Introduction to Algorithms	**Solution Set 4**
CS 4820, Spring 2018	

(1) *You've been hired to consult for a local ride sharing service. The map of Ithaca is divided into a finite set of locations, X, and you have a dataset which reports the amount of time required to drive between any two locations, and the fare charged for driving a customer between those two locations. (In other words, you have matrices D and F such that $D(x, y)$ denotes the amount of time required to drive from x to y, and $F(x, y)$ denotes the fare to be charged. For the purpose of this homework problem, we will make the unrealistic assumption that $D(x, y)$ is known* precisely, *with no uncertainty.)*

Your job is to serve requests from people who ask to be picked up at a specific location x_i at a specific time t_i, and dropped off at another location y_i. A set of requests is feasible *if it is possible for a single vehicle to serve all of them. In other words, the vehicle can't serve two requests simultaneously, and it also needs to have enough time to drive from the drop-off point of one request to the pick-up point of the next one in order to arrive there at (or before) the requested pick-up time.*

Given an input consisting of the set X, the distance matrix $D(x, y)$, and a finite set of requests (x_i, y_i, t_i), design an efficient algorithm to compute the maximum total fare *that can be charged for serving a feasible subset of the requests.*

Solution: For each request, define the finishing time to be $f_i = t_i + D(x_i, y_i)$. This is the time at which the cab will arrive at the passenger's destination after satisfying request i. Assume the requests are numbered in order of increasing f_i. (In other words, sort them so they are in this order.) For $i = 1, 2, \dots, n$, let $\mathsf{OPT}(i)$ denote the maximum fare that can be collected from a feasible subset of requests $1, \dots, i$ that finishes with request i. For convenience define $\mathsf{OPT}(0) = 0$.

Let $R(i)$ denote the union of $\{0\}$ with the set of all requests $j < i$ such that $f_j + D(y_j, x_i) \le t_i$. (These are requests such that the cab has time to drop off passenger j and still arrive at the pick-up location for passenger i on time.)

Lemma 1. *For all $i > 0$,*

$$\mathsf{OPT}(i) = F(x_i, y_i) + \max\{\mathsf{OPT}(j) \mid j \in R(i)\}. \tag{1}$$

Proof. Let $S(i)$ denote the set of all feasible request sequences finishing with request i. For convenience, define $S(0)$ to be a singleton set consisting of the empty sequence. There is a one-to-one correspondence

$$H_i : S(i) \to \left(\bigcup_{j \in R(i)} S(j) \right)$$

defined by removing request i from the end of each sequence in $S(i)$. Let $TF(\sigma)$ denote the total fare of a sequence of requests. For any feasible request sequence σ ending with i, we have the equation

$$TF(\sigma) = F(x_i, y_i) + TF(H_i(\sigma))$$

which implies

$$\begin{aligned}
\mathsf{OPT}(i) &= \max\{TF(\sigma) | \sigma \in S(i)\} \\
&= F(x_i, y_i) + \max\{TF(H_i(\sigma)) \mid \sigma \in S(i)\} \\
&= F(x_i, y_i) + \max\left\{TF(\sigma') \mid \sigma' \in \bigcup_{j \in R(i)} S(j)\right\} \\
&= F(x_i, y_i) + \max\{\mathsf{OPT}(j) \mid j \in R(i)\}
\end{aligned}$$

as claimed. □

The lemma prompts us to design the following algorithm.

```
1:  for i = 1, ..., n do
2:      Compute f_i = t_i + D(x_i, y_i).
3:  end for
4:  Sort requests by finish time. Assume henceforth that f_1 ≤ f_2 ≤ ··· ≤ f_n.
5:  for i = 1, ..., n do
6:      Compute R(i) = {j | f_j + D(y_j, x_i) ≤ t_i}.
7:  end for
8:  Let M[0] = 0.
9:  for i = 1, ..., n do
10:     M[i] = F(x_i, y_i) + max{M[j] | j ∈ R(i)}
11: end for
12: Return max{M[i] | i = 1, ..., n}.
```

The algorithm's running time is $O(n^2)$. The sorting step takes $O(n \log n)$. There are three (non-nested) loops, each running for n iterations. The first loop takes $O(1)$ time per iteration, the second and third loops take $O(n)$ time per iteration, so they predominate the running time.

The proof of correctness is an easy induction. The induction hypothesis is that for all $i = 0, \ldots, n$, the value $M[i]$ computed by the algorithm equals the value $\mathsf{OPT}(i)$ defined earlier in this solution. The base case $i = 0$ is trivial, since both $M[0]$ and $\mathsf{OPT}(0)$ are defined to be zero. The induction step is an application of Lemma 1: assuming $M[j] = \mathsf{OPT}(j)$ for all $j < i$, the right-hand side of the formula defining $M[i]$ in line 10 of the algorithm equals the right-hand side of equation (1), and hence the left-hand sides are equal as well: $M[i] = \mathsf{OPT}(i)$.

Alternate Solution: Create a graph G whose paths model feasible request sequences. The graph is defined to have the following nodes and edges.

- Source node s, sink node t.
- For each request i, a pair of nodes a_i, b_i representing starting and finishing the ride.
- For each request i, a directed edge (a_i, b_i) with length $-F(x_i, y_i)$.
- For each request i, directed edges (s, a_i) and (b_i, t) with length 0.
- For each pair of requests i, j such that $D(x_i, y_i) + D(y_i, x_j) \leq t_j - t_i$, an edge (b_i, a_j) with length 0.

Lemma 2. *The graph G is a directed acyclic graph.*

Proof. Let $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_n\}$. Assign a "timestamp" $t(u)$ to every node $u \in A \cup B$ by setting $t(a_i) = t_i$ and $t(b_i) = t_i + D(x_i, y_i)$. Sort the vertices by putting s first in the ordering, followed by the elements of $A \cup B$ in order of non-decreasing timestamp, breaking ties by putting elements of B before elements of A and otherwise resolving ties arbitrarily. Finally put t last in the ordering. This is a topological sort ordering of the vertices of G, because every edge defined above goes from an earlier vertex to a later vertex in the specified ordering. Hence G is a DAG. □

For a feasible request set $S = \{i(1), i(2), \ldots, i(k)\}$, numbered by increasing start time, define $Q(S)$ to be the path

$$Q(S) := \left(s, a_{i(1)}, b_{i(1)}, a_{i(2)}, b_{i(2)}, \ldots, a_{i(k)}, b_{i(k)}, t\right). \tag{2}$$

Conversely, for a path P in G, define the request set

$$R(P) := \{i \mid (a_i, b_i) \text{ is an edge of } P\}. \tag{3}$$

Lemma 3. *The functions P and Q define mutually inverse one-to-one correspondences between feasible request sequences and paths from s to t in G. Under this correspondence, a request sequence with total fare y corresponds to a path with length $-y$.*

Proof. The path $Q(S)$ is indeed a path in G, because the feasibility of request set S — combined with our assumption on the numbering of elements of S — ensures that each edge $(b_{i(j-1)}, a_{i(j)})$ is an edge of G, and all other pairs of consecutive nodes in $Q(S)$ are certainly joined by an edge. The equation $R(Q(S)) = S$ follows immediately from equations (**??**)-(2). If y is the total fare of the requests in S, then the only edges in $Q(S)$ are those of the form (a_i, b_i) for $i \in S$, and the length of each such edge is obtained by multiplying the fare of the corresponding element of S by -1. Hence, the total length of $Q(S)$ is $-y$.

Finally, to derive $Q(R(P)) = P$, we reason inductively about the structure of paths from s to t. The induction hypothesis is that every such path P is of the form $Q(S)$ for some feasible request set S. The proof is by induction on the number of elements of $P \cap A$. Every edge leaving s goes to A, so $|P \cap A| \geq 1$. The base case is $|P \cap A| = 1$, whereas the induction step is $|P \cap A| > 1$. In both cases, the path begins with an edge (s, a_i), which must be followed by (a_i, b_i) since it is the only edge leaving a_i. In the base case, the edge (a_i, b_i) must be followed by (b_i, t) since the only other edges leaving b_i go to A, and the base case assumes that a_i is the only element of $P \cap A$. Thus, in the base case we have confirmed that $P = Q(\{i\})$, as claimed. For the induction step, the edge of P following (a_i, b_i) must be of the form (b_i, a_j), since the only other alternative is (b_i, t) which would imply that $P \cap A = \{a_i\}$, contradicting the induction step's assumption that $|P \cap A| > 1$. Continuing with the induction step, observe that (s, a_j) is an edge of G so we can define path P' from s to t by deleting the initial segment (s, a_i, b_i, a_j) from P and replacing it with (s, a_j). As $|P' \cap A| = |P \cap A| - 1$, we can use the induction hypothesis to assert that $P' = Q(S')$ for some feasible request sequence S' beginning with request j. Since (b_i, a_j) is an edge of G, we know that $D(x_i, y_i) + D(y_i, x_j) \leq t_j - t_i$, implying that there is enough time for the vehicle to serve request and still arrive in time to serve request j. Hence $S = \{i\} \cup S'$ is a feasible set of requests, and $P = Q(S)$, which completes the induction step. Finally, the equation

$$Q(R(P)) = Q(R(Q(S))) = Q(S) = P$$

completes the verification that Q is the inverse of R. □

The design and analysis of the algorithm follow easily from the preceding lemmas. Given an input to the taxi scheduling problem, construct the graph G described above. This takes $O(n^2)$ time, with the most time-consuming step being the construction of the set of edges of the form (b_i, a_j), which requires iterating over all pairs of requests i, j. (Or at least, all such pairs for which $t_i < t_j$.) After having constructed G, run the Bellman-Ford algorithm to find a minimum-length path P from s to t, let $-y$ denote the length of P, and output the number y.

To verify that the algorithm is correct, note first that G is a DAG (Lemma 2), and in particular it has no negative-length cycles because it has no cycles at all. Therefore, the Bellman-Ford algorithm correctly computes a minimum-length path from s to t. Furthermore, if the length of this path is $-y$, then Lemma 3 ensures that there is a feasible request set $R(P)$ whose total fare is y. Finally, this is the maximum total fare that can be charged for serving a feasible subset of the requests, because if S is any other feasible subset of the requests with total fare z, then $Q(S)$ is a path in G with length $-z$, and the fact that P has the minimum length implies that $-y \leq -z$ which in turn implies $y \geq z$.

Finally, the running time of the algorithm consists of the time required to build the graph G, which was already determined to be $O(n^2)$, plus the time required to run the Bellman-Ford algorithm in G. Since G is a DAG with $2n + 2$ vertices and $O(n^2)$ edges, the running time of Bellman-Ford is $O(n^2 + 2n + 2) = O(n^2)$. Hence, the overall running time of the algorithm is $O(n^2)$.