**9913 Mark Lopes**

**SE Comps A Batch-C**

**Bellman Ford:-**

#include <stdio.h>

#include <stdlib.h>

#define INF 999

typedef struct

{

    int u, v, w; // u=start, v=end, w=weight

} Edge;

typedef struct

{

    int V, E;   // vertices, edges

    Edge \*edge; // array of edges

} Graph;

void display(int array[], int size)

{

    for (int i = 0; i < size; i++)

    {

        printf("%d ", array[i]);

    }

    printf("\n");

}

void bellmanFord(Graph \*g1, int source)

{

    int totalVertex = g1->V;

    int totalEdge = g1->E;

    int distance[totalVertex];

    int predecessor[totalVertex];

    // Initialize distances and predecessors

    for (int i = 0; i < totalVertex; i++)

    {

        distance[i] = INF;

        predecessor[i] = -1;

    }

    distance[source] = 0;

    // Relax edges (V-1 times)

    for (int i = 0; i < totalVertex - 1; i++)

    {

        for (int j = 0; j < totalEdge; j++)

        {

            int u = g1->edge[j].u;

            int v = g1->edge[j].v;

            int w = g1->edge[j].w;

            if (distance[u] != INF && distance[v] > distance[u] + w)

            {

                distance[v] = distance[u] + w;

                predecessor[v] = u;

            }

        }

    }

    // Check for negative weight cycles

    for (int i = 0; i < totalEdge; i++)

    {

        int u = g1->edge[i].u;

        int v = g1->edge[i].v;

        int w = g1->edge[i].w;

        if (distance[u] != INF && distance[v] > distance[u] + w)

        {

            printf("Negative weight cycle detected!\n");

            return;

        }

    }

    // Display results

    printf("Distance array: ");

    display(distance, totalVertex);

    printf("Predecessor array: ");

    display(predecessor, totalVertex);

}

int main()

{

    Graph \*g = (Graph \*)malloc(sizeof(Graph));

    g->V = 4;

    g->E = 5;

    g->edge = (Edge \*)malloc(g->E \* sizeof(Edge));

    // Initialize edges

    g->edge[0].u = 0;

    g->edge[0].v = 1;

    g->edge[0].w = 4;

    g->edge[1].u = 0;

    g->edge[1].v = 2;

    g->edge[1].w = 5;

    g->edge[2].u = 1;

    g->edge[2].v = 2;

    g->edge[2].w = -2;

    g->edge[3].u = 1;

    g->edge[3].v = 3;

    g->edge[3].w = 6;

    g->edge[4].u = 2;

    g->edge[4].v = 3;

    g->edge[4].w = 1;

    bellmanFord(g, 0);

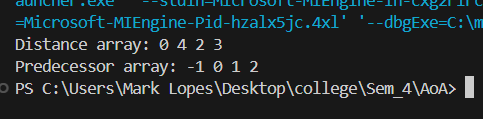
    // Free dynamically allocated memory

    free(g->edge);

    free(g);

    return 0;

}

****

