	Bansilal Ramnath Agarwal Charitable Trust's Vishwakarma Institute of Information Technology Department of Artificial Intelligence and Data Science

Name: Siddhesh Dilip Khairnar		
Class: SY-B tech	Division: B	Roll No: 272028
Semester: 3rd	Academic Year:2022-2023	
Subject Name & Code: ES21201AD: Discrete Mathematics		
Title of Assignment: Program to calculate Indegree and outdegree of node		
Date of Performance: 28/11/2022	Date of Submission: 05/12/2022	

Problem Statement: Write a program to calculate Indegree and outdegree of node in directed and undirected graph with adjacency matrix.

Introduction to Graph Data Structure: Graphs in data structures are non-linear data structures made up of a finite number of nodes or vertices and the edges that connect them. Graphs in data structures are used to address real-world problems in which it represents the problem area as a network like telephone networks, circuit networks, and social networks.

Graphs in data structures are used to represent the relationships between objects. Every graph consists of a set of points known as vertices or nodes connected by lines known as edges. The vertices in a network represent entities. The most frequent graph representations are the two that follow:

- Adjacency matrix
- Adjacency list

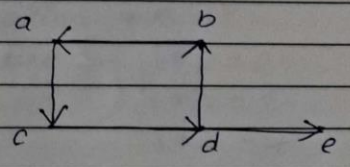
Types of Graph Data Structure:

- Finite Graph
- Infinite Graph
- Trivial Graph
- Simple Graph
- Multi Graph
- Null Graph
- Complete Graph
- Pseudo Graph
- Regular Graph
- Weighted Graph, etc.

Tutorial 7

Problem Statement -
Write a program to calculate in degree & out-degree of node in directed & undirected graph with adjacency matrix

Introduction to graph structure -
A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by points termed as vertices, & the links that connect the vertices are called edges.
Formally, a graph is a pair of sets (V,E), where V is the set of vertices & E is the set of edges, connecting the pairs of vertices.



$V = \{a, b, c, d, e\}$
$E = \{ab, ac, bd, cd, de\}$

Degree of a vertex in a directed graph -
In case of directed graph, the in-degree, out-degree of a vertex can be determined by the following steps:

* In-degree -
It can be written by deg-(V), is number of edges with V as the terminated vertex. To find the in-degree, just count the no. of edges ends at the vertex

* Out-degree -
It can be written by $deg^+(V)$, is the no. of edges with V it's the initial vertex. To find out-degree, just count the no. of edges starting from vertex.

Page :
Date : / /

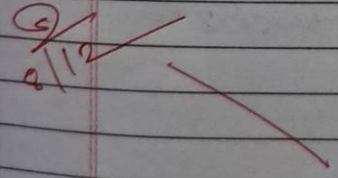
In-degree	Out-degree
deg(a) = 1	deg(a) = 1
deg(b) = 1	deg(b) = 1
deg(c) = 1	deg(c) = 1
deg(d) = 1	deg(d) = 2
deg(e) = 1	deg(e) = 0

Algorithm-

* Traverse adjacency list for every vertex, if size of the adjacency list of vertex i is x, then the out degree for i=x.
* Increment the in degree of every degree that has an incoming edge from i
* Repeat the steps for every vertex & print the in & out degrees for all vertices in the end.

Conclusion :-

Thus, we calculated in degree & out degree of node in directed & undirected graph with adjacency matrix using program.



Program Input:

```cpp
// C++ program to find the in and out degrees
// of the vertices of the given graph
#include <bits/stdc++.h>
using namespace std;

// Function to print the in and out degrees
// of all the vertices of the given graph
void findInOutDegree(vector<vector<int>> adjlist,
                     int n)
{
    vector<int> iN(n, 0);
    vector<int> ouT(n, 0);

    for (int i = 0; i < n; i++)
    {
        // Out degree for ith vertex will be the count
        // of direct paths from i to other vertices
        ouT[i] = adjlist[i].size();
        // Every vertex that has an incoming
        // edge from i
        for (int j = 0; j < adjlist[i].size(); j++)
            iN[adjlist[i][j]]++;
    }

    cout << "Vertex\t\tIn\t\tOut" << endl;
    for (int k = 0; k < n; k++)
    {
        cout << k << "\t\t"
             << iN[k] << "\t\t"
             << ouT[k] << endl;
    }
}
```

```cpp
// Driver code
int main()
{

    // Adjacency list representation of the graph
    vector<vector<int>> adjlist;

    // Vertices 1 and 2 have an incoming edge
    // from vertex 0
    vector<int> tmp;
    tmp.push_back(1);
    tmp.push_back(2);
    adjlist.push_back(tmp);
    tmp.clear();

    // Vertex 3 has an incoming edge
    // from vertex 1
    tmp.push_back(3);
    adjlist.push_back(tmp);
    tmp.clear();

    // Vertices 0, 5 and 6 have an incoming
    // edge from vertex 2
    tmp.push_back(0);
    tmp.push_back(5);
    tmp.push_back(6);
    adjlist.push_back(tmp);
    tmp.clear();

    // Vertices 1 and 4 have an incoming
    // edge from vertex 3
    tmp.push_back(1);
    tmp.push_back(4);
    adjlist.push_back(tmp);
    tmp.clear();

    // Vertices 2 and 3 have an incoming
    // edge from vertex 4
    tmp.push_back(2);
    tmp.push_back(3);
    adjlist.push_back(tmp);
    tmp.clear();

    // Vertices 4 and 6 have an incoming
    // edge from vertex 5
    tmp.push_back(4);
    tmp.push_back(6);
    adjlist.push_back(tmp);
    tmp.clear();

    // Vertex 5 has an incoming
    // edge from vertex 6
    tmp.push_back(5);
    adjlist.push_back(tmp);
    tmp.clear();

    int n = adjlist.size();

    findInOutDegree(adjlist, n);
}
```

Program Output:

```
PS C:\Program language\C++> cd "c:\Program language\C++\" ; if ($?) { g++ tut2.cpp -o tut2 } ; if ($?) { .\tut
2 }
Vertex          In              Out
0               1               2
1               2               1
2               2               3
3               2               2
4               2               2
5               2               2
6               2               1
PS C:\Program language\C++>
```