Bellman-Ford

```
#include <iostream>
#include <stdlib.h>
#include <string.h>
#include <limits.h>

using namespace std;

struct Edge{
    int source, destination, weight;
};

struct Graph{
    int V, E;
    struct Edge* edge;
};

struct Graph* createGraph(int V, int E){
    struct Graph* graph = (struct Graph*) malloc( sizeof(struct Graph));

    graph->V = V;
    graph->E = E;
    graph->edge = (struct Edge*) malloc( graph->E * sizeof( struct Edge ) );

    return graph;
}

void FinalSolution(int dist[], int n){
    cout<<"\nVertex\tDistance from Source Vertex\n";
    for (int i = 0; i < n; ++i)
        cout<<i<<"\t\t"<<dist[i]<<"\n";
}

void BellmanFord(struct Graph* graph, int source){
    int V = graph->V;
    int E = graph->E;
    int StoreDistance[V];

    for (int i = 0; i < V; i++)
        StoreDistance[i] = INT_MAX;

    StoreDistance[source] = 0;

    for (int i = 1; i <= V-1; i++){
```

```
        for (int j = 0; j < E; j++){
            int u = graph->edge[j].source;
            int v = graph->edge[j].destination;
            int weight = graph->edge[j].weight;

            if (StoreDistance[u] + weight < StoreDistance[v])
                StoreDistance[v] = StoreDistance[u] + weight;
        }
    }

    for (int i = 0; i < E; i++){
        int u = graph->edge[i].source;
        int v = graph->edge[i].destination;
        int weight = graph->edge[i].weight;

        if (StoreDistance[u] + weight < StoreDistance[v])
            cout<<"\nThis graph contains negative edge cycle\n";
    }

    FinalSolution(StoreDistance, V);

    return;
}

int main(){
    int V,E,S;
    cin >> V >> E >> S;

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

    int i;
    for(i=0;i<E;i++){
        cin>>graph->edge[i].source;
        cin>>graph->edge[i].destination;
        cin>>graph->edge[i].weight;
    }

    BellmanFord(graph, S);
    return 0;
}
```