Advanced Algorithm HW9

Seungho Jang

November 18, 2020

1 Q1

Tracing the Dijkstra Algorithm is shown as below.

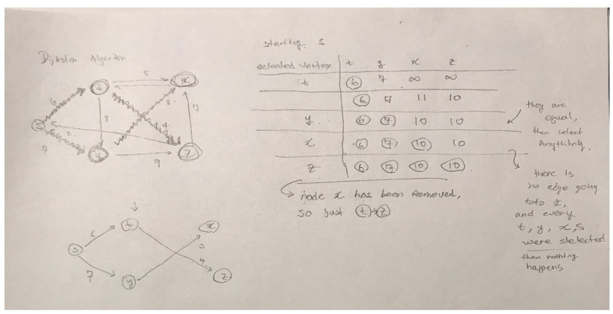

Figure 1: Tracing Dijkstra Algorithm

2 Q2

2.1 a

Tracing the Dijkstra Algorithm is shown as below,

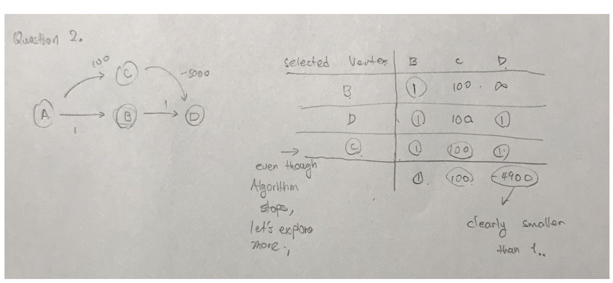

Figure 2: Dijkstra Algorithm : Negative Edges

The Dijkstra Algorithm can be done successfully with this graph, but it contradict the assumption that the output of the Dijkstra algorithm. The path cost C to D is -4900, but it's 1 after running the algorithm. Numerically, -4900 is smaller than 1, then it should be the solution for the Dijkstra algorithm. This basically contradiction.

2.2 b

In this case, if 5001 was added to all the weight, it will be valid as shown below,

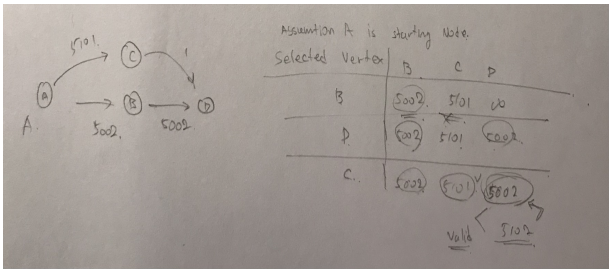

Figure 3: Adding Constant Value into Each Weight

but the issue is that how we can determine the constant value, 5001 before viewing every edge's weight. Then, it means that every edges should be traversed to find the negative edge, and operation will be involved in following way; find the smallest weight(cost), convert that number to positive, and add that number to each weight. This will lead to additional computational resource to store that the smallest edges and longer latency to convert every edges weight if the graph is way bigger than the problem and to compute the shortest path. Due to those reasons, this will be more bigger issue when it comes to optimizing the shortest path problem.

3 Q3

The graph on the problem can be done with the Dijkstra's algorithm because it has negative weight a to b, which has -4. However, it's possible to perform Bellman-Ford algorithm because the source(a) is disconnected from the negative-weight cycles such as e and f. Since the goal is to find the shortest path a through g, it is possible. After the relaxation done at node a, there is only a edge going towards to b. Also there is only a edge that is connected b to g. The bellman-ford algorithm can run on this domain problem. If the starting node is s then, the bellman-ford algorithm can be done because source is connected to e and f which is negative cycle.

4 Q4

Assuming that extract-min takes O(lgV), which means that it used binary heap implementation. So while loop itself will take $O(|v|)$ times with O(lgv) from extract-min. Therefore, the running time will be $O(|v| * lg|v|)$. Then relax operation will be done in maximum $|E|$, which means to exploring every edges. Then the total time of it is going to be $|E| * |V|$. Then the total time will be $O(|V|lg|V| + |E| * |V|)$ where V represents magnitude of number of vertex and E represent magnitude of number of edges.

5 Q5

The work is shown below,

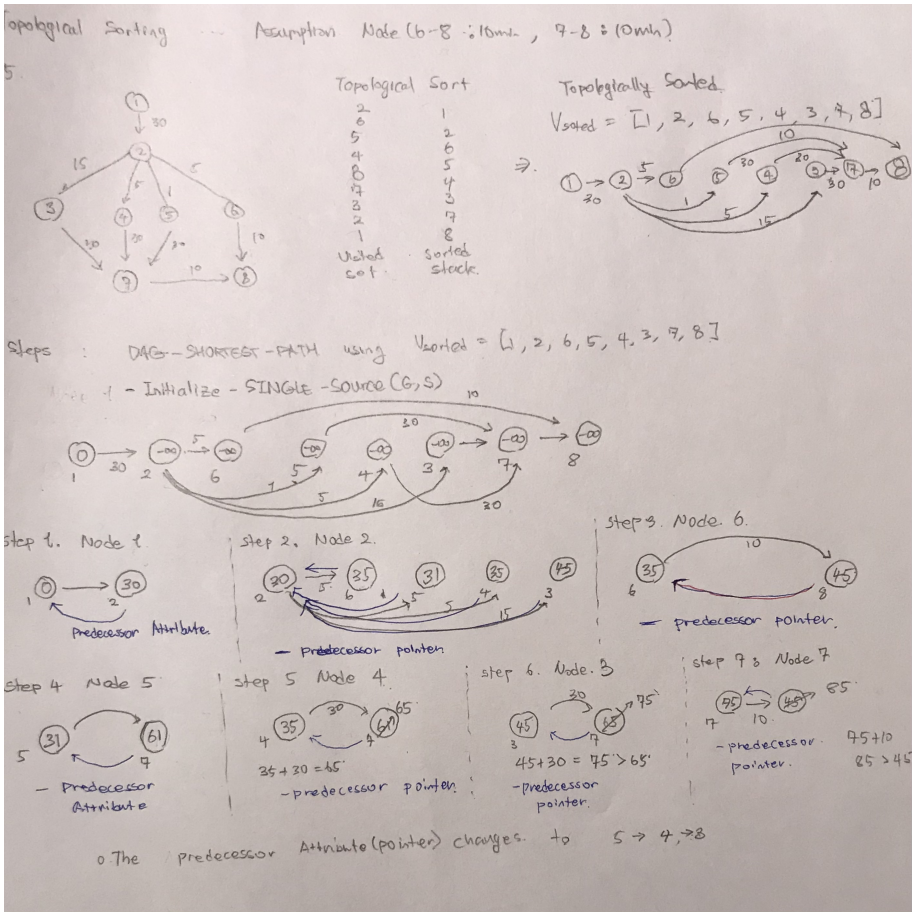

Figure 4: DAG - Longest Path Algorithm and Problem

6 Q6

The iteration happens until all the cost in vertex doesn't change, and one of the assumption is to use **S is my starting node** as shown in the problem.

One thing to note is that the maximum iteration will be 4 because it will iterate $|V| - 1$.

1. The number of iteration is equal to the moment that vertex of the cost starting not to change, which means relaxation stops at 4. Also, the order of the smallest weight is following $(t, z), (y, x), (t, x), (z, s), (t, x), (s, t), (s, y), (x, z), (t, y)$ $, (y, z)$

2. The number of iteration is equal to the moment that vertex of the cost starting not to change, which means relaxation stops at 3. Also, the oder of the largest weight is following $(y, z), (t, y), (x, z), (s, y), (s, t), (t, x), (z, s), (t, x), (y, x)$ $, (t, z)$

3. The number of iteration in the problem will run 4 times, which means that the vertex of the cost starting not to change, which means relaxation stops at 4.

So we may be able to conclude that the Non-increasing weight could converge faster than the non-decreasing weight if there is no negative cycle.

7 Q7

Assumption : The Silicon Valley map provided from HW has three missing nodes, and those names(Golden Gate Bridge, Half Moon Bay, and San Gregorio) were added from google map. The modified Silicon Valley map is shown below,

Then, the bellman-ford shortest path algorithm were used, and the code link can be found in here : Silicon Valley Map Problem

Then, to determine the copy of the code from here Bellman-Ford Algorithm - GeeksforGeeks: is correct, I used the Scipy library to determine whether it's correct.

The Scipy will return two outputs: shortest distance from point i to point j along the graph, and their predecessor.

The documentation of Scipy library for bellman-ford algorithm is from this link The Scipy Documentation.

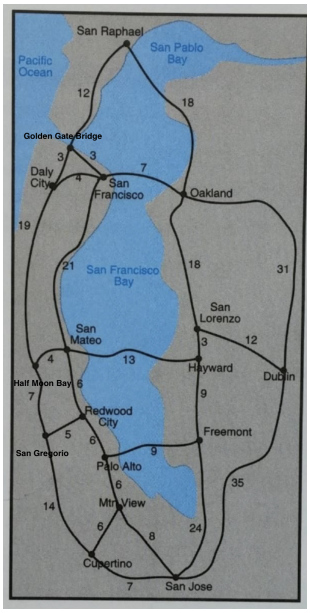

Figure 5: Silicon Valley Map