Problem 1:

Assumption: The distance in graph G is well-defined, which means there are no negative distance for all edges.

Proof by contradiction:

Suppose T’ becomes the new shortest path tree. By running dijkstra’s algorithm, at vertex v, the algorithm goes for v’ rather than v’’, to make a new shortest path tree. We know that:

Based on the shortest path tree T, we know

Which means there are positive numbers a<b, such that b2>a2. Which is a contradiction.

So the statement holds.

Problem 2:

a) This can be a variation of dijkstra’s algorithm to find shortest path tree.

Function: shortestPathToDest(graph, d):

For each edge e in graph:

reverse direction

End For

Create vertex set Q

For each vertex v in graph:

dist[v] ← INFINITY

prev[v]← UNDEFINED

Add v to Q

End For

Dist[d]←0

While Q is not empty:

u← vertex in Q with min dist[u]

Remove u from Q

For each neighbor v of u:

alt← dist[u] + weight(u,v)

If alt < dist[v]

dist[v]← alt

prev[v]← u

End if

End for

End While

For each vertex v other than d

return\_dist[v]← dist[v]

End For

Return return\_dist

The overhead is to reverse the graph (reverse again if necessary), which is O(E) operation. The dijkstra’s algorithm take O(|E| + |V|log|V|), so the whole algorithm complexity is O(|E| + |V|log|V|)

b) Apply the algorithm:

By reversing the edges, I got a new graph G’. By applying the graph on G’:

1. I have the shortest path from d to e is: d→b→e, the distance is 7.
2. The shortest path from d to c is d→b→a→c, the distance is 9.

Then reverse result, so the cost from e to d is 7, e→b→d. The cost from c to d is 9, c→a→b→d

Problem 3:

a)

Exactly k cluster:

Function kCluster(dist):

Sort distances in ascending order

i←0

For each vertex v in dist:

label[v]←i

i←i+1

End for

c← |dist|

For each edge e in sorted list:

If u, v have the same label:

Continue

End if

label vertex u,v to min(u,v)

Set all label with max(u,v) to min(u,v)

c←c-1

If c == k:

Break

End if

End for

Return label

Minimum distance:

Function minDistanceCluster(dist, theta):

Sort distances in ascending order

i←0

For each vertex v in dist:

label[v]←i

i←i+1

End for

c← |dist|

For each edge e in sorted list:

If u, v have the same label:

Continue

End if

label vertex u,v to min(u,v)

Set all label with max(u,v) to min(u,v)

c←c-1

If weight(e) > theta:

Break

End if

End for

Return label

b) This definition of distance can be problematic in the presence of outlier. If an outlier is classified to A but really close to B, the distance between A and B are really close but actual distance can be really large.

c) Bonus:

For title file, result files and source code, please refer to attachment. (I printed it below, I didn’t print the test file, its waste space)

I manually made a list of 20 items which cover 10 singers, 5 soccer players and 5 medicines. After getting results from web service, I parse the result and put them in a list of tuples. By applying the above algorithm, the output of program is:

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 10, 10, 10, 10, 10, 15, 15, 15, 15, 15]

The array means the labels w.r.t titles. As we can see the first 10 all clustered to label 0, which means it correctly find that those titles are in the same group. The same as all label 10 (soccer player) and label 15 (medicine)

titles = [

# singer

'Katy\_Perry',

'Taylor\_Swift',

'Ariana\_Grande',

'Mariah\_Carey',

'Justin\_Bieber',

'Lady\_Gaga',

'Michael\_Jackson',

'Britney\_Spears',

'Johnny\_Cash',

'Jennifer\_Lopez',

# soccer

'Lionel\_Messi',

'Cristiano\_Ronaldo',

'Neymar',

'Wayne\_Rooney',

'Diego\_Maradona',

# Medicine

'Penicillin',

'Insulin',

'Polio\_vaccine',

'Ether',

'Arsphenamine',

]

"""

with open('test.tsv', 'w') as f:

for t1 in titles:

for t2 in titles:

if t1 != t2:

f.write(t1 + '\t' + t2 + '\n')

print('Writing done.')"""

result\_list = []

with open('1498532617747-test.out.tsv') as f:

lines = f.readlines()

for l in lines:

split\_array = l.split('\t')

result\_list.append((split\_array[0], split\_array[1], float(split\_array[2].split('\n')[0])))

result\_list.sort(key=lambda tup: tup[2], reverse=True)

labels = list(range(20))

cluster = 20

for result in result\_list:

i1 = titles.index(result[0])

i2 = titles.index(result[1])

if labels[i1] != labels[i2]:

min\_label = min(labels[i1], labels[i2])

max\_label = max(labels[i1], labels[i2])

for n,l in enumerate(labels):

if l == max\_label:

labels[n] = min\_label

cluster = cluster - 1

if cluster == 3:

break

print(labels)