**Submitted by :**

**Diwanshu Karwal**

**16csu111**

**Bellman Ford's Algorithm Code**

**Algorithm:**

Following are the detailed steps

.

Input: Graph and a source vertex src  
Output: Shortest distance to all vertices from src. If there is a negative weight cycle, then shortest distances are not calculated, negative weight cycle is reported.

**1)** This step initializes distances from source to all vertices as infinite and distance to source itself as 0. Create an array dist[] of size |V| with all values as infinite except dist[src] where src is source vertex.

**2)** This step calculates shortest distances. Do following |V|-1 times where |V| is the number of vertices in given graph.  
…..**a)** Do following for each edge u-v  
………………If dist[v] > dist[u] + weight of edge uv, then update dist[v]  
………………….dist[v] = dist[u] + weight of edge uv

**3)** This step reports if there is a negative weight cycle in graph. Do following for each edge u-v  
……If dist[v] > dist[u] + weight of edge uv, then “Graph contains negative weight cycle”  
The idea of step 3 is, step 2 guarantees shortest distances if graph doesn’t contain negative weight cycle. If we iterate through all edges one more time and get a shorter path for any vertex, then there is a negative weight cycle

**Code :**

#include <stdio.h>

#include <stdlib.h>

#define INFINITY 99999

struct Edge {

int u;

int v;

int w; };

struct Graph {

int V;

int E;

struct Edge \*edge;

};

void bellmanford(struct Graph \*g, int source);

void display(int arr[], int size);

int main(void) {

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

g->V = 4;

g->E = 5;

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

g->edge[0].u = 0;

g->edge[0].v = 1;

g->edge[0].w = 5;

g->edge[1].u = 0;

g->edge[1].v = 2;

g->edge[1].w = 4;

g->edge[2].u = 1;

g->edge[2].v = 3;

g->edge[2].w = 3;

g->edge[3].u = 2;

g->edge[3].v = 1;

g->edge[3].w = -6;

g->edge[4].u = 3;

g->edge[4].v = 2;

g->edge[4].w = 2;

bellmanford(g, 0);

return 0;

}

void bellmanford(struct Graph \*g, int source) {

int i, j, u, v, w;

int tV = g->V;

int tE = g->E;

int d[tV];

int p[tV];

for (i = 0; i < tV; i++) {

d[i] = INFINITY;

p[i] = 0;

}

d[source] = 0;

for(i = 1; i <= tV-1; i++) {

for(j = 0; j < tE; j++) {

u = g->edge[j].u;

v = g->edge[j].v;

w = g->edge[j].w;

if(d[u] != INFINITY && d[v] > d[u] + w) {

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

p[v] = u;

}

}

}

for(i = 0; i < tE; i++) {

u = g->edge[i].u;

v = g->edge[i].v;

w = g->edge[i].w;

if(d[u] != INFINITY && d[v] > d[u] + w) {

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

return;

}

}

printf("Distance array: ");

display(d, tV);

printf("Predecessor array: ");

display(p, tV);

}

void display(int arr[], int size) {

int i;

for(i = 0; i < size; i ++) {

printf("%d ", arr[i]);

}

printf("\n");

}