**Solution 1**

No, The algorithm does not work as it does not take the minimum distance among the neighbors which leads to incorrect solution.

For example consider the following graph,

1

1

C

A

1

D

10

B

10

We start from A, => dist [C] = 1, dist[B] =dist[D]=10

Then we suppose take B(because now we are not going with minimum neighbor distance)

Q : A **B** D C

We don’t update anything

Q : A B **D** C

Now we take D, Again we do nothing

Q : A B D **C**

Now we take C, we update D as dist[D]=2

The algorithm ends and we have dist[B] = 10 which is not true as the dist[B] should be clearly 3 (path A->C->D->B).

**Solution 2**

Pseudo-code:

**Dijkstra ( G=(V, E, w), s )**

1. Let H=∅

2. For every vertex v do

3. dist[v]=∞

4. dist[s]=0

5. Update (s)

6. For i=1 to n-1 do

7. u=extract vertex from H of smallest cost

8. Update(u)

9. Return dist[]

7

**Update (u)**

1. For every neighbor v of u

2. If dist[v]>dist[u]+w(u,v) then

3. dist[v]=dist[u]+w(u,v)

4. If v not in H then

5. Add v to H

Verbal Description:

To find single-source shortest path on positive-vertex-weighted undirected graphs from the given source we can quite simply implement the Dijkstra’s Algorithm. The above algorithm is Dijkstra’s Algorithm. This algorithm in a way solves the shortest path problem in a dynamic way.

The heart here is: the min distance between any two points.

The only difference here is that we have weights directly on the vertices which does not make any changes in Algorithm but a little change in the program (depending on the implementation).

Correctness:

Say, there is a shortest path defined between S and T (S=> source and T=> Target)

And for contradiction, there is another path

from S to X X being an intermediate path.

and X to T

which is shorter.

Then this algorithm would not have chosen the path S-T instead the path S-X-T.

Thus by contradiction we know that this algorithm gives the shortest path.

Running Time:

O(n2)

As the Dijkstra’s Algorithm runs in O(n2).

**Solution 3**

Pseudo-code:

**StartDijkstra(G=(V, E, w))**

1. for i=1 to n

2. cnt[] = Dijkstra (G, i )

3. Print (dist[i]/cnt[i])

**Dijkstra ( G=(V, E, w), s )**

1. Let H=∅

2. For every vertex v do

3. dist[v]=∞

4. cnt[v] = 1

5. dist[s]=0.

6. ret[][] = Update (s, 1, cnt)

7. cnt[ret[0][0]] = ret[0][1];

8. For i=1 to n-1 do

9. u=extract vertex from H of smallest cost

10. ret[][] = Update(u, 1, cnt)

11. cnt[ret[0][0]] = ret[0][1];

11. return cnt .

**Update (u, cnt1, cnt)**

1. For every neighbor v of u

2. If dist[v]>dist[u]+w(u,v) then

3. cnt1++

4. ret[0][0] = v;

5. ret[0][1] = cnt1;

6. If dist[v]>dist[u]+w(u,v) then

7. cnt[v] = cnt[u];

8. dist[v]=dist[u]+w(u,v)

9. If v not in H then

10. Add v to H

11. return ret;

Verbal Description & Proof of Correctness:

We run the Dijkstra algorithm for all the pair of vertices . As the weights are non negative, Dijkstra works perfectly for all pairs.

To get the count of all multiple paths:

If there exists multiple paths of minimum distances, for any vertices (u)🡪(v) they could be merging on the vertice (v) or if we pick a path of minimum distances from(u)🡪(v) they could be merging on any intermediate vertice(s) on that minimum path.

So for the condition

If dist[v]=dist[u]+w(u,v)

we increase the count of vertice (v) as there is a different path of same length to (v).

If dist[v]>dist[u]+w(u,v) then

dist[v]=dist[u]+w(u,v)

we assign all the paths of (u) to (v) coz if multiple paths exists to (u) they all will reach (v) through (u)

Running Time:

Running time will be same as , Dijkstra . that is O(n2) for a single source , here we compute for every vertex so the running time will be O(n3).