**AIM :-**Write a program to implement Bellman Ford Algorithm and analyze its complexity.

**SOURCE CODE :**

// A C++ program for Bellman-Ford's single source

// shortest path algorithm.

#include <bits/stdc++.h>

// a structure to represent a weighted edge in graph

struct Edge

{

    int src, dest, weight;

};

// a structure to represent a connected, directed and

// weighted graph

struct Graph

{

    // V-> Number of vertices, E-> Number of edges

    int V, E;

    // graph is represented as an array of edges.

    struct Edge\* edge;

};

// Creates a graph with V vertices and E edges

struct Graph\* createGraph(int V, int E)

{

    struct Graph\* graph = new Graph;

    graph->V = V;

    graph->E = E;

    graph->edge = new Edge[E];

    return graph;

}

// A utility function used to print the solution

void printArr(int dist[], int n)

{

    printf("Vertex   Distance from Source\n");

    for (int i = 0; i < n; ++i)

        printf("%d \t\t %d\n", i, dist[i]);

}

// The main function that finds shortest distances from src to

// all other vertices using Bellman-Ford algorithm.  The function

// also detects negative weight cycle

void BellmanFord(struct Graph\* graph, int src)

{

    int V = graph->V;

    int E = graph->E;

    int dist[V];

    // Step 1: Initialize distances from src to all other vertices

    // as INFINITE

    for (int i = 0; i < V; i++)

        dist[i]   = INT\_MAX;

    dist[src] = 0;

    // Step 2: Relax all edges |V| - 1 times. A simple shortest

    // path from src to any other vertex can have at-most |V| - 1

    // edges

    for (int i = 1; i <= V-1; i++)

    {

        for (int j = 0; j < E; j++)

        {

            int u = graph->edge[j].src;

            int v = graph->edge[j].dest;

            int weight = graph->edge[j].weight;

            if (dist[u] != INT\_MAX && dist[u] + weight < dist[v])

                dist[v] = dist[u] + weight;

        }

    }

    // Step 3: check for negative-weight cycles.  The above step

    // guarantees shortest distances if graph doesn't contain

    // negative weight cycle.  If we get a shorter path, then there

    // is a cycle.

    for (int i = 0; i < E; i++)

    {

        int u = graph->edge[i].src;

        int v = graph->edge[i].dest;

        int weight = graph->edge[i].weight;

        if (dist[u] != INT\_MAX && dist[u] + weight < dist[v])

            printf("Graph contains negative weight cycle");

    }

    printArr(dist, V);

    return;

}

// Driver program to test above functions

int main()

{

    /\* Let us create the graph given in above example \*/

    int V = 5;  // Number of vertices in graph

    int E = 8;  // Number of edges in graph

    struct Graph\* graph = createGraph(V, E);

    // add edge 0-1 (or A-B in above figure)

    graph->edge[0].src = 0;

    graph->edge[0].dest = 1;

    graph->edge[0].weight = -1;

    // add edge 0-2 (or A-C in above figure)

    graph->edge[1].src = 0;

    graph->edge[1].dest = 2;

    graph->edge[1].weight = 4;

    // add edge 1-2 (or B-C in above figure)

    graph->edge[2].src = 1;

    graph->edge[2].dest = 2;

    graph->edge[2].weight = 3;

    // add edge 1-3 (or B-D in above figure)

    graph->edge[3].src = 1;

    graph->edge[3].dest = 3;

    graph->edge[3].weight = 2;

    // add edge 1-4 (or A-E in above figure)

    graph->edge[4].src = 1;

    graph->edge[4].dest = 4;

    graph->edge[4].weight = 2;

    // add edge 3-2 (or D-C in above figure)

    graph->edge[5].src = 3;

    graph->edge[5].dest = 2;

    graph->edge[5].weight = 5;

    // add edge 3-1 (or D-B in above figure)

    graph->edge[6].src = 3;

    graph->edge[6].dest = 1;

    graph->edge[6].weight = 1;

    // add edge 4-3 (or E-D in above figure)

    graph->edge[7].src = 4;

    graph->edge[7].dest = 3;

    graph->edge[7].weight = -3;

    BellmanFord(graph, 0);

    return 0;

}