

Contents

1 Find the shortest path with the greatest skill level between two users	1
1.1 Task	1
1.2 Algorithm	1
1.3 Example	2
1.3.1 The Json file	4

1 Find the shortest path with the greatest skill level between two users

1.1 Task

The task is motivated by wanting to find the shortest path from a coder to another coder in a social network graph. The path should be the shortest possible, but consist of the strongest coders possible.

Each coder has a skill level, and to make the task look like a minimisation/shortest path problem, the skill level is inverted to find a weighting for edges between two coders in the graph.

The shortest path is the one where the sum of the weights (inverted skill levels), on edges, is a minimum.

NOTE: The value 2 is arbitrarily chosen for links to coders with skill level=0.

1.2 Algorithm

Setup:

1. Set the start node's cost to 0
2. Set all other node's costs to some Maximum value (infinite)
3. Insert the start node onto an ordered (initially empty) list nList
4. Run the findShortest algorithm

The findShortest algorithm is:

```
while targetNode hasn't been visited:
    if nList is empty - we can't find a path
    pop the first element of nList into curNode
    for each node f in the friends list of curNode:
        if f has not been visited, set it's tentative cost if required:
```

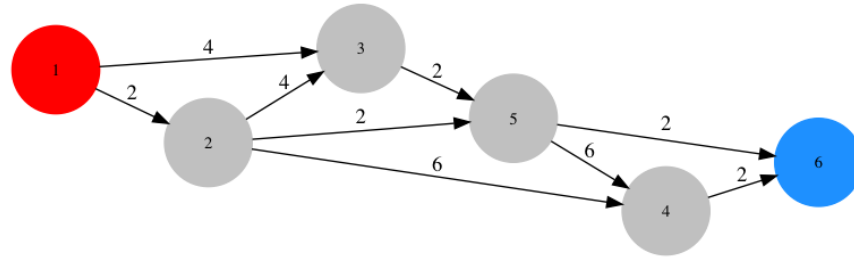
```

if f.skill != 0
    tentativeCost = curNode.cost + 1 / f.skill
else:
    tentativeCost = curNode.cost + 2 (or chosen value.)
if tentativeCost < f.tentativeCost:
    f.tentativeCost = curNode.cost + 2 (or chosen value.)
    insert f in order into nList

```

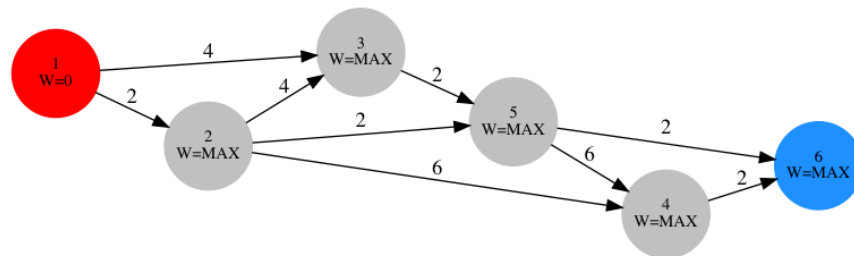
NOTE: An addition to the above algorithm, we record the path by setting a prev link from destination to source, between each f and curNode whenever we reduce f's tentativeCost.

1.3 Example



Given we want to find the shortest path with the above criteria between coder 1 and coder 6, we start at 1.

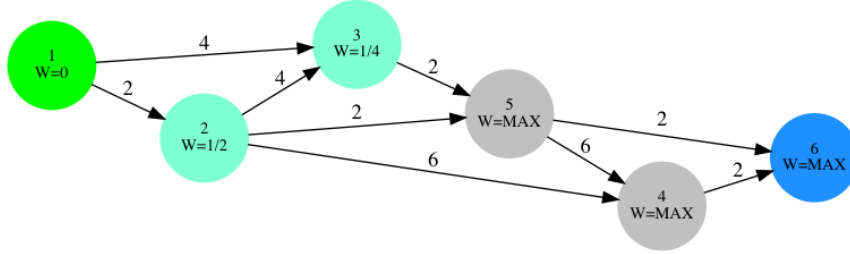
Initialise the graph with all costs = some max value, except 1, which has a cost of zero:



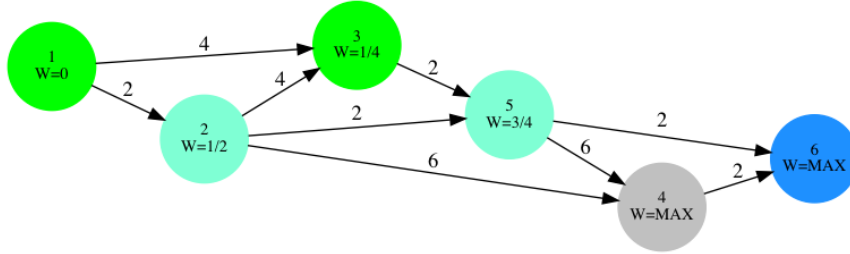
At this point, we push node 1 onto nList, mark it visited, and start with the findShortestPath Algorithm.

The algorithm pop's node1, and updates both of it's friends with tentative costs, $\frac{1}{4}$ for node 3, and $\frac{1}{2}$ for node 2. After the inner for loop completes, nList should contain node 3 and node 2 in that order.

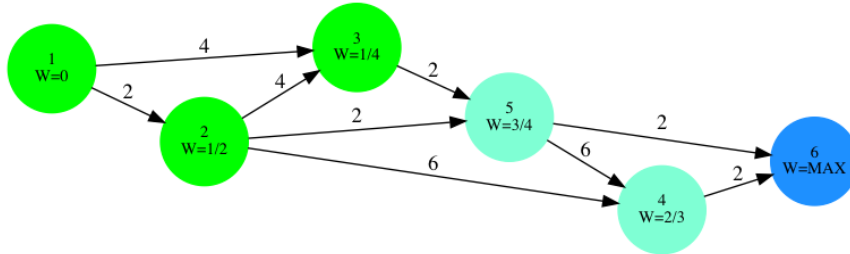
The graph is now like this:



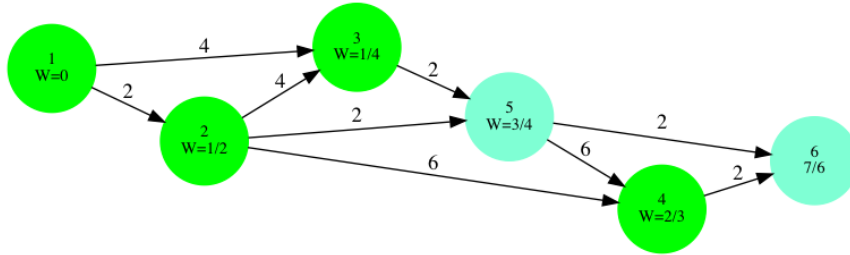
The first node in nList is now node 3 with weight $\frac{1}{4}$, so we process that node in the same way, and end up with the following graph:



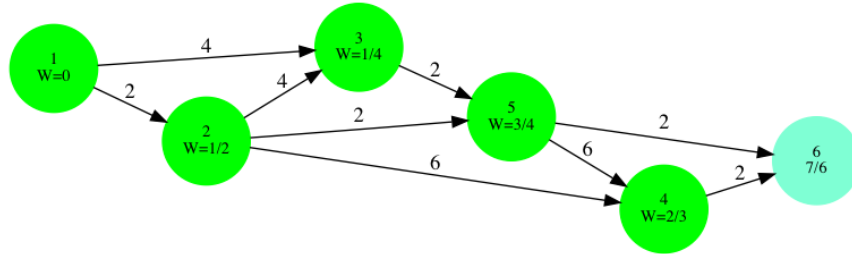
Once again, nList has the inorder list of nodes, ie. node 2 then node 5, so the next iteration processes Node 2, and we end up with the graph:



Since $\frac{2}{3} < \frac{3}{4}$, the first node in nList is now node 4, followed by node 5. We pop node4 from nList, and process it to produce:



At this point, nList contains node 5 followed by node 6, so node 5 is processed. NOTE: In this case, neither of the friends of node 5 are updated because the cost at node 5 + the weights to each of the nodes from node 5, are greater than their currently assigned costs, so the graph now looks unchanged except node 5 has changed colour to green to indicate it's been visited:



Finally, there is only one node left in nList, and we visit that node 6, which terminates the algorithm.

So our shortest path is:

1 -> 2 -> 4 -> 6

This path has a length $\frac{1}{2} + \frac{1}{6} + \frac{1}{2} = \frac{7}{6}$

1.3.1 The Json file

```
{"user" : 1 , "friends" : [2, 3], "skill" : 0}
{"user" : 2, "friends" : [3, 4, 5], "skill" : 2}
{"user" : 3, "friends" : [5], "skill" : 4}
{"user" : 4, "friends" : [6], "skill" : 6}
{"user" : 5, "friends" : [4, 6], "skill" : 2}
{"user" : 6, "friends" : [], "skill" : 2}
```