            scanf("%d", &start);
            DFS(start);
            printf("\n");
            menu();
            break;
        case 3:
            createGraph();
            menu();
            break;
        case 4:
```

```
            exit(0);
        default:
            printf("\nInvalid Input!\n");
            menu();
    }
}

void enqueue(int x)
{
    if (count == 0)
    {
        queue[rear] = x;
        count++;
    }
    else
    {
        rear = rear + 1;
        queue[rear] = x;
        count++;
    }
}

void dequeue()
{
    front = front + 1;
    count--;
}
```

```
void push(int x)
{
   top = top + 1;
   stack[top] = x;
}

int pop()
{
   int p;
   p = stack[top];
   top = top - 1;
   return p;
}

int isEmpty_stack()
{
   if(top == -1)
      return 1;
   else
      return 0;
}

void createGraph()
{
   int i, j;
   char c;
   printf("\nEnter number of vertices: ");
   scanf("%d", &n);
```

```
    for(i = 1; i <= n; i++)
    {
        for(j = i; j <= n; j++)
        {
            if(i == j)
            {
                adj_mat[i][j] = 0;
                continue;
            }
            printf("\nVertices %d & %d are Adjacent? (Y/N): ", i, j);
            fflush(stdin);
            scanf("%c", &c);
            if(c == 'y' || c == 'Y')
            {
                adj_mat[i][j] = 1;
                adj_mat[j][i] = 1;
            }
            else
            {
                adj_mat[i][j] = 0;
                adj_mat[j][i] = 0;
            }
        }
    }
}

void BFS(int s)
{
```

```
    int j;
    front = 0;
    rear = 0;
    count = 0;
    visited[0] = 1;
    for (j = 1; j <= n; j++)
    {
        visited[j] = 0;
    }
    visited[s] = 1;
    enqueue(s);
    printf("\nBFS Traversal: ");
    while(count > 0)
    {
        for(j = 1; j <= n; j++)
        {
            if(adj_mat[s][j] == 1 && visited[j] == 0)
            {
                enqueue(j);
                visited[j] = 1;
            }
        }
        printf("%d ", s);
        dequeue();
        s = queue[front];
    }
    printf("\n");
}
```

```
void DFS(int s)
{
    int i, flag;
    top = -1;
    visited[0] = 1;
    for (i = 1; i <= n; i++)
    {
        visited[i] = 0;
    }
    push(s);
    visited[s] = 1;
    printf("\nDFS Traversal: ");
    printf("%d ", s);
    while(!isEmpty_stack())
    {
        flag = 0;
        for(i = 1; i <= n; i++)
        {
            if(adj_mat[s][i] == 1 && visited[i] == 0)
            {
                push(i);
                printf("%d ", i);
                visited[i] = 1;
                s = i;
                flag = 1;
                break;
            }
```

```
        }
        if(flag == 0)
        {
            pop();
            if (top != -1)
                s = stack[top];
            else
                break;
        }
    }
}

int main()
{
    printf("*****************GRAPH TRAVERSAL******************");
    createGraph();
    menu();
    return 0;
}
```

OUTPUT

```
*****************************************************GRAPH TRAVERSAL*****************************************************

Enter number of vertices: 7

Vertices 1 & 2 are Adjacent? (Y/N): y

Vertices 1 & 3 are Adjacent? (Y/N): n

Vertices 1 & 4 are Adjacent? (Y/N): y

Vertices 1 & 5 are Adjacent? (Y/N): n

Vertices 1 & 6 are Adjacent? (Y/N): n

Vertices 1 & 7 are Adjacent? (Y/N): n

Vertices 2 & 3 are Adjacent? (Y/N): y

Vertices 2 & 4 are Adjacent? (Y/N): y

Vertices 2 & 5 are Adjacent? (Y/N): n

Vertices 2 & 6 are Adjacent? (Y/N): y

Vertices 2 & 7 are Adjacent? (Y/N): y

Vertices 3 & 4 are Adjacent? (Y/N): y

Vertices 3 & 5 are Adjacent? (Y/N): y
```

```
Vertices 3 & 5 are Adjacent? (Y/N): y

Vertices 3 & 6 are Adjacent? (Y/N): y

Vertices 3 & 7 are Adjacent? (Y/N): n

Vertices 4 & 5 are Adjacent? (Y/N): y

Vertices 4 & 6 are Adjacent? (Y/N): n

Vertices 4 & 7 are Adjacent? (Y/N): n

Vertices 5 & 6 are Adjacent? (Y/N): n

Vertices 5 & 7 are Adjacent? (Y/N): y

Vertices 6 & 7 are Adjacent? (Y/N): n

1. BFS
2. DFS
3. Re-enter
4. Exit

Enter choice (1, 2, 3, 4):
```

```
1. BFS
2. DFS
3. Re-enter
4. Exit

Enter choice (1, 2, 3, 4): 1

Enter starting node: 4

BFS Traversal: 4 1 2 3 5 6 7

1. BFS
2. DFS
3. Re-enter
4. Exit

Enter choice (1, 2, 3, 4):
```

```
1. BFS
2. DFS
3. Re-enter
4. Exit

Enter choice (1, 2, 3, 4): 2

Enter starting node: 6

DFS Traversal: 6 2 1 4 3 5 7

1. BFS
2. DFS
3. Re-enter
4. Exit

Enter choice (1, 2, 3, 4): 4

Process returned 0 (0x0)   execution time : 102.696 s
Press any key to continue.
```

Outcomes: Concepts of BFS DFS and Graph are understood

Conclusion: BFS is depicted of a graph using a queue with help of an adjacency matrix

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