Assignment 1 (C)

Aim: To illustrate the concept of graph.

Problem Statement :

Represent a given graph using adjacency matrix/list to perform DFS and using adjacencylist to perform BFS. Use the map of the area around the college as the graph. Identify the prominent land marks as nodes and perform DFS and BFS on that.

Learning Objectives:

To understand directed and undirected graph.

To implement program to represent graph using adjacency matrix and list.

Learning Outcome:

Student able to implement program for graph representation.

Theory:

Graph is a data structure that consists of following two components:

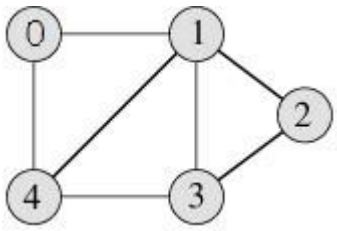
1. A finite set of vertices also called as nodes.

2. A finite set of ordered pair of the form (u, v) called as edge. The pair is ordered because (u, v) is not same as (v, u) in case of directed graph(di-graph). The pair of form (u, v) indicates that there is an edge from vertex u to vertex v. The edges may contain weight/value/cost.

Graphs are used to represent many real life applications: Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network. Graphs are

also used in social networks like linkedIn, facebook. For example, in facebook, each person is represented with a vertex(or node). Each node is a structure and contains information like person id, name, gender and locale. See this for more applications of graph.

Following is an example undirected graph with 5 vertices.



Following two are the most commonly used representations of graph.

1. Adjacency Matrix

2. Adjacency List

There are other representations also like, Incidence Matrix and Incidence List. The choice of the graph representation is situation specific. It totally depends on the type of operations to be performed and ease of use.

Adjacency Matrix:

Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.

The adjacency matrix for the above example graph is:

Adjacency Matrix Representation

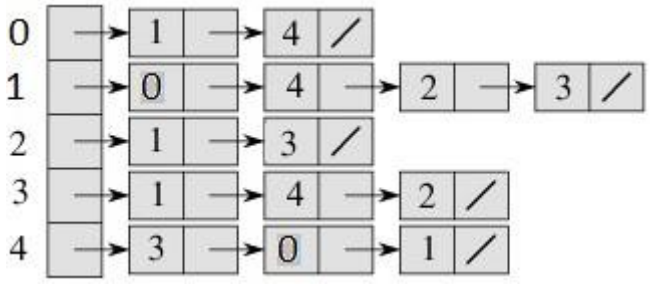
	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0

Pros: Representation is easier to implement and follow. Removing an edge takes O(1) time. Queries like whether there is an edge from vertex ‘u’ to vertex ‘v’ are efficient and can be done O(1).

Cons: Consumes more space O(V^2). Even if the graph is sparse(contains less number of edges), it consumes the same space. Adding a vertex is O(V^2) time.

Adjacency List:

An array of linked lists is used. Size of the array is equal to number of vertices. Let the array be array[]. An entry array[i] represents the linked list of vertices adjacent to the ith vertex. This representation can also be used to represent a weighted graph. The weights of edges can be stored in nodes of linked lists. Following is adjacency list representation of the above graph.



Pros: Saves space O(|V|+|E|) . In the worst case, there can be C(V, 2) number of edges in a graph thus consuming O(V^2) space. Adding a vertex is easier.

Cons: Queries like whether there is an edge from vertex u to vertex v are not efficient and can be done O(V).

Conclusion:

Student implemented program for graph presentation in adjacency matrix and list.

Source Code:

```
#include <iostream>
#include <stdlib.h>
using namespace std;

int cost[10][10], i, j, k, n, qu[10], front, rear, v, visit[10],
visited[10];
int stk[10], top, visit1[10], visited1[10];

int main()
{
    int m;
    cout << "Enter number of vertices : ";
    cin >> n;
    cout << "Enter number of edges : ";
    cin >> m;

    cout << "\nEDGES :\n";
    for (k = 1; k <= m; k++)
    {
        cin >> i >> j;
        cost[i][j] = 1;
        cost[j][i] = 1;
    }

    //display function
    cout << "The adjacency matrix of the graph is : " << endl;
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            cout << " " << cost[i][j];
        }
        cout << endl;
    }

    cout << "Enter initial vertex : ";
    cin >> v;
    cout << "The BFS of the Graph is\n";
    cout << v<<endl;
    visited[v] = 1;
    k = 1;
    while (k < n)
    {
        for (j = 1; j <= n; j++)
            if (cost[v][j] != 0 && visited[j] != 1 && visit[j] != 1)
            {
                visit[j] = 1;
```

```
                qu[rear++] = j;
            }
        v = qu[front++];
        cout << v << " ";
        k++;
        visit[v] = 0;
        visited[v] = 1;
    }

    cout <<endl<<"Enter initial vertex : ";
    cin >> v;
    cout << "The DFS of the Graph is\n";
    cout << v<<endl;
    visited[v] = 1;
    k = 1;
    while (k < n)
    {
        for (j = n; j >= 1; j--)
            if (cost[v][j] != 0 && visited1[j] != 1 && visit1[j] != 1)
            {
                visit1[j] = 1;
                stk[top] = j;
                top++;
            }
        v = stk[--top];
        cout << v << " ";
        k++;
        visit1[v] = 0;
        visited1[v] = 1;
    }

    return 0;
}
```