Sample Solution for Problem Set 10

Data Structures and Algorithms, Fall 2020

January 4, 2021

Contents

1 Problem 1

Algorithm: Let $P'[1...n]$ be n pairs $(P'[i].key, P'[i].ID)$ where $P'[i].key = P[i]$ and $P'[i].ID = i$. Similary, let $S'[1..n]$ be n pairs $(S'[i].key, S'[i].ID)$ where $S'[i].key = S[i]$ and $S'[i].ID = i$.

Sort the array P' by increasing order on $P'[i].key$, and sort the array S' by increasing order on $S'[i].key$. Let $\pi(P'[i].ID) = S'[i].ID$.

In other words, we sort P and S by increasing order and assign the shoes $S[i]$ to $P[i]$.

Complexity: Sorting procedure cost $O(n \log n)$ if we use merge sort. Other operation cost $O(n)$. The total time complexity is $O(n \log n)$.

Correctness: Suppose P and S are the array of sizes of feet and shoes in increasing order. Suppose there is an optimal solution π_{opt} such that there exists $1 \leq i < j \leq n$ and π_{opt} assign shoes $S[j']$ to $P[i]$ and $S[i']$ to $P[j]$ where $i' < j'$. We consider another optimal solution π'_{opt} that assign shoes $S[i']$ to $P[i]$ and $S[j']$ to $P[j]$, while other assignment is the same as π'_{opt}. Let $cost[\pi_{opt}]$ be the average difference between the size of a person's feet and the size of his/her assigned pair of shoes. Then $cost[\pi_{opt}] - cost[\pi'_{opt}]$ is

$$\Delta \triangleq \left|P[i] - S[j']\right| + \left|P[j] - S[i']\right| - \left|P[i] - S[i']\right| - \left|P[j] - S[j']\right|$$

With out loss of generality, we assume $P[i] \geq S[i']$ (Otherwise we can swap P and S).

If $P[i] \geq S[j']$, then $\Delta = 0$.

If $P[i] \leq S[j'] \leq P[j]$, then $\Delta = 2\left|S[j'] - P[i]\right| \geq 0$.

If $S[j'] \geq P[j]$, then $\Delta = 2\left|P[i] - P[j]\right| \geq 0$.

In all cases, π'_{opt} is still an optimal solution. Since any permutation can get to the increasing permutation by finite steps of swapping two inverse element, the increasing permutation in our algorithm is an optimal solution.

2 Bonus

Algorithm: Initially let $U' = U$, $S' = S$ and $cost = 0$. Do the following loop:

- While $U' \neq \emptyset$, do
  - For $s \in S'$, define the weight of s as $w(s) = \frac{|s \cap U'|}{c(s)}$. Let $s_m \in S'$ be the element in S' with the biggest weight.
  - $cost \leftarrow cost + c(s_m)$, $S' \leftarrow S' \backslash \{s_m\}$, $U' \leftarrow U' \backslash s$.
- Return $cost$.

Complexity: Each loop cost $O(n)$ to find the element with the biggest weight. There are at most n loop (We assume each element can be covered by at least one set). The complexity is $O(n^2)$.

Correctness: Suppose in the i-th loop, we choose s_i as the element with the biggest weight (i.e., denote s_m as s_i in the second line of the algorithm for the i-th loop). Denote U' before the i-th loop as U_i, denote S' before the i-th loop as S_i. Obviously we have $U_1 = U$.

Suppose $S_{OPT} \subseteq S$ is one of the optimal solution and $OPT = \sum_{s \in S_{OPT}} c(s)$. We claim that $w(s_i) \geq \frac{|U_i|}{OPT}$. Otherwise, since $w(s_i)$ is the biggest among all the weights of elements in U_i, we have $c(s) > |s \cap U_i| \cdot \frac{OPT}{|U_i|}$ for all $s \in U_i$. Since $\cup_{s \in S_{OPT}} s = U$, and for all $s \in S \backslash S_i$ we have $s \in U \backslash U'$, we get $\cap_{s \in S_{OPT} \cap S_i} = U_i$, which means $\sum_{s \in S_{OPT} \cap S_i} \geq |U_i|$. Thus, $\sum_{s \in S_{OPT} \cap S_i} c(s) > OPT$, which is impossible.

Now $c(s_i) \leq |s_i \cap U_i| \cdot \frac{OPT}{|U_i|}$. Write $k_i = |U_i|$. Obviously $|s_i \cap U_i| = k_i - k_{i+1}$ and we get

$$c(s_i) \leq \frac{k_i - k_{i+1}}{k_i} \cdot OPT \leq \sum_{k_{i+1} < j \leq k_i} \frac{1}{j} \cdot OPT$$

Thus the total cost of our approximation algorithm is

$$\sum_i c(s_i) \leq \sum_i \sum_{k_{i+1} < j \leq k_i} \frac{1}{j} \cdot OPT$$

Since we have $k_1 = n$ and $k_1 > k_2 > ....$, while finally get 1, the value is

$$\sum_{1 \leq j \leq n} \frac{1}{j} \cdot OPT = O\left(\ln n \cdot OPT\right)$$

iteration	1	2	3	4	5	6	7	8	final
A	0								0
B	1								1
C		3							3
D			4						4
E	4								4
F	8	7				6			6
G		7	5						5
H			6						6

3 Problem 2

3.1 (a)

Use the graph in (c) as example:
Minimum spanning tree: $\{$(A,B),(B,C),(C,D),(A,E),(D,G),(G,F),(G,H)$\}$
Shortest path tree: $\{$(A,B),(B,C),(C,D),(A,E),(C,G),(G,F),(G,H)$\}$
They are not identical.

3.2 (b)

Use $flag$ to record whether the distance of nodes changed during the m-th iteration. If not, break.

3.3 (c)

The final shortest-path tree: $\{$(A,B),(B,C),(C,D),(A,E),(C,G),(G,F),(G,H)$\}$.

4 Problem 3

4.1 (a)

Use DFS, only consider the edge that $l_e \leq L$.

4.2 (b)

Use Kruscal, judge whether $Find(s) == Find(t)$ after each $Union(u, , v)$. If so, let $L = l_e(u, v)$ and break. The fuel tank has a capacity of at least L.

Proof: Call the edge that merges s and t into the same set (u_0, v_0). If $L < l_e(u_0, v, 0)$, we can't have enough oil to go through (u_0, v_0). In that case, s and t are not connected. If $L \geq l_e(u_0, v_0)$, we can go from s to t along the minimum spanning tree.

5 Problem 4

Initially, we will make each vertex have a D value of 0, which corresponds to taking a path of length zero starting at that vertex. Then, we relax along each edge exactly $V - 1$ times. Then, we do one final round of relaxation, which if any thing changes, indicated the existence of a negative weight cycle. The code for this algorithm is identical to that for Bellman ford, instead of initializing the values to be infinity except at the source which is zero, we initialize every d value to be 0.

Another solution. Use $\min(w_{u,v}, d_u + w_{u,v})$ to relax d_v. Then d_v will be $\min_{u \in V \wedge u \neq v}\{\text{dist}(u, v)\}$.

6 Problem 5

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.

7 Problem 6

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 $O(n^3)$.