Homework 11, Solution set

1. We augment the standard Dijkstra's algorithm. Note that the path from s to v via a node u costs $\text{Dist}(u) + \ell(u,v)$. If this value was equal to $\text{Dist}(v)$, then there is a node $u' \neq u$ such that the path from s to v via u' has the same cost as the one from s to v via u. Clearly, this means that there is more than one path from s to v with cost $\text{Dist}(u) + \ell(u,v)$. We record whether a unique shortest path to v has been found in the array $\text{USP}[1:n]$. This leads to the following changes to Dijkstra's algorithm (underlined).

```
procedure Dijkstra($G, s$)
    for $v = 1$ to $n$ do
        $\text{Dist}[v] = \infty$; $\underline{\text{USP}[v] \leftarrow \text{False}}$
    end for
    $\text{Dist}[s] = 0$; $\underline{\text{USP}[s] \leftarrow \text{True}}$;
    $Q = \text{BuildQueue}(V, \text{Dist})$;
    while $Q$ is not empty do
        $u = \text{DeleteMin}(Q)$
        for all edges $(u,v) \in E$ do
            if $\text{Dist}[v] > \text{Dist}[u] + \ell(u,v)$ then
                $\text{Dist}[v] \leftarrow \text{Dist}[u] + \ell(u,v)$;
                $\underline{\text{USP}[v] \leftarrow \text{USP}[u]}$;
                $\text{ReduceKey}(v, \text{Dist}[v])$
            else
                if $\text{Dist}[v] = \text{Dist}[u] + \ell(u,v)$ then $\underline{\text{USP}[v] \leftarrow \text{False}}$
                end if
            end if
        end for
    end while
end procedure
```

However, this modification may not handle the case with 0-length edges correctly. The problem is that we may process the outedges from v when it appears to have a unique path from s, and later, due to a 0-length edge into v we may need to update the neighbors of v to be non-unique.

To handle this we also compute the array $\text{Pred}[1:n]$ in our augmented Dijkstra algorithm, and in a postprocessing phase we build a tree of shortest paths from s using this array. We then traverse the tree from its root s, updating $\text{USP}[v]$ to be $\text{USP}[v] \wedge \text{USP}[\text{Pred}[v]]$. This causes any late discovery of non-uniqueness of shortest paths to some vertex w to propagate to all of w's descendants in the shortest path tree.

The processing per edge in Dijkstra's algorithm remains at an $O(1)$ cost. The processing per vertex is unchanged. The construction of the shortest path tree and its subsequent traversal take $O(n)$ time, or equivalently $O(1)$ per vertex. Therefore the overall algorithm has the same runtime as Dijkstra's algorithm up to constant factors.

2. Notice that the cheapest path from s to v uses the subway either once or not at all.

Accordingly, we create the following graph H. It comprises G plus one additional vertex z, which indicates that one is in the subway system. We then add the following additional edges: for each $v \in S$, cost \$2.75 edges (v,z), which correspond to entering the subway, and cost \$0 edges (z, v), which correspond to traveling in the subway and exiting the subway system at vertex v.

We run Dijkstra's algorithm on graph H with start vertex s. We then report, for each vertex $v \in V$, the length of the shortest path from s to v in H.

Notice that a shortest path in H from s to v comprises an initial path of taxi edges ending at some vertex w, possibly followed by new edges (w, z) and (z, x), followed by another path of taxi edges. This corresponds to the journey in G of taking a taxi from s to w, the subway from w to x, and a taxi from x to v, and the two routes have the same cost. Thus the above algorithm has computed the shortest paths for our problem.

H has $n+1$ vertices and $m+2n$ edges. Building H from G will take $O(n+m)$ time. Therefore, the overall runtime is at most a constant factor larger than the runtime of Dijkstra's algorithm on a graph of n vertices and m edges (note that by assumption, $n \leq m$).

3.i. We compute the time at which each vertex catches fire by applying Dijkstra's algorithm to G with start vertex c. We store these values in the array OnFire$[1 : n]$.
ii. Our modification of Dijkstra's algorithm is to ignore a vertex once it catches fire. What this means is that if at the time a vertex u is removed from the Priority Queue by a DeleteMin operation, if it is already on fire, u is discarded and is not processed; i.e. it is ignored by Dijkstra's algorithm.

This modification of Dijkstra's algorithm ensures that only vertices that have not caught fire are included in the computation of shortest paths; i.e. it is now computing shortest safe paths. We show the small modification to the standard algorithm below.

```
$u \leftarrow$ DeleteMin($Q$);
**if** Dist[$u$] $<$ OnFire[$u$] **then**
    **for** each edge $(u, v)$ **do**
        the rest of the algorithm is as before
    **end for**
**end if**
```

The asymptotic runtime of the modified Dijkstra's algorithm in part (ii) is the same as for the unmodified algorithm, as the additional test adds $O(1)$ cost per edge. Thus the overall runtime is the cost of two runs of Dijkstra's algorithm, which is the same as the upper bound on the cost of one run on a graph of n vertices and m edges, up to constant factors. Note that in order to compute the actual paths, we need to compute the Pred array, and we can then output the tree of paths as in Problem 4, homework 10, though here we need a single path per vertex v, not all of the shortest paths to v. Note also that for some vertices there may be no path.

4. We build the same graph H as in Problem 2 in the additional problems. As argued there, a shortest even number of edges path in G from u to v has the same length as a shortest path from u^e to v^e in H (all that changes is that we replace s by u in the argument).

Building H and its associated weight matrix W_H is readily done in $O(n^2)$ time as H has $2n$ vertices.

We then run the Floyd-Warshall algorithm on H. It will take $O(n^3)$ time.

Finally, for each pair of vertices $u, v \in V_G$, we report the length of the shortest path from u^e to v^e in H. This takes a further $O(n^2)$ time.

Thus, the overall run time is $O(n^3)$.