Given data, design, and assumption:

1. The number of actors, they are the vertices in graph terms.
2. Each actor has direct invitation cost, and if an actor invites another actor, the cost to invite both actors is a1 + a2 \* (1- w(a1,a2)), where w(a1,a2) is the degree the two actors knows each other. The degree values are given and each actor knows each other to certain degree. This is modeled in the implementation as weighted edge, weight is the invitation cost, and the edge is bidirectional. The graph is stored as an adjacency matrix; three actors is a 3 x 3 matrix, and four actors is a 4 x 4 matrix, as on so.
3. In this implementation, the starting action is always first actor, or 1, even though the program supports starting from different actor. The starting actor can’t be changed unless changing source code.
4. The implementation is based on Dijkstra’s algorithm. The final output suits the expected result.

The flow of the program

1. Reads the input.txt, and parse the data to get the number of actors which are implemented as vertices of the graph; direct invitation cost and indirect invitation matrix (or acquaint degree), they form the weight of the edge in the graph; Invitation Cost object array is created to store the minimum invitation cost which is basically the product of Dijkstra’s algorithm
2. Pseudo code of Dijkstra’s algorithm [1]. There are major two core functions: findminimumCost() and calculateCost()

**findMinimumCost(int startActor)**

for each actor

initialize and store element in invitaCost array

for each actor

// get the actor with minimum cost in invitaCost array

currActor = getLeastCost()

//mark the actor with minimum as invited

actorList[currActor].invited = true

//call method to calculate the minimum cost of each actor

//this method is the core of entire Dijkstra’s algorithm

calculateCost();

**calculateCost();**

for each actor

if actorList[otherActor] is not invited

degree = acqMatrix[currActor][otherActor]

//calculate the total cost of invitation from current actor to his

//friend actor

iaskOtherCost = getiaskCost(oCost, degree) + accmuCost;

//update adjacency actor cost to lower cost if necessary

//this is the core logic to find and save minimum cost

if iaskOtherCost < invitaCost[otherActor].cost

invitaCost[otherActor].srcInvite=currActor

invitaCost[otherActor].cost = iaskotherCost;

1. Each actor is examined to get the least cost in findMinimumCost function, and by using sequential unsorted invitaCost array, each actor has to be tested to get and update the minimum cost, the running time is amount to O(n^2).

References

[1] Michael T. Goodrich,Roberto Tamassia, Data Structures and Algorithms in Java, 5th (2010), chapter 13.