CS 670 Spring 2015 - Solutions to HW 6

Problem 24.1-3

We know that in the Bellman Ford Algorithm, values $d[v]$ converge to length of a shortest path, from the source to node v. Suppose in a certain pass none of the $d[v]$'s were changed, then in subsequent passes there will be no change either. Hence making this check in the algorithm will ensure that the procedure terminates in $m + 1$ or fewer passes.

Problem 24.1-4

We modify the Bellman Ford Algorithm to obtain our new algorithm. Instead of exiting at the first node where the triangle inequality is not satisfied, we keep on going marking all the nodes for which the triangle inequality is not satisfied. Once through we find all nodes reachable from the set of the marked nodes. This can be done by doing a BFS or DFS from the marked nodes. The correctness of the algorithm follows from the fact that each negative weighted cycle has at least one marked node.

Correctness: In the algorithm, for every negative cycle reachable from the source, at least one edge is marked. For every node v reachable from the marked nodes, the algorithm sets $d[v] = -\infty$.

We claim that every vertex marked can be reached from s through a negative cycle. Lets assume that the claim is false. Thus a vertex b was marked and there exists no negative cycle through which b can be reached from s. Say, the edge (a, b) violated the triangle inequality and so $d[b] > d[a] + w(a, b)$. Clearly, a cannot be reached through a negative cycle either. From the proof of correctness of the Bellman-ford algorithm (Thm 24.4 in text), $d[a] = \delta(s, a)$ and $d[b] = \delta(s, b)$. Thus $\delta(s, b) > \delta(s, a) + w(a, b)$, a contradiction. Hence our claim is true.

If the algorithm sets $d[v] = -\infty$, then v can be reached from s through a negative cycle. This is true because, all vertices for which $d[v] = -\infty$ were reached through a marked vertex which (from our claim) can be reached through a negative cycle.

If v can be reached from s through a negative cycle, then the algorithm sets $d[v] = -\infty$. This is true because, every negative cycle reachable from s has at least one vertex marked. Hence $d[v] = \infty$.

Problem 24.3-10

Recall in the proof (Thm 24.6 in the text) of the correctness of Dijkstra's algorithm, we prove that the following loop invariant holds:

At the start of each iteration of the while loop of lines $4 - 8$, $d[v] = \delta(s, v)$ for each vertex $v \in S$.

In other words when a vertex is added to the set S, we have determined a shortest path from the source to the particular vertex. If there were negative edges present in the graph, the concern (†) is that for $v \in S$ there might be a path from s to v through some other node $u \notin S$ that is shorter than path associated to $d[v]$ which was computed.

In the context of our problem, when we perform EXTRACT-MIN(Q) for the second time (with $S = \{s\}$) we will be including a vertex u, such that the edge-weight $w(s, u)$ is possibly negative but bear in mind that this is the minimum and therefore the scenario (†) does not arise and similar reasoning holds for other (adjacent to the source) vertices which are included in S.

Problem 24-3

a

Starting with the *multiplicative* property (◇) stated in the problem which we have to detect and applying logarithms and some simple manipulation we obtain an equivalent *additive* property (◇◇):

$$\begin{array}{rlcl} & R[i_1, i_2] \cdot R[i_2, i_3] \ldots R[i_{k-1}, i_k] \cdot R[i_{k-1}, i_1] & > & 1\ (\diamond) \\ \Leftrightarrow & \log(R[i_1, i_2]) + \log(R[i_2, i_3]) \ldots \log(R[i_{k-1}, i_k]) + \log(R[i_{k-1}, i_1]) & > & 0 \\ \Leftrightarrow & \log(R[i_1, i_2]^{-1}) + \log(R[i_2, i_3]^{-1}) \ldots \log(R[i_{k-1}, i_k]^{-1}) + \log(R[i_{k-1}, i_1]^{-1}) & < & 0\ (\diamond\diamond) \end{array}$$

Algorithm: We construct a table R' where if $R[i, j] \neq 0$ then $R'[i, j] = \log R[i, j]^{-1}$, otherwise set $R'[i, j] = \infty$.

We construct a graph with currencies as vertices and with edges/edge weight given by the table R' and run Floyd-Warshall to detect a negative cycle (see Q25.2-6).

Alternatively, we could also use Bellman-Ford with the table R with making appropriate modifications realizing that the property is multiplicative (and not additive).

b

Algorithm:

```
 1. INITIALIZE-SINGLE-SOURCE(G,s)
 2. for $i \leftarrow 1$ to $|V(G)| - 1$
 3.   do for each edge $(u, v) \in E(G)$
 4.     do $RELAX(u, v, w)$
 5. for each edge $(u, v) \in E(G)$
 6.   do if $d(v) > d(u) + w(u, v)$
 7.      then mark $v$
 8.         $x \leftarrow v$
 9.         while $\pi(x)$ is not marked
10.           do mark $\pi(x)$
11.             $x \leftarrow \pi(x)$
12.         return the cycle in marked nodes
13. return NIL
```

Proof: Call a path $p : x_1, ..., x_k$ (with $x_i \in V$) a π-path if $\pi(x_i) = x_{i-1}$ for $i = 2, ..., k$.
Claim Suppose $p : s = x_1, ..., x_k$ is a simple π-path after the Bellman-Ford is run on G. Then $d(x_k) = w(p)$. The Claim is trivial for $k = 1$. Inductively suppose $d(u) = w(p_u)$ where $u = x_{k-1}$ and p_u denotes the subpath of p from s to x_{k-1}. Let $v = x_k$. At sometime $\pi(v)$ became u and from that point on $\pi(v)$ has not changed. This means $d(u) + w(u, v) = d(v)$ right after $\pi(v)$ became u, and since $\pi(v)$ has not changed since then, $d(v)$ has not changed either. Meanwhile $d(u)$ can either remain the same or become smaller, therefore at the end we have $d(u) + w(u, v) \leq d(v)$. Therefore, $w(p) = w(p_u) + w(u, v) = d(u) + w(u, v) \leq d(v)$. On the other hand we recall that for all vertex x, after i iterations in the running of Bellman-Ford, $d(x) \leq w(q)$ for all paths q from s to x with no more than i edges. Therefore $d(v) \leq w(p)$ since p is simple. We conclude that $w(p) = d(v)$ hence the Claim.

Now suppose the triangle inequality is violated at an edge (u, v), that is $d(u) + w(u, v) < d(v)$. Tracing the predecessors of u via π, we are either led through a simple path to s, or to some vertex x and a cycle containing x.

In the first case we have a simple π-path $p : s \overset{*}{\to} u$. In this case v must lie on the path, for otherwise the path p followed by (u, v) is also simple, hence $w(p) + w(u, v) \geq d(v)$. On the other hand, we have $d(v) > d(u) + w(u, v) = w(p) + w(u, v)$, hence a contradiction.

Since v lies on p, the subpath from v to u followed by (u, v) is a cycle C. It is in fact a negative cycle. This is because $w(p_v) = d(v)$ and $w(p_u) = d(u)$ by the Claim, but now $d(v) > d(u) + w(u, v) = w(p_v) + w(C) = d(v) + w(C)$. Hence $w(C) < 0$.

In the second case we have a π-cycle $x_0, x_2, ..., x_k = x_0$, and without loss of generality we may assume that $\pi(x_0)$ became x_{k-1} last to complete the π-cycle. Right after x_i became $\pi(x_{i+1})$, $d(x_i)+w(x_i, x_{i+1}) = d(x_{i+1})$. Since then $d(x_{i+1})$ has not changed since $\pi(x_i)$ has not changed. Meanwhile $d(x_i)$ has either stayed unchanged or become smaller. Hence right before $\pi(x_{k-1})$ became x_0, $d(x_i) + w(x_i, x_{i+1}) \leq d(x_{i+1})$ for $i = 0, ..., k - 2$. At that time, $d(x_{k-1}) + w(x_{k-1}, x_0) < d(x_0)$. Consequently,

$$\sum_{i=0}^{k-1} d(x_i) + w(x_i, x_{i+1}) < \sum_{i=0}^{k-1} d(x_{i+1}).$$

From this it follows that the weight of the π-cycle is negative.

The relax step costs $O(VE)$, the length of a shortest path is $O(V)$, so the second part of the algorithm, which finds a negative cycle, also costs $O(VE)$, thus the running time of the algorithm is $O(VE)$.

25.2-6

We say a path is of type k if all immediate vertices are within the set $\{1, ..., k\}$, then the following properties hold:

- $d_{ij}^{(k)} \leq w(p)$ for all simple paths p of type k from i to j.
- there exists a path of type k from i to j with the weight $d_{ij}^{(k)}$.

Claim. There exists an i satisfying if $d_{ii}^{(n)} < 0$ if and only if there exists a negative-weight cycle.

Proof. If there exists an i such that $d_{ii}^{(n)} < 0$ then there exists a negative-weight cycle, as there is a path from vertex i back to itself whose weight is negative. If there exists a negative-weight cycle then clearly there exists a minimal cycle of negative weight whose path length is less than or equal to n. Therefore there exists an i satisfying if $d_{ii}^{(n)} < 0$.

Algorithm. For each i, check if $d_{ii}^{(n)}$ is less than 0.

Problem 25.3-5

We know that for a path $p = \{v_0, \ldots, v_k\}$, $\hat{w}(p) = w(p) + h(v_0) - h(v_k)$. Since for a cycle $h(v_0) = h(v_k)$ and moreover $w(c) = 0$ for a 0-weight cycle c, hence, $\hat{w}(c) = 0$. This in conjunction with the fact that the values of $\hat{w}$ is non-negative for all edges, leads to the fact that the value of $\hat{w}$ for each edge in a 0-weight cycle is 0.