CSCI-UA.0310-004/005 Basic Algorithms May 11, 2015

Solutions to Problem 1 of Homework 12 (20 points)

Name: Jason Yao *Due: Monday, May 11*

You are given a directed graph $G = (V, E)$ with non-negative edge weights $w(u, v)$ representing the time it takes to go from a node u to v when u and v are directly connected. (We assume all $w(u, v) < \infty$ for simplicity.) Some subset F of vertices in G are pharmacies, while some other subset D are doctor offices. You live at a node s and have a medical emergency. You now need to go to some doctor office $d \in D$ to get a prescription for the drug, then to some pharmacy $f \in F$, and finally back home. Recall that a proper output must be the full path described above. However some paths are better then others. Solve the following variants of this problem, where each variant describes how to compare different valid paths when choosing the optimal path you are looking for. Each variant could be solved by one "clever" run of the Dijkstra's algorithm, on an appropriate modification of our graph G. You have to explain your choice and state the resulting running time as the function of n (remember, we assume $m = \Theta(n^2)$ here, as $w(u, v) < \infty$).

(a) (3 points) Assume only the distance from your home s to the doctor's office matters (e.g., you need to get doctor's emergency help asap, and the time it then takes to the pharmacy and back home are not important).

Solution:

```
map[] doctorPriorityDijkstra(Graph G, vertex source)
    distance[source] = 0
    map[source] = null
    Q = queue.initialize()

    for each vertex v in G
        if (v != source)
            distance[v] = infinity
            map[v] = null
        Q.enqueue(v)
    endFor

    while (!Q.isEmpty)
        u = Q.dequeue()
        for each adjacent vertex v from u
            temp = u.distance + distance(u, v)
            if (temp < distance[v])
                distance[v] = temp
                map[v] = u
    endWhile

    return map
END doctorPriorityDijkstra
```

□

(b) (5 points) Assume only the distance between the doctor office d and the pharmacy f matters. E.g., the doctor would perform some painful procedure, but does not have a tranquilizer to numb the subsequent pain. Thus, you must choose an office $d \in D$ and pharmacy $f \in F$ with the smallest distance between them, but do not care how far d is from s or s is from f. (E.g., if there is a pharmacy in some doctor's office in Antarctica, you do not mind going there, as the answer you care will be 0, even if you live in NYC.)
(**Hint**: Add an artificial source node s' to G and compute the distance from s' to "fill the blank" in your new graph.)

Solution:

```
map[] doctorPharmacyPriorityDijkstra(Graph G, vertex source)
    G.add(node sPrime)
    distance[sPrime] = 0
    map[sPrime] = null
    Q = queue.initialize()

    for each vertex v in G
        if (v != sPrime)
            distance[v] = infinity
            map[v] = null
        Q.enqueue(v)
    endFor

    while (!Q.isEmpty)
        u = Q.dequeue()
        for each adjacent vertex v from u
            temp = u.distance + distance(u, v)
            if (temp < distance[v])
                distance[v] = temp
                map[v] = u
    endWhile

    return map
END doctorPharmacyPriorityDijkstra
```

□

(c) (5 points) Assume only the distance from the pharmacy f back to home s matters. E.g., the medicine must be administered by the pharmacist and has some quick side effect, so you must get to bed asap after taking the medicine in the pharmacy.
(**Hint**: Two solutions are possible: one does something to the edges of G, and the other again adds some artificial source similar to part (b).)

Solution:

```
map[] pharmacyHomePriorityDijkstra(Graph G, vertex source)
    G.add(node sPrime)
    distance [sPrime] = 0
    map[sPrime] = null
    Q = queue.initialize()

    for each vertex v in G
        if (v != sPrime)
            distance[v] = infinity
            map[v] = null
        Q.enqueue(v)
    endFor

    while (!Q.isEmpty)
        u = Q.dequeue()
        for each adjacent vertex v from u
            temp = u.distance + distance(u, v)
            if (temp < distance[v])
                distance[v] = temp
                map[v] = u
    endWhile

    return map
END pharmacyHomePriorityDijkstra
```

□

(d) (7 points) Finally, assume that the *overall* trip time from s to d to f to s matters. Show how to combine the ideas in parts (a)-(c) to solve this variant.
(**Hint**: Make several "copies" of G and connect them "appropriately" by 0-weight edges. The copies will ensure that every valid path must pass through a doctor office followed by a pharmacy.)

Solution:

```
map[] overallPriorityDijkstra(Graph G, vertex source)
    gPrime = G
    gPrime2 = G
    gPrime3 = G
    map [] = Arraylist.initialize()
    map1 [] = Arraylist.initialize()
    map2 [] = Arraylist.initialize()
    overallMap [] = Arraylist.initialize()

    map = doctorPriorityDijkstra(gPrime, v)
    map2 = doctorPharmacyPriorityDijkstra(gPrime2, v)
    map3 = pharmacyHomePriorityDijkstra(gPrime2, v)

    overallMap = map + map2 + map3
    return overallMap

END overallPriorityDijkstra
```

□

CSCI-UA.0310-004/005 Basic Algorithms May 11, 2015

Solutions to Problem 2 of Homework 12 (12 points)

Name: Jason Yao *Due: Monday, May 11*

Consider the medical emergency problem, except some of the edge weights $w(u, v)$ in the original graph could be negative (but no negative cycles exist).

(a) (4 points) Which of your solutions for parts (a)-(d) of the previous problem would still work if you run Bellman-Ford instead of Dijkstra on the graphs you constructed? For those that work, please state the new running time. For those that do not, state the running time of the fastest algorithm you can find.

Solution: ********** INSERT YOUR SOLUTION HERE ********** □

(b) (4 points) Assume now that the original graph G is a acyclic. Which of your solutions for parts (a)-(d) of the previous problem would still work if you run "one iteration" of Bellman-Ford instead of Dijkstra on the graphs you constructed? For those that work, please state the new running time. For those that do not, state the running time of the fastest algorithm you can find.

Solution: ********** INSERT YOUR SOLUTION HERE ********** □

(c) (4 points) Coming back to general (possibly cyclic) graphs, assume that you want to solve part (d) of the previous problem (but now with negative edge weights) for all possible home locations s, and also that we drop the assumption that $w(u, v) < \infty$. Design the fastest algorithm you can find, and state its running time as a function of n and m (which can now be much less than n^2).

Solution: ********** INSERT YOUR SOLUTION HERE ********** □

CSCI-UA.0310-004/005 Basic Algorithms May 11, 2015

Solutions to Problem 3 of Homework 12 (16 points)

Name: Jason Yao *Due: Monday, May 11*

You are given a directed graph $G = (V, E)$ representing some financial choices. Each edge $(u, v) \in E$ has a weight $w(u, v)$, where $w(u, v) > 0$ represents a cost, and $w(u, v) < 0$ represents a profit. Your initial portfolio is a vertex $s \in V$, and at each step you are allowed to go from your current node $u \in V$ to a neighboring node $v \in Adj(u)$, incurring a cost $w(u, v)$ if $w(u, v) > 0$, or a profit $-w(u, v)$ otherwise.

(a) (4 points) We say that a vertex s is *super-lucky* if s itself is part of a cycle C of negative weight, so that starting from s one can repeatedly come back to s with some profit. Using the "matrix multiplication" approach, design $O(n^3 \log n)$ algorithm to find all super-lucky vertices.

Solution: ********** INSERT YOUR SOLUTION HERE ********** □

(b) (4 points) Say that s is *lucky* if there exists a way to eventually make unbounded profit starting from s (but not necessarily coming back to s infinitely many times as with super-lucky vertices). Give the fastest algorithm you can for finding all lucky vertices (from scratch, without assuming you already solved part (a)).

Solution: ********** INSERT YOUR SOLUTION HERE ********** □

(c) (4 points) Solve the same problem as in part (b), but assuming somebody already gave you for free the list of all super-lucky vertices (or, alternatively, you already ran your solution in part (a), and want to use it to compute lucky vertices faster). Namely, give the fastest algorithm you can think of for finding all lucky vertices given all super-lucky vertices.
(**Hint**: Make sure you use super-lucky vertices instead of computing from scratch!)

Solution: ********** INSERT YOUR SOLUTION HERE ********** □

(d) (4 points) Assume s is not lucky (and, hence, not super-lucky). Design a fast algorithm to compute the best finite "financial strategy" to make as much profit starting from s as possible. State the running time of your algorithm.
(**Hint**: Think Bellman-Ford.)

Solution: ********** INSERT YOUR SOLUTION HERE ********** □

CSCI-UA.0310-004/005 Basic Algorithms May 11, 2015

Solutions to Problem 4 of Homework 12 (12 points)

Name: Jason Yao *Due: Monday, May 11*

Let $G = (V, E)$ be a directed graph with weighted edges; edge weights could be positive, negative or zero.

(a) (4 points) Describe an $O(n^2)$ algorithm that takes as input $v \in V$ and returns an edge weighted graph $G' = (V', E')$ such that $V' = V \setminus \{v\}$ and the shortest path distance from u to w in G' for any $u, w \in V'$ is equal to the shortest path distance from u to w in G.

Solution: ********** INSERT YOUR SOLUTION HERE ********** □

(b) (4 points) Now assume that you have already computed the shortest distance for all pairs of vertices in G'. Give an $O(n^2)$ algorithm that computes the shortest distance in G from v to all nodes in V' and from all nodes in V' to v.

Solution: ********** INSERT YOUR SOLUTION HERE ********** □

(c) (4 points) Use part (a) and (b) to give a recursive $O(n^3)$ algorithm to compute the shortest distance between all pairs of vertices $u, v \in V$.

Solution: ********** INSERT YOUR SOLUTION HERE ********** □