**CHAROTAR UNIVERSITY OF SCIENCE &**

**TECHNOLOGY**

**DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY & RESEARCH**

**Computer Science & Engineering**

**NAME: PARTH NITESHKUMAR PATEL**

**ID: 19DCS098**

**SUBJECT: DESIGN AND ANALYSIS OF**

**ALGORITHM**

**CODE: CS 351**

**GRAPH**

**PRACTICAL-6.1**

**AIM:**

Write a program to detect cycles in an directed graph.

**PROGRAM CODE:**

#include <iostream>

#include <list>

#include <limits.h>

using namespace std;

class Graph

{

    int V;

    list<int> \*adj;

    bool isCyclicUtil(int v, bool visited[], bool \*rs);

public:

    Graph(int V);

    void addEdge(int v, int w);

    bool isCyclic();

};

Graph::Graph(int V)

{

    this->V = V;

    adj = new list<int>[V];

}

void Graph::addEdge(int v, int w)

{

    adj[v].push\_back(w);

}

bool Graph::isCyclicUtil(int v, bool visited[], bool \*recStack)

{

    if (visited[v] == false)

    {

        visited[v] = true;

        recStack[v] = true;

        list<int>::iterator i;

        for (i = adj[v].begin(); i != adj[v].end(); ++i)

        {

            if (!visited[\*i] && isCyclicUtil(\*i, visited, recStack))

                return true;

            else if (recStack[\*i])

                return true;

        }

    }

    recStack[v] = false;

    return false;

}

bool Graph::isCyclic()

{

    bool \*visited = new bool[V];

    bool \*recStack = new bool[V];

    for (int i = 0; i < V; i++)

    {

        visited[i] = false;

        recStack[i] = false;

    }

    for (int i = 0; i < V; i++)

        if (isCyclicUtil(i, visited, recStack))

            return true;

    return false;

}

int main()

{

    Graph g(4);

    g.addEdge(0, 1);

    g.addEdge(1, 2);

    g.addEdge(1, 2);

    g.addEdge(2, 0);

    g.addEdge(2, 3);

    g.addEdge(3, 3);

    if (g.isCyclic())

        cout << "GRAPH CONTAINS CYCLE";

    else

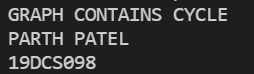
        cout << "GRAPH DOES NOT CONTAIN CYCLE";

    cout << "\nPARTH PATEL\n19DCS098" << endl;

    return 0;

}

**OUTPUT:**



**CONCLUSION:**

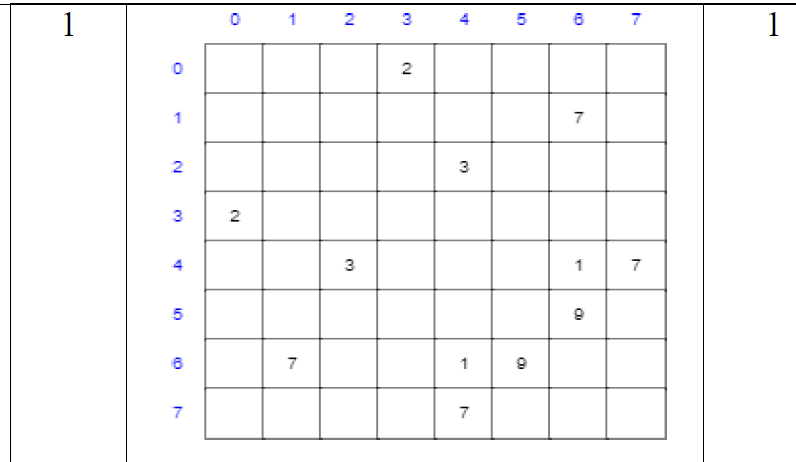
* DFS for a connected graph produces a tree.
* Time Complexity: **O(V + E)**
* Depth First Traversal can be used to detect a cycle in a Graph.

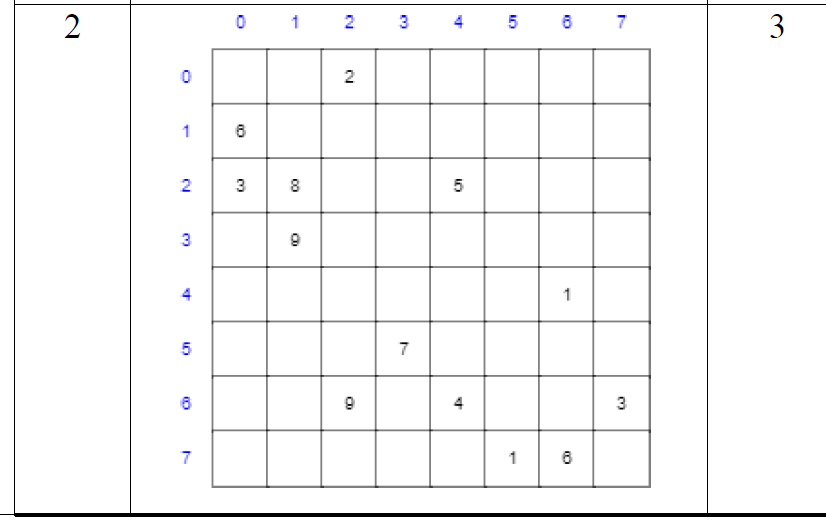
**PRACTICAL-6.2**

**AIM:**

From a given vertex in a weighted graph, implement a program to find shortest paths to other vertices using Dijkstra’s algorithm.







**PROGRAM CODE:**

#include <iostream>

#define INFINITY 10000

#define MAX 10

using namespace std;

void dijikstraAlgorithm(int G[MAX][MAX], int n, int startnode)

{

    int cost[MAX][MAX], distance[MAX], pred[MAX];

    int visited[MAX], count, mindistance, nextnode, i, j;

    for (i = 0; i < n; i++)

        for (j = 0; j < n; j++)

            if (G[i][j] == 0)

                cost[i][j] = INFINITY;

            else

                cost[i][j] = G[i][j];

    for (i = 0; i < n; i++)

    {

        distance[i] = cost[startnode][i];

        pred[i] = startnode;

        visited[i] = 0;

    }

    distance[startnode] = 0;

    visited[startnode] = 1;

    count = 1;

    while (count < n - 1)

    {

        mindistance = INFINITY;

        for (i = 0; i < n; i++)

            if (distance[i] < mindistance && !visited[i])

            {

                mindistance = distance[i];

                nextnode = i;

            }

        visited[nextnode] = 1;

        for (i = 0; i < n; i++)

            if (!visited[i])

                if (mindistance + cost[nextnode][i] < distance[i])

                {

                    distance[i] = mindistance + cost[nextnode][i];

                    pred[i] = nextnode;

                }

        count++;

    }

    for (i = 0; i < n; i++)

        if (i != startnode)

        {

            cout<<"\nDISTANCE OF The NODE "<<i<<" : "<<distance[i];

            cout<<"\nPATH : "<<i;

            j = i;

            do

            {

                j = pred[j];

                printf("<-%d", j);

            } while (j != startnode);

        }

}

int main()

{

    int G[MAX][MAX], i, j, n, u;

    cout<<"ENTER THE NUMBER OF VERTICES : ";

    cin>>n;

    cout<<"\nENTER THE ADJACENCY MATRIX : "<<endl;

    for (i = 0; i < n; i++)

        for (j = 0; j < n; j++)

            cin>>G[i][j];

    cout<<"\nENTER THE STARTING NODE : ";

    cin>>u;

    dijikstraAlgorithm(G, n, u);

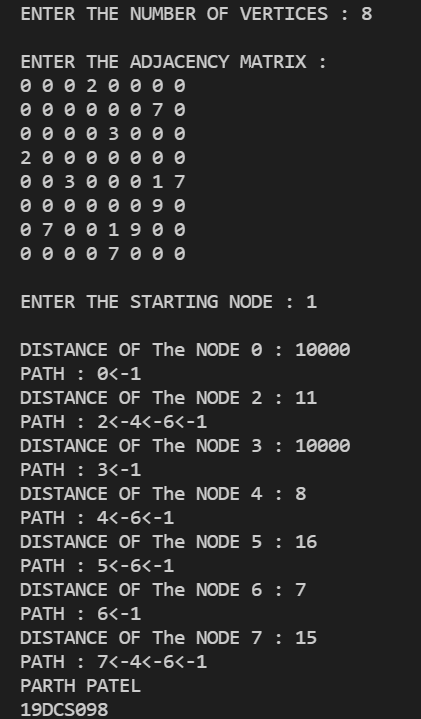
    cout<<"\nPARTH PATEL\n19DCS098";

    return 0;

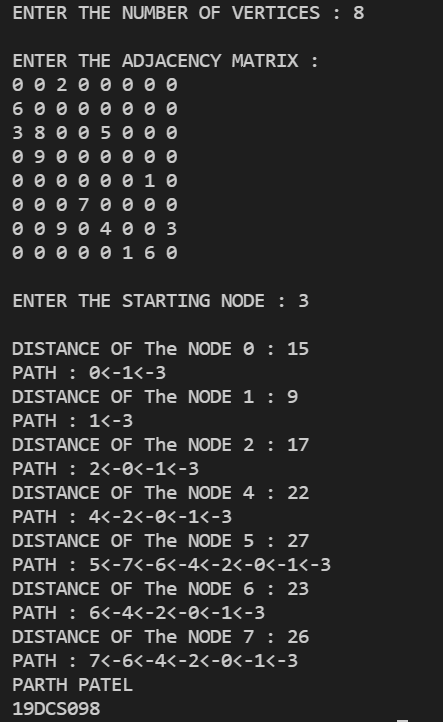
}

**OUTPUT:**

**Test Case-1:**



**Test Case-2:**



**CONCLUSION:**

* Dijkstra’s algorithm is very similar to Prim’s algorithm for minimum spanning tree.
* Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph
* Complexity: **O(ElogV)**
* It is also known as **SINGLE SOURCE SHORTEST PATH ALGORITHM**