演算法 Homework#3

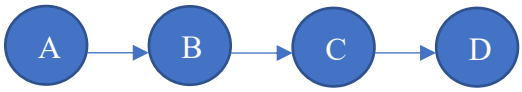
山下夏輝（Yamashita Natsuki）
R08922160

Reference: 陳建宏（d08944003），TA

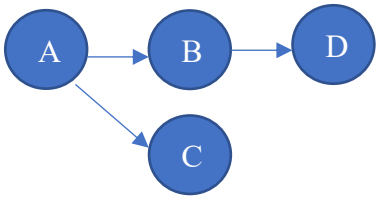
ProblemD

1.

Same order

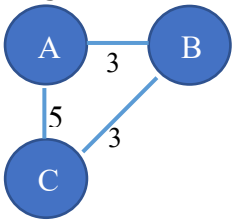


Different order

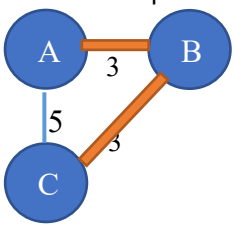


2.

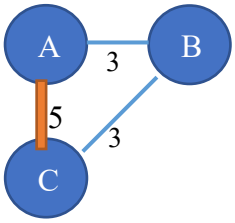
Original



Minimum Spanning Tree



Shortest Path Tree



Consider the graph showed above. The minimum spanning tree has 2 edges and is satisfied to reach every vertex. On the other hand, let us set the condition that the start point is on

A and the destination is on C. Then the shortest path tree has only 1 edge and has a shortest path from the start point to the destination.

So it is disproved that the minimum spanning tree is the same as the shortest path tree.

3-(a). //reference: the resume of the class

```
Function_M(G, u,  stack)
        u.color = GRAY
        for each v in G.Adj[u]
                if v.color == WHITE
                        v.pi = u
                        v.pi.m_fanc = u.m_func
                        Function_M(G, v,  stack)
        u.color = BLACK
        stack.push(u)


Minimum_time(G)
for each vertex u in G.V
        u.color = WHITE
        u.pi = NIL
for each vertex u in G.V
        if u.color == WHITE
                Function_M (G, source,  stack)
For each vertex u in G.V
        u.m_value = INF
source.m_value = 1
While Stack == empty
        u = stack.top()
        stack.pop()
        if m_value != INF
                for each v in G.Adj[u]
                        if v.m_value > u.m_value + uv.r_value + v.c_value
                                v.m_value = u.m_value + uv.r_value + v.c_value
For each vertex u in G.V
        Print “The value of the function M of” vertex “is” v.m_value
```

3-(b).

This algorithm is based on Topological Sort. So it runs on input DAG G with O(m+n). Calculating the m_values runs with O(m + n). So that this runs with O(m + n).

About the correctness of this algorithm, Topological Sort ensures the order of calculation for accumulating and saving the minimum m_value correctly. When this program accumulate the m value from u to v, it checks whether the m_value of the current vertex u

is smaller than of the other adjacent vertex v-pai. Then the m_value is updated only when the condition is satisfied. In addition to it, based on the feature the Topological Sort, the updating the value is not influenced by the u-pai(v-pai-pai). So the correctness of the calculation is guaranteed.

Problem E //reference: the resume of the class

1.

```
MAILING_FEES(G, w, s)
//initialize
for v in G.V
        v.d = INF-MAX
        v.π = NIL
//src distance is set as 0
s.d = 0
S = empty
Q = G.v          //Priority queue
while Q ≠ empty
        u = Q.pop()
        S = S∪{u}
        for v in G.adj[u]
                if v.d > u.d + w(u, v)
                        v.d = u.d + w(u, v)
                        v.π = u
                        Q.push(v)

for v in G.V
        print v, v.d*2
```

2.

Modification stage

a. Modify the graph from bi-direction to one-direction
b. Delete mutilation of 2 on v.d in print function
c. Add the same algorithm with conversed directions and record the results in v’ and v’.d
d. Print every v and (v.d +v’.d)

About the correctness of algorithm, because of the original algorithm, the value of the shortest path from source to the destinations is recorded in v.d and because of modification stage step c, the value of the shortest paths from the destinations to the source is recorded in v’.d respectively. In print function, the two value is added each other, which means the minimal value of the shortest path of on the way to the destination from the source and on the way back to source from the destination.

About the time complexity, this algorithm can run in O(ElogE) because it is based on Dijkstra’s algorithm with priority queue. Specifically speaking, the function of EXTRACT_MIN, the function of the while loop and the function of the calculating the total weights of the paths and updating them runs in O(logV), O(V) and O(E) each. Then the

time complexity is O(VlogV) + O(ElogV). According to the characteristic of the graph, E > V. Therefore O(ElogE) > O(ElogV). So the time complexity requirement is satisfied.
//reference : https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-using-priority_queue-stl/

3.

```
MAILING_FEES(G, w, s)
        //initialize
        for v in G.V
                v.d = INF-MAX
                v.π = NIL
        //src distance is set as 0
        s.d = 0
        //Bellman-Ford algorithm : sufficient times of relaxation are G
        for i = 1 to |G.V| - 1
                for (u, v) in G.E
                        if v.d > u.d + w(u, v)
                                v.d = u.d + w(u, v)
                                v.π = u
        //detect negative cycles
        for (u, v) in G.E
                if v.d > u.d + w(u, v)
                        return “I am rich!”

MAIN(G, w, s)
        MAILING_FEES(G, w, s)
        //converse directions
        New G’
        for (u, v) in G.E
                G’.E(u’,v’) = G.E(v, u)
        MAILING_FEES(G’, w, s)

for v in G.V
        print v, v.d + v’.d
```

About the Correctness of the Algorithm, according to Bellman-Ford algorithm, (V-1) times of relaxation are sufficient to finish relaxing all of the costs unless one or more negative cycles exist. So in the Vth iteration can detect whether one or more negative cycles exist or not. When it exists, “I am rich!” is returned.

About the minimum value, the way to calculate the minimum value from the source to each destinations and from each destinations to source is the same as the answer of subproblem 2.

About the time complexity requirement, this algorism runs O(mn) because it is based on Bellman-Ford algorithm. Specifically, the initialization, the relaxation, the detection negative cycles, the conversion of directions and the print of the results runs O(n), O(mn), O(n), O(n), O(n). So this algorithm is run in O(mn). Therefore the time complexity requirement is satisfied.

4.

About Time Complexity

The time complexity of Subproblem 2 is better than its of subproblem 3: O(ElogE) < O(EV). In the algorithm of 2, the edges of the adjacent of u is checked in the V-times-loop. On the other hand, in the algorithm of 3, every edges in Graph should be checked in the V-times-loop. This cause the difference between the time complexity of two algorithm.

About Space Complexity

The space complexity of Subproblem 3 is better than its of subproblem 2. In the algorithm of 2, the space of the 4 stack for store each information of the graph. On the other hand, in the algorithm of 3, just the space for 2 set of the graph-information is needed.

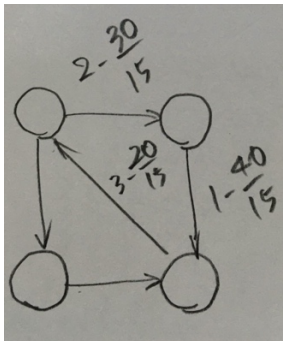
5.

In order to find a “trustful cycle” with Bellman-Ford algorithm, the cycle need to be a negative cycle. To speak in the other word, when there is “trustful cycle”, the cycle should be a negative cycle. Then, the calculation for reweighting and the image of the example is below.

K	< r1 + r2 + r3 / m1 + m2 + m3
m1 + m2 + m3	< r1 + r2 + r3 / k
0	< r1/k – m1 + r2/k - m2 + r3/k - m3
0	> m1 - r1/k + m2 – r2/k + m3 – r3/k

r: reliability

m: mailing fee



After reweighting like this, the negative cycle can be detected with Bellman-Ford Algorithm. If the negative cycle exists, it means that the ratio of total reliability to total mailing fees is larger than K, which means that the “trustful cycle” exists.

About the time complexity, the reweighting runs in O(n) and Bellman-Ford Algorithm runs in O(N*E). So this algorithm runs in in O(N*E). Therefore the time complexity requirement is satisfied.