CSCI-GA.1170-001/002 Fundamental Algorithms December 4, 2014

Solutions to Problem 1 of Homework 11 (15 points)

Name: Keeyon Ebrahimi *Due: Wednesday, December 5*

Dijkstra's algorithm solves the single-source shortest-path problem on a weighted directed graph $G = (V, E)$, when all edge weights are non-negative. Suppose we wish to modify the algorithm so that it works on graphs which has negative weight edges as long as there is no *negative cycle*. Consider the following modified Dijkstra's algorithm.

ModifiedDijkstra(G, w, s)
1. Initialize-Single-Source(G, s)
2. $S = \emptyset$
3. $Q = G.V$
4. **While** $Q \neq \emptyset$ **Do**
5. $\quad u =$ Extract-Min(Q)
6. $\quad S = S \cup \{u\}$
7. $\quad$ **For** each vertex $v \in G.Adj[u]$
8. $\quad\quad$ **If** $v.d > u.d + w(u, v)$
9. $\quad\quad\quad v.d = u.d + w(u, v)$
10. $\quad\quad\quad v.\pi = u$
11. $\quad\quad\quad$ **If** $v \in Q$, **Then** Decrease-Key$(Q, v, v.d)$
12. $\quad\quad\quad$ **Else** Insert(Q, v) and $S = S \setminus \{v\}$.

(a) (3 points) Note that the only step where the above algorithm differs from the original Dijkstra algorithm is in Step 12. Give an example with the smallest possible number of vertices to show that if we remove step 12 from ModifiedDijkstra, then it does not solve the single-source shortest-path problem if the edge weights may be negative, even if there are no negative weight cycle.

Solution:
Here is an example demonstrating when Dijkstra's doesn't work with negative edges.

Lets say we have 3 nodes, A, B, C, with the source being A

$A \to B$ with weight 2.
$A \to C$ with weight 3.
$C \to B$ with weight -10.

Bellman-Ford with A source: B distance $= -7$

(i) A *distance* : 0
A is source

(ii) $C\ distance:\ 3$
$A \to C$

(iii) $B\ distance:\ -7$
$A \to C \to B$

Dijkstra's with A source: B distance $= 2$

1. With this graph, the priority queue will dequeue node B first. Set it's distance to 2. B has no outgoing edges as well, so it will not enqueue up anything new.
2. We will then dequeue C, and give it a correct distance of 3. We then check all of C's outgoing edges, and we will find a path to B with a total of -7. B is no longer in the priority queue though, it will not have it's key decreased. Without line 12 it will not be placed back in the queue as well, making B keeps its distance as 2.

As we can see, Dijkstra's algorithm will give us the incorrect distance for node B when we have a negative edge weight. □

(b) (3 points) Assume that the input graph G has no negative weight cycle, although there may be some edges with negative weight. Show that the total number of times the value of $v.d$ changes in the above algorithm is finite. Hence argue that the algorithm terminates in a finite number of steps.

Solution:
There are a few points we need to prove this.

1. In order to change $v.d$, we must decrease the cost of a path to $v.d$. This if from line 8 of the algorithm

2. The total amount of paths that can lead to $v.d$ is E. Path's are constructed of edges, making the maximum amount of paths be E.

3. This means that we must visit the same edges infinite times if we want to infinitely change the value of $v.d$.

4. The only time we can visit the same more than once is if a cycle exists.

5. Cycles can only be negative cycles or positive cycles.

6. Positive cycles mean they add a positive value to the total cost of the path. We only change $v.d$ if the cost to the path to $v.d$ is decremented, meaning we will not change $v.d$ in a positive cycle, meaning that positive cycles would still lead to a finite amount of $v.d$ changes.

7. The only other option is a negative cycle. A negative cycle can repeat multiple times, and also by definition, a negative cycle will decrease the total cost. This means that a negative cycle will infinitely decrease the total path cost value, which in tern could potentially infinitely decrease the value $v.d$.

Summary We either have a cycle or we don't. If there is no cycle, we cannot infinitely change $v.d$, as explained in proof line 5. If there is a cycle, we can only have a positive and negative cycle. Positive cycles cannot cause infinite amount of $v.d$ changes as shown in proof line 6. Negative cycles can change $v.d$ infinite amount of times as shown in proof line 7. We have exhausted all possible outcome, and have deduced that the only potential infinite changes to $v.d$ is only when there are negative cycles. $\square$

(c) (4 points) Notice that for all $v \in V$, the value of $v.d$ is at least the shortest distance from s to v in an execution of the above algorithm. Argue that when the algorithm terminates, for all $v \in V$, the value of $v.d$ is equal to the shortest distance from s to v. Hence conclude the above algorithm correctly solves the single-source shortest path problem even for graphs with negative weight, as long as there is no negative weight cycle. Explicitly state where you need step 12 of the above algorithm in your proof.

Solution: ***************** INSERT PROBLEM 1c SOLUTION HERE *************** $\square$

(d) (5 points) Consider an example of a graph G with vertices $(s, v_1, \ldots, v_n)$ and edge weights $w(s, v_i) = 0$, and $w(v_i, v_j) = -2^{-i}$ for all $1 \leq i < j \leq n$. By finding an appropriate recurrence relation, show that ModifiedDijkstra takes $\Omega(2^n)$ time in the worst case, when executed on (G, s, w).

Solution: ***************** INSERT PROBLEM 1d SOLUTION HERE *************** $\square$

CSCI-GA.1170-001/002 Fundamental Algorithms December 4, 2014

Solutions to Problem 2 of Homework 11 (14 Points)

Name: Keeyon Ebrahimi *Due: Wednesday, December 5*

John, who lives in a node s of a weighted undirected graph G (with non-negative weights), is getting late and has to reach the venue of his high school final exam at node h as soon as possible. However, he has to buy some pencils on his way to the examination hall. He can get pencils at any stationary, and the stationary shops form a subset of the vertices $B \subset V$. Thus, starting at s, he must go to some node $b \in B$ of his choice, and then head from b to h using the shortest total route possible (assume he wastes no time in the stationary). Help John to reach the examination hall as soon as possible, by solving the following sub-problems...

(a) (2 points) Compute the shortest distance from s to all stationary shops $b \in B$.

Solution: ***************** INSERT PROBLEM 2a SOLUTION HERE *************** $\square$

(b) (4 points) Compute the shortest distance from every stationary shop $b \in B$ to h. Can one simply add a new "fake" source s' connected to all stationary shops with zero-weight edges and run Dijkstra from s'?

Solution: ***************** INSERT PROBLEM 2b SOLUTION HERE *************** $\square$

(c) (2 points) Combine parts (a) and (b) to solve the full problem.

Solution: ***************** INSERT PROBLEM 2c SOLUTION HERE *************** $\square$

(d) (6 points) Your solution in part (c) used two calls to the Dijkstra's algorithm (one in part (a) and one in part (b)). Define a new graph G' on at most $2n$ vertices and at most $2m + n$ edges (and "appropriate" weights on these edges), so that the original problem can be solved using a *single* Dijkstra call on G'.

Solution: ***************** INSERT PROBLEM 2d SOLUTION HERE *************** $\square$

CSCI-GA.1170-001/002 Fundamental Algorithms	December 4, 2014

Solutions to Problem 3 of Homework 11 (12 points)

Name: Keeyon Ebrahimi *Due: Wednesday, December 5*

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 PROBLEM 3a 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). Assume also you know all super-lucky vertices. Give the fastest algorithm you can for finding lucky vertices given super-lucky vertices. State its running time as a function of m and n.
(**Hint**: Make sure you use super-lucky vertices instead of computing from scratch.)

Solution: ***************** INSERT PROBLEM 3b SOLUTION HERE *************** □

(c) (4 points) Assume s is not lucky (or super-lucky). Design the best finite strategy to make as much profit starting from s as possible. State the running time of your algorithm.
(**Hint**: Think Bellman-Ford.)

Solution: ***************** INSERT PROBLEM 3c SOLUTION HERE *************** □

CSCI-GA.1170-001/002 Fundamental Algorithms	December 4, 2014

Solutions to Problem 4 of Homework 11 (22 points)

Name: Keeyon Ebrahimi — *Due: Wednesday, December 5*

You are given a map $G = (V, E)$ with cities V connected by roads E. Each road (edge) is labeled with a weight which is a real number. You are located in city $s \in V$. You are also given an array A of boolean values that tells you if there is a toll on that road. More precisely, for road $e \in E$ we have $A[e] = 1$ if and only if there is a toll on e. Being the low budget traveler that you are, your budget allows for at most one such toll to be payed for any given trip (path).

Let $d(s,t)$ denote the minimum possible sum of weights of a path from s to t for any $s, t \in V$. If there is no path from s to t, then $d(s,t) = \infty$, and if there is a path from s to t that contains a negative cycle, then $d(s,t) = -\infty$ (since one can cycle along the path an arbitrary number of times).

Similarly, let $c(s,t)$ denote the minimum possible sum of weights of a path from s to t that passes through at most 1 toll.

(a) (10 points) Construct a graph $G' = (V', E')$ and mappings $f : V \mapsto V'$, $g : V \mapsto V'$ such that $|V'| = 2|V|$, $|E'| \leq 2|E| + |V|$ and for any $s, t \in V$, $c(s,t) = d(f(s), g(t))$. Namely, you reduce the "constrained" problem on G to "unconstrained" problem in G'. Remember to consider the case when $c(s,t)$ is $-\infty$, and prove the correctness of your solution.

Solution: ***************** INSERT PROBLEM 4a SOLUTION HERE ***************

□

(b) (3 points) Give an algorithm that takes as input s and finds a shortest path with at most one toll road from s to all cities in V. Analyze the running time of your algorithm.

Solution: ***************** INSERT PROBLEM 4b SOLUTION HERE ***************

□

(c) (3 points) Now assume your buddy Billybob who works at a major airlines company has given you a free plane ticket to any city in V, meaning you can start your trip at any node. As before, once you start your road trip, you are still refusing to pay more then a single toll on any such trip. To help plan the trip, your job is to give an algorithm to find a shortest path with at most one toll road between *all pairs* of cities in V. Analyze the running time of your algorithm. (**Hint**: Remember Johnson.)

Solution: ***************** INSERT PROBLEM 4c SOLUTION HERE ***************

□

(d) (6 points) Here you will solve the problem in part (c) directly on graph G, without constructing the helper graph G'. Let $W = \{w(i,j)\}$ be the original edge weight matrix and $W' = \{w'(i,j)\}$ be the same edge matrix except we replace $w'(i,j) = \infty$ if $A(i,j) = 1$ (i.e., never use toll roads in W'). For simplicity, assume W' is pre-computed for you. For $0 \leq k \leq n$, let

– D^k be the matrix of all shortest distances w.r.t. W' which only use nodes $\leq k$ as intermediate nodes.

– C^k be the matrix or all shortest distances w.r.t. W which only use nodes $\leq k$ as intermediate nodes, but *also use at most one toll.*

Fill in the blanks below to directly modify the Floyd-Warshall algorithm to compute the correct answer for problem (c) in time $O(n^3)$. Argue the correctness of your algorithm.

Floyd-Warshall(W, W')
  $n = W.rows$
  $D^0 = ...$
  $C^0 = ...$
  For $k = 1$ **to** n **Do**
    $C^k = (c^k_{i,j})$ be a new $n \times n$ matrix
    $D^k = (d^k_{i,j})$ be a new $n \times n$ matrix
    For $i = 1$ **to** n **Do**
      For $j = 1$ **to** n **Do**
        $d^k_{i,j} = \min(\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots)$
        $c^k_{i,j} = \min(\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots)$
  Return ....

Solution: **************** INSERT PROBLEM 4d SOLUTION HERE ***************

□