# Project 8 Part a

## Group Members

Abhishek Gupta (axg137230) and Manuj Singh (mxs135630)

## Objective

To implement an algorithm to find out number of shortest paths from a given source node to a destination node.

## Problem Statement

Given a directed graph G with V number of vertices and E number of edges. For a given source vertex S, and a destination vertex T, we need to find total number of distinct shortest paths from S to T. Program can raise an exception and terminate if it finds a non-positive cycle.

## Implementation

We have used the improved version of bellman ford algorithm from the class notes to find the shortest paths and tweaked the relax function a little bit to achieve our objective.

### Abstract Data Type of Node

A vertex or a node in the graph contains following properties:

* dist: holds the distance of this node from the source node S.
* prev: holds the pointer to the previous node in the shortest path from the source.
* noOfPaths: holds the number of shortest paths from source node S to this node.
* count: holds the number of times the dist attribute of this node is changed.
* adj: holds the list of outgoing edges from this node.

### Abstract Data Type of Edge

A directed edge between two nodes contains the following properties:

* u: holds starting node of an edge.
* v: holds ending node of an edge.
* w: holds the weight of an edge.

### Abstract Data Type of Graph

A graph contains following properties:

* V: holds number of vertices in the graph.
* Node[]: holds all the vertices in the graph.

### Algorithm

The algorithms used is the improved version of bellman ford to find shortest path tweaking relax and initialize methods and adding some more attributes to the node. Below is the pseudo code of all the various procedures.

Assuming all the graph nodes are globally accessible.

**Initialize ()**

**// Initialize each node with default properties and source node with // dist = 0 and noOfPaths =1**

for each vertex u in V do

u.dist = INFINITY;

u.prev = NULL;

u.noOfPaths = 0;

u.count = 0;

u.adj = new List<Edge> ();

s.dist = 0;

s.noOfPaths = 1;

**Relax (u, v)**

**// Relax each edge and updates node properties if needed.**

**// Returns TRUE if distance to v is updated else FALSE.**

if (v.dist > u.dist + w(u,v)) then

v.dist = u.dist + w(u,v);

v.prev = u;

v.noOfPaths = u.noOfPaths;

return TRUE;

if (v.dist = u.dist + w(u,v)) then

if (u != v.prev) then

// Different shortest path is found.

v.noOfPaths += u.noOfPaths;

return FALSE;

**FindShortestPaths ()**

**// Find shortest paths from Source node S to every other node in the graph.**

Queue Q = new Queue<Nodes> () // Holds the nodes whose dist has been changed.

Q.enqueue (S);

While (Q is not Empty ()) do

u <- Q.dequeue();

if (u.count++ > |V|) then

raise exception “Non-positive cycle in the graph”;

for each edge (u,v) in u.adj do

if (relax (u,v) && Q not contains (v)) then

Q.enqueue(v);

Return;

### Conclusion

At the end, **T.dist** contains the distance of destination node from the source node and **T.noOfPaths** contains the number of shortest paths from the source node S to destination T.