## Shortest-Path-in-a-Directed-Acyclic-Graph

```
#include<iostream>
#include <list>
#include <stack>
#include <limits.h>
#define INF INT_MAX
using namespace std;

class AdjListNode{
    int v;
    int weight;

    public:
    AdjListNode(int _v, int _w) { v = _v; weight = _w;}
    int getV()   { return v; }
    int getWeight() { return weight; }
};

class Graph{
    int V;
    list<AdjListNode> *adj;
    void topologicalSortUtil(int v, bool visited[], stack<int> &Stack);

    public:
    Graph(int V);
    void addEdge(int u, int v, int weight);
    void shortestPath(int s);
};

Graph::Graph(int V){
    this->V = V;
    adj = new list<AdjListNode>[V];
}

void Graph::addEdge(int u, int v, int weight){
    AdjListNode node(v, weight);
    adj[u].push_back(node);
}

void Graph::topologicalSortUtil(int v, bool visited[], stack<int> &Stack){
    visited[v] = true;

    list<AdjListNode>::iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); ++i){
        AdjListNode node = *i;
        if (!visited[node.getV()])
            topologicalSortUtil(node.getV(), visited, Stack);
    }

    Stack.push(v);
}

void Graph::shortestPath(int s){
    stack<int> Stack;
    int dist[V];

    bool *visited = new bool[V];
    for (int i = 0; i < V; i++)
        visited[i] = false;

    for (int i = 0; i < V; i++)
        if (visited[i] == false)
            topologicalSortUtil(i, visited, Stack);

    for (int i = 0; i < V; i++)
        dist[i] = INF;
    dist[s] = 0;

    while (Stack.empty() == false){
        int u = Stack.top();
        Stack.pop();

        list<AdjListNode>::iterator i;
        if (dist[u] != INF)
        {
        for (i = adj[u].begin(); i != adj[u].end(); ++i)
            if (dist[i->getV()] > dist[u] + i->getWeight())
                dist[i->getV()] = dist[u] + i->getWeight();
        }
    }

    for (int i = 0; i < V; i++)
        (dist[i] == INF)? cout << "INF ": cout << dist[i] << " ";
}

int main(){
    Graph g(6);
    g.addEdge(0, 1, 5);
    g.addEdge(0, 2, 3);
    g.addEdge(1, 3, 6);
    g.addEdge(1, 2, 2);
    g.addEdge(2, 4, 4);
    g.addEdge(2, 5, 2);
    g.addEdge(2, 3, 7);
    g.addEdge(3, 4, -1);
    g.addEdge(4, 5, -2);

    int s = 1;
    cout << "Following are shortest distances from source " << s <<" n";
    g.shortestPath(s);

    return 0;
}
```