Sample Solution for Problem Set 12

Data Structures and Algorithms, Fall 2022

February 13, 2023

Contents

1 Problem 1

Give weight 0 to each edge in original graph. Split a vertex v into two vertexes v_0, v_1, and add a new edge (v_0, v_1) with weight v_w. Then, every path through this dual graph corresponds to a path through the original graph.

A job can start if all its previous jobs has finished.

(a) Set d_{s_0} to 0, and then use PERT algorithm in the textbook to calculate the "shortest" paths to each vertex.

Then d_{v_0} is the earliest start time of the job represented by v.

(b) Set d'_{t_1} to 0, and reverse every edge. Then use PERT algorithm in the textbook to calculate the "shortest" paths to each vertex.

Then $d_{t_1} - d'_{v_0}$ is the latest start time of the job represented by v, without affecting the project's duration.

(c) Use d to find the "shortest" paths from s to t. Then v is in some critical path if and only if $v_0(v_1)$ is in some "shortest" paths.

2 Problem 2

Consider the following graph $G = (V, E, w)$.

- $V = \{v_1, v_2, \ldots, v_n\}$.
- for $1 \leq i, j \leq n$ such that $i \neq j$, $v_i v_j \in E$, and $w(v_i, v_j) = -\log r_{ij}$.

We may run Bellman-ford algorithm on it for detecting the presence of anomaly and calculate the most advantageous sequence of currency exchanges for converting currency s into currency t. Time complexity for this algorithm is $O(n^3)$.

3 Problem 3

(a) $k = 1$

$$\begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ \infty & 2 & 0 & \infty & \infty & -7 \\ -5 & \infty & \infty & 0 & -6 & \infty \\ \infty & 6 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{pmatrix}$$

$k = 2$

$$\begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ 3 & 2 & 0 & 4 & 2 & -7 \\ -5 & \infty & \infty & 0 & -6 & \infty \\ 7 & 6 & \infty & 8 & 0 & \infty \\ 6 & 5 & 10 & 7 & 5 & 0 \end{pmatrix}$$

$k = 3$

$$\begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ 3 & 2 & 0 & 4 & 2 & -7 \\ -5 & \infty & \infty & 0 & -6 & \infty \\ 7 & 6 & \infty & 8 & 0 & \infty \\ 6 & 5 & 10 & 7 & 5 & 0 \end{pmatrix}$$

$k = 4$

$$\begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ -3 & 0 & \infty & 2 & -4 & \infty \\ -1 & 2 & 0 & 4 & -2 & -7 \\ -5 & \infty & \infty & 0 & -6 & \infty \\ 3 & 6 & \infty & 8 & 0 & \infty \\ 2 & 5 & 10 & 7 & 1 & 0 \end{pmatrix}$$

$k = 5$

$$\begin{pmatrix} 0 & 5 & \infty & 7 & -1 & \infty \\ -3 & 0 & \infty & 2 & -4 & \infty \\ -1 & 2 & 0 & 4 & -2 & -7 \\ -5 & 0 & \infty & 0 & -6 & \infty \\ 3 & 6 & \infty & 8 & 0 & \infty \\ 2 & 5 & 10 & 7 & 1 & 0 \end{pmatrix}$$

$k = 6$

$$\begin{pmatrix} 0 & 5 & \infty & 7 & -1 & \infty \\ -3 & 0 & \infty & 2 & -4 & \infty \\ -5 & -2 & 0 & 0 & -6 & -7 \\ -5 & 0 & \infty & 0 & -6 & \infty \\ 3 & 6 & \infty & 8 & 0 & \infty \\ 2 & 5 & 10 & 7 & 1 & 0 \end{pmatrix}$$

(b) If there was a negative-weight cycle, there would be a negative number occurring on the diagonal upon termination of the Floyd-Warshall algorithm.

4 Problem 4

(a) Here is the values of h and $\hat{w}$ computed by the algorithm to simplify the solution.

v	$h(v)$
1	-5
2	-2
3	0
4	0
5	-6
6	-7

u	v	$\hat{w}(u,v)$	u	v	$\hat{w}(u,v)$
1	2	NIL	4	1	0
1	3	NIL	4	2	NIL
1	4	NIL	4	3	NIL
1	5	0	4	5	9
1	6	NIL	4	6	NIL
2	1	4	5	1	NIL
2	3	NIL	5	2	2
2	4	0	5	3	NIL
2	5	NIL	5	4	NIL
2	6	NIL	5	6	NIL
3	1	NIL	6	1	NIL
3	2	4	6	2	0
3	4	NIL	6	3	3
3	5	NIL	6	4	NIL
3	6	0	6	5	NIL

So, the d_{ij} values that we get are

$$\begin{pmatrix} 0 & 5 & \infty & 7 & -1 & \infty \\ -3 & 0 & \infty & 2 & -4 & \infty \\ -5 & -2 & 0 & 0 & -6 & -7 \\ -5 & 0 & \infty & 0 & -6 & \infty \\ 3 & 6 & \infty & 8 & 0 & \infty \\ 2 & 5 & 10 & 7 & 1 & 0 \end{pmatrix}$$

.

(b) By lemma 25.1, we have that the total weight of any cycles is unchanged as a result of the reweighting procedure. This can be seen in a way similar to how the last claim of lemma 25.1 was proven. Namely, we consider the cycle c as a path that has the same starting and ending vertices, so, by the first half od lemma 25.1, we have that

$$\hat{w}(c) = w(c) + h(v_0) - h(v_k) = w(c) = 0$$

This means that in the reweighted graph, we still have that the same cycle as before had a total weight of zero. Since there are no longer any negative weight edges after we reweight, this is precicely the second property of the reweighting procedure shown in the section. Since we have that the sum of all the edge weights in c is still equal to zero, but each of them individually has a nonnegative weight, it must be the case that each of them individually is equal to 0.

5 Problem 5

5.1 (a)

Update(u,v): For u and its ancestors, they can reach v and its descendants.

Algorithm 1: UPDATE(u, v)

```
1 for i ∈ V do
2     if T[i][u] or i == u then
3         for j ∈ V do
4             if T[v][j] or j == v then
5                 T[i][j] = true;
6             end
7         end
8     end
9 end
```

5.2 (b)

Assume $G =< V, E >$, $V = \{v_1, v_2, \ldots, v_n\}$, $E = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n)\}$. Now add $e = (v_n, v_1)$. We need update $\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$ terms. Therefore, $\Omega(|V|^2)$ time is required for any algorithm.

5.3 (c)

Update2(u,v): Modify the algorithm in (a) by adding a condition.

Correctness:
If $T[i][v]$ is true, no transitive closure need to update.
If $T[i][v]$ is false, the same as the algorithm in (a).

Time complexity:
Use aggregate analysis. If $T[i][v]$ is **false**, the inner loop can make it **true**. Since T has $\Theta(n^2)$ elements, the inner loop (line 3-5) executes $O(n^2)$ times and costs $O(n^3)$. Since there are $x \in O(n^2)$ insertions, the outer loop (line 1-2) executes x times and cost $\Theta(x * n) = O(n^3)$. Therefore, this algorithm is $O(n^3)$.

Algorithm 2: UPDATE2(u, v)

```
1 for i ∈ V do
2     if (T[i][u] or i == u) and ¬T[i][v] then
3         for j ∈ V do
4             if T[v][j] or j == v then
5                 T[i][j] = true;
6             end
7         end
8     end
9 end
```

6 Problem 6

To index the subproblem, we will give the first k items from the item list and the maximum weight of current knapsack. There can be at most $O(nW)$ of these subproblem. For each item, we can decide to put it in or not. If we put the ist item in, we need to prepare w_i weight for it and earn v_i value. The value is v_i plus the solution to the subproblem with the first $i-1$ items, and $W - w_i$ capacity. Otherwise, the value is as same as the solution to the subproblem with the first $i-1$ items, and W capacity. We take the maximum value from the two choices for each item.

Algorithm

```
1 for i in V:
2     for j in W:
3         if j >= w[i]:
4             dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - w[i]] + v[i])
5         else:
6             dp[i][j] = dp[i - 1][j]
```

The time complexity is $O(nW)$