Problem Set 6

Name: Gabriel Chiong

Collaborators: None

Problem 6-1.

(a) DAG Relaxation order: $[(a,b),(a,d),(d,b),(d,e),(b,c),(b,e),(b,f),(e,f),(f,c)]$
Dijkstra order: $[(a,b),(a,d),(b,c),(b,e),(b,f),(f,c),(e,f),(d,b),(d,e)]$

(b) Both algorithms output the same shortest-path results:
$\delta(a,a)=0$
$\delta(a,b)=2$
$\delta(a,c)=3$
$\delta(a,d)=6$
$\delta(a,e)=5$
$\delta(a,f)=4$

Problem 6-2. We assume that the graph could contain negative cycles and negative weights, since this is not mentioned in the problem statement. We are given that the shortest path to each vertex traverses at most k edges, but we do not know what k is. Since we can obtain the shortest paths using at most k' edges (the k'-th layer of the constructed graph G' for Bellman-Ford), we can modify Bellman-Ford to only construct the duplicated graph layers incrementally then relaxing edges into the new layer, rather than constructing the entire graph G' initially. Since we do not know the value of k, the criteria for termination will be when the k'-edge shortest path distance do not change between layers.

We prove by induction on layer i of G' that this algorithm correctly computes $\delta(s_0, v_{i+1})$ for all $v \in V$. We first initialize G' as containing only the first layer ($v_0 : \forall v \in V$) and δ initialized to 0 for (s_0, s_0), and ∞ otherwise. This proves the base case, since in layer $i = 0$, only the source is reachable ($\delta(s_0, s_0) = 0$), and every other vertex is not reachable ($\delta(s_0, v_0) = \infty$, for all $v \in V$).

In the inductive step, we assume that G' contains layers from $0 \ldots i$ and $\delta(s_0, v_j)$ have been computed for all $v \in V$ and $j \in \{0, \ldots, i\}$. We append layer $i + 1$ by adding vertices v_{i+1} for $v \in V$ and their associated edges from layer i. This new layer does not affect distances in previous layers since they come later in the topological sort order. So applying DAG Relaxation on the new edges out of the vertices in layer i correctly computes $\delta(s_0, v_{i+1})$ for all $v \in V$.

Since we know that the shortest path from s to any vertex uses at most k edges, we will eventually reach the layer i with at most k' edges such that $k' \geq k$ and $\delta(s_0, v_{k'}) = \delta(s_0, v_{k'+1})$ for all $v \in V$. When this condition first occurs, we terminate the algorithm and output $\delta(s, v) = \delta(s_0, v_k)$ for all $v \in V$.

We need to construct $k + 1$ layers of G, and each layer takes $O(|E|)$ time for DAG relaxation, therefore, this algorithm runs in $O(|V| + k|E|)$ time in the worst case, as required. Since the graph is connected, $|V| = O(|E|)$, and the running time can be abbreviated to $O(k|E|)$.

Problem 6-3. We first construct a graph G_1, where the clearings of the problem are represented as vertices, and slopes as edges. We only include those clearings and slopes which have a decreasing elevation from one to another. We also construct a graph G_2, which is constructed similarly to G_1, except that it uses the elevation of clearings after the dynamite located in that clearing has been detonated.

We add a directed edge between D_1 and D_2 the vertices of G_1 and G_2 respectively, where the detonator is located. This edge represents Bames choosing to detonate, and has a weight of 0. We also add a supernode S, which has two incoming edges of weight 0 from S_1 and S_2, which represent Bames reaching the target either through not detonating (S_1) or detonating (S_2). Let us call this combined graph G.

Since the elevations are strictly decreasing until clearing S_1, S_2, then G must be a DAG. We can use DAG Relaxation from source L_1 to S, with the shortest path being equivalent by definition to the minimum distance for Bames to reach S, with or without detonation, and always on a decreasing elevation path. Graph G has $2n + 1 = O(n)$ vertices and edges, so DAG Relaxation on G runs in $O(n)$.

Problem 6-4. It is easier to determine the existence of non-zero weight cycles, rather than showing that zero weight cycles exist in a graph. Therefore, we construct a graph G with particle locations as vertices and transitions as edges. Add a supernode s with 0 weight to each vertex to ensure that the entire G is a single connected component.

Now we can run Bellman-Ford to detect negative weight cycles in G. Next, negate all edge weights (transition weights) in G and run Bellman-Ford a second time to determine if there are any positive weight cycles in the negated edge weight graph of G. If either of these two runs of Bellman-Ford produce a non-zero weight cycle, then the force field is not conservative. Otherwise, no non-zero weight cycles exist, and we can return that the force field is conservative.

Since there are $n + 1 = O(n)$ vertices in G and $O(n)$ edges, two runs of Bellman-Ford give a worst-case running time of $O(n^2)$ as desired.

Problem 6-5. Create a graph G, with the vertices representing an intersection. Each intersection is represented by g vertices v_i, with $i \in \{1, \ldots, g\}$. Vertex v_i represents being at that intersection with i units of gas in her tank.

Edges in G represent roads. For any bidirectional road connecting intersections u, v, let $t(u, v)$ be the driving time of the road, and $e(u), e(v)$ be the elevation of intersections u, v respectively. Without loss of generality, assume $e(u) < e(v)$ for the three cases when constructing edges in graph G:

- Down-hill road edges: add a weight $t(u, v)$ edge (v_i, u_i) for every $i \in \{1, \ldots, g\}$ since travelling in this direction requires no gas.
- Up-hill road edges: if $e(u) = e(v)$, add an edge in the same way as for down-hill edges. Otherwise, add a weight $t(u, v)$ edge (u_i, v_{i-l}) for every $i \in \{l + 1, \ldots, g\}$ since going up-hill requires gas and she cannot have a negative tank.
- Gas station edges: for each gas station at intersection v, add a weight t_G edge $(v_i, v_{i+1}$ for every $i \in \{1, \ldots, g - 1\}$ corresponding to filling up the tank by one unit at the gas station.

This graph has the property that any path from vertex s_g to any vertex t_i for $i \in \{1, \ldots, g\}$ will be a valid route from source to destination, never having a nonempty tank along the way. The weighted shortest path corresponds to the minimum time required to reach the destination. So we can use a SSSP algorithm from s_g to t_i for all $i \in \{1, \ldots, g\}$, and return a path with minimum weight (or return no such path exists). This graph may have cycles, but only positive edge weights, so Dijkstra can be used.

Since the average degree of each intersection is bounded by a constant and every road connects two intersections, then $2r < 5n$ and $r = O(n)$. Since $g < n$ by the problem statement, G has $O(n^2)$ vertices and $O(n^2)$ edges, so it takes $O(n^2)$ time to construct. Then Dijkstra takes $O(n^2 \log(n^2) + n^2) = O(n^2 \log n)$ time in total, as required.

Problem 6-6. Programming question only.