Pune Institute of Computer Technology, Pune
Department of Computer Engineering

Sub - DSAL Date:- 26/05/2020

Name : - Shrikrushna Santosh Zirape
Roll No : - 21286

Assignment - 06

Problem Statement :-

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

Learning Objective :-

1. To understand directed and undirected graphs.
2. To implement a program to represent a graph using adjacency matrix and list.

Learning Outcome :-

Students are able to implement programs for graph representation.

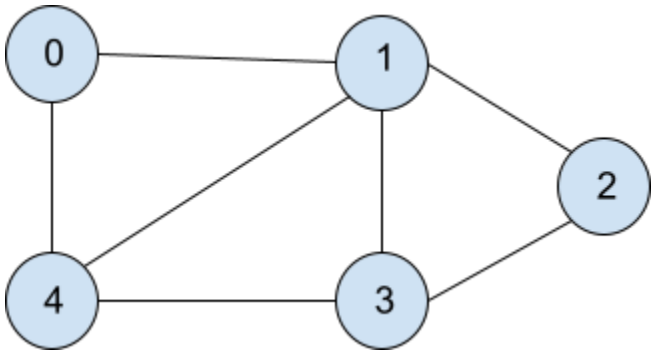
Theory:

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 pairs of the form (u, v) called edge. The pair is ordered because (u, v) is not the same as (v, u) in case of directed graphs(di-graph). The pair of forms (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 an 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).

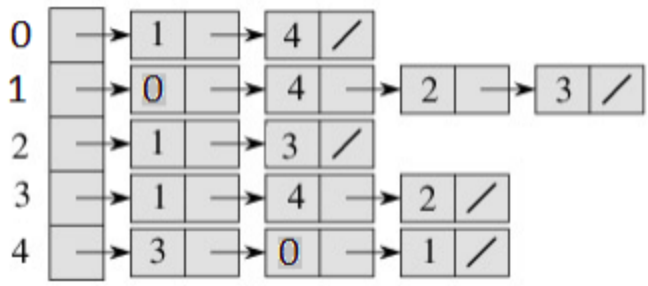
Cons:

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 the 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).

Operations on Graph:-

1. DFS:-

Algorithm:-

Procedure DFS(v)
//given an undirected graph G = (v,e) with n vertices and
An array Visited(n) initially set to zero this algorithm visits all vartices Reachable from v.G and visited are global.

For each vartex w in adjacent to v do
If visited(w) = 0 then call DFS(w)
End
endDFS

2. BFS:-

It starting at a vertex v and marking it as visited
All unvisited vertices adjacent to v are visited next.
Then unvisited verties adjacent to these vertices are visited and so on.

Algorithm
Procedure BFS(v)
// a breadth first search of G is carried out beginning at vertex v. all vertices visited are marked as visited(i)=1 the graph G and array visited are global and visited is initialized to zero //

visited(v) :=1
Loop
For all vertices w adjacent to v do
If Visited(w) = 0 // add w to queue //
Then ADDQ(w,q) ; //visited(w) :=1 //mark w as visited
End
If Q is empty then return
Call DELETEQ(v,q)
Forever
End BFS

Complexity:-

BFS - O(v+e)
DFS- O(v+e)

Test Cases

SR No	Test Input	Exp Output	Act Output	Result
1	No of vertices : - 6 No of edges :- 8 For 1st edge: s=0 d=1 For 2nd edge : s=1 d=3 For 3rd edge : s=3 d=5 For 4th edge : s=4 d=5 For 5th edge : s=2 d=4 For 6th edge : s=0 d=2 For 7th edge : s=3 d=4 For 8th edge : s=1 d=4 Display table Display DFS Display BFS	Table display :- 0 -> 1 - 2 1 -> 0 - 3 - 4 2 -> 4 - 0 3 -> 1 - 5 - 4 4 -> 5 - 2 - 3 - 1 5 -> 3 - 4 DFS:- 0 1 3 5 4 2 BFS :- 0 1 2 3 4 5	Table display :- 0 -> 1 - 2 1 -> 0 - 3 - 4 2 -> 4 - 0 3 -> 1 - 5 - 4 4 -> 5 - 2 - 3 - 1 5 -> 3 - 4 DFS:- 0 1 3 5 4 2 BFS :- 0 1 2 3 4 5	Pass

Conclusion :

We implemented a program for graph presentation in adjacency matrix and list.