CS 4820, Spring 2018 **Homework 4, Problem 1**

Name: Rongguang Wang
NetID: rw564
Collaborators: Siyuao Liu (sl2928); Yihao Chen (yc2288)

(1) *(10 points)*
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:

The problem is aiming to develop an algorithm which can compute the maximum total fare that can be charged for serving a feasible subset of requests (x_i, y_i, t_i). Given that the there is a set of different locations X, a matrix $D(x_i, y_i)$ storing the amount of time required to drive from x to y and a matrix $F(x_i, y_i)$ storing the fare to be charged from x to y. To charge the maximum total fare, the algorithm should find the longest path in which all requests are non-conflicting in time. Thus, Bellman-Ford algorithm could be a good choice to solve the longest path problem. This problem can be reduced to the shortest path problem which is valid to input to the Bellman-Ford algorithm.

The problem is a variation of the weighted interval scheduling problem. The requests of the problem can be seen as intervals; The weight of the intervals are the fare of each request; The starting time of the intervals is the requests time; The finishing time of the intervals is the sum of the request time and the travel time $D(x_i, y_i)$. Therefore, a valid input for the Bellman-Ford algorithm can be generated by creating a directional acyclic graph (DAG). The DAG can be created based on the weighted intervals which are the requests of the problem. The starting time and finishing time of each interval can be calculated by referencing the matrix $D(x_i, y_i)$. Then, the starting time and finishing time of each interval should be sorted in increasing order. Next, two vertices the start and sink points should be created for reference. Currently, there are only intervals, which are edges in the graph, the start and end points of each interval, which are vertices in the graph, and the start and sink vertices in the graph G. More edges should be add to connect the concrete edges and vertices.

Basically, there are four kinds of edges to be add to the graph G: linking start vertex s and the sink vertex t, linking the start vertex s and the starting points s_i of the intervals, linking the finishing points f_i of the intervals and the sink vertex t, and linking the starting points s_i and finishing points f_i of different intervals. The the first three kinds of edges can be created directly. But a checking should be performed before creating the edges between starting points s_i and finishing points f_i of different interval. We should make sure that the car can arrive at the starting point of the next requests after they finished the current request. So, the requirement that the starting time of the next request should be at least the sum of the finishing time of the last request and the time needed to travel from the end vertex of the last request to the start vertex of the next request, which is the time cost between two requests, should be met before creating the edges. The time cost between two requests can be obtained by reference to the matrix $D(x_i, y_i)$. Additionally, weights should be added to the edges. Only intervals (edges) will be assigned weight, the weights for all other edges are zero. To find the longest path through the shortest path algorithm, the weights for the intervals are negative value of the fare of them by referencing to the matrix $F(x_i, y_i)$. Finally, the DAG was created which can be used as the input of Bellman-Ford algorithm.

The output of Bellman-Ford algorithm is $P[i]$ and $T[i]$. $P[i]$ stores a list of vertices which representing the shortest path and $P[i]$ stores the shortest path length from the start vertex to the end vertex. We need only $T[n]$ which is the longest path length. It can be transformed to the maximum total fare by negate itself.

Algorithm:

1) Pre-process the requests:
a. Add the duration of each request to the start time s_i by checking the distance matrix $D(x, y)$
b. Sort the starting time s and the finishing time f of each interval

2) Create DAG graph for input
a. Add the opposite (negative) value of the fare from the fare set $F(x, y)$ for intervals (s_i, f_i)
b. Create source vertex s and sink vertex t and create edge (s, t) with weight 0
c. Create vertices for the starting point and the finishing point of each interval by checking requests set (x_i, y_i, t_i)
d. Create edges (s, s_i) from the start vertex s to the starting point s_i of each interval with weight 0
e. Create edges (f_i, t) from the sink vertex t to the starting point s_i of each interval with weight 0
f. Starting from the first element of the sorted starting time and finishing time array from the pre-processing step.
Create edges (f_i, s_j) from the finishing point f_i to the starting point s_j of each interval if the time cost $(s_j - f_i)$ between the finishing point f_i and the starting point s_j is positive and less or equal than the the starting time interval i with weight 0

3) Apply the graph to Bellman-Ford Algorithm

Output $T[k]$, which stores the shortest path length from the start vertex to vertex k

4) Transfer the result from B-F to total fare The opposite (negative) value of $T[t]$ (t is sink vertex) is the total fare

Algorithm's Correctness:

Lemma 1. *The result of the reduction is a valid input for Bellman-Ford algorithm.*
Proof. The input to the Bellman-Ford algorithm is a directional acyclic graph (DAG) with weight for each path. In this problem, a graph was created by converting the fare of the requests to the weight of the intervals. The intervals can be seen as directed paths pointing from the start vertex to the end vertex. Although the weights of each interval are negative numbers, they are all negative integers which can be used to compare.

Lemma 2: *Any start-to-sink path are feasible set of intervals.*
Proof. When creating edges between the finishing vertex and the starting vertex of different intervals, each interval will match with all intervals whose starting time is larger than the finishing time of itself. Thus, the graph contains all feasible paths that the car can run according to the question, which makes sure that each start-to-sink path are feasible set of intervals.

Lemma 3: *Each feasible set of interval is a start-to-sink path.*
Proof. Since the starting point of each interval was connected to the start vertex s through zero weight edges and the finishing point of each interval was connected to the sink vertex t through zero weight edges, all intervals can be reached either from the start point or the sink point. So, any feasible combinations of the intervals can be reached either from the start vertex or the sink vertex. Therefore, each feasible set of interval is a start-to-sink path.

Lemma 4: *The output of Bellman-Ford algorithm can be transformed into a valid output of the problem.*
Proof. The output of the algorithm is the shortest length of the given graph from the start to the sink and the valid output of the problem is the maximum total fare that can be charged from a feasible subset of the requests. The valid output of the problem is the negative value of the output of the algorithm. Both outputs are integers but with opposite sign, which has no cost.

Running Time Analysis:

The time needed to run the algorithm can be divided into three parts: pre-processing, DAG creating and Bellman-Ford algorithm. Given that there are i requests in the problem. Assuming there are m vertices and n edges in the created DAG. Since the requests in the problem are just part of the edges of the DAG, both m and n is larger than i. In the pre-processing step, the time complexity is $O(2i \log 2i)$. As for the DAG creation, the simplest checking procedure for create edges (f_i, s_j) from the finishing point f_i to the starting point s_j of each interval can be implemented as a nested for loop which spend $O(i^2)$ time. Finally, the time needed for the Bellman-Ford algorithm is $O(m + n)$. Therefore, the total time spend in the algorithm is $max\{O(i^2), O(m + n)\}$.