Jacob Alspaw
EECS 340 Algorithms
Assignment #7

24.3-8

Let G = (V, E) be a weighted, directed graph with weight function $w : E \rightarrow \{0, 1, \ldots, W\}$ for some nonnegative integer W. Modify Dijkstra's algorithm to compute the shortest path weights from a given source vertex $s \in V$ in O(W V + E) time.

DIJKSTRA_ORIGINAL(G , w , s) Total Running Time = O(V^2 + E) *when unmodified*

```
INITIALIZE-SINGLE-SOURCE( G , s )
S = Ø
Q = G.V
while Q ≠ Ø
      u = EXTRACT-MIN( Q )
      S = S ∪ { u }
      for each vertex v ∈ G.Adj[ u ]
            RELAX( u , v , w )
```

The running time of Dijkstra's algorithm unmodified depends on how we implement the min-priority queue. Consider first the case in which we maintain the min-priority queue by taking advantage of the vertices being numbered 1 to |V|. We simply store *v*.d in the v^{th} entry of an array. Each INSERT and DECREASE-KEY operation takes O(1) time, and each EXTRACT-MIN operation takes O(V) time (since we have to search through the entire array), for a total time of O(V^2 + E).

However, if we provide a modification to this algorithm, we can achieve runtime O(W V + E). As a result of our defined weight function, we know that each edge has at most weight W, so the maximum possible value of the longest shortest-path in the graph is (V – 1)W, and also an integer. For the priority queue, we will still prioritize vertices based off of their upper bounds on the weight of a shortest path from the source. Because our longest possible path is (V – 1)W in length, we will need a queue of size (V – 1)W + 2 buckets. Vertex *v* is placed in the *v*.d^{th} bucket of the queue. The source is found in the first bucket of the priority queue. INITIALIZE-SINGLE-SOURCE will have all vertices *v*.d = ∞, excluding the source, and will have the last bucket in the queue reserved for undiscovered vertices. In what will take O(VW) time, we will need to scan the buckets in increasing order, until we encounter a non-empty bucket. The first vertex in the bucket is removed, using the EXTRACT-MIN method. All adjacent vertices are then relaxed. The relaxation still happens over E edges. Therefore our final run time is O(W V + E).

24-1

Suppose that we order the edge relaxations in each pass of the Bellman-Ford algorithm as follows. Before the first pass, we assign an arbitrary linear order $v_1, v_2, \ldots, v_{|V|}$ to the vertices of the input graph G = (V,E). Then, we partition the edge set E into $E_f \cup E_b$, where $E_f = \{(v_i, v_j) \in E : i<j\}$ and $E_b = \{(v_i, v_j) \in E : i>j\}$. Assume that G contains no self-loops, so that every edge is in either E_f or E_b. Define $G_f = (V, E_f)$ and $G_b = (V, E_b)$.

24-1A

Proof by contradiction for G_f: Let us consider that G_f does indeed have a cycle. A cycle would indicate that at least one edge (u,v) would have u with a higher index than v. However, [v_1, v_2, …, $v_{|V|}$] is a topological sort for G_f because from the definition of E_f, the edges are directed from smaller to larger indices. In other words, this edge would have $\{(u_i, v_j) \in E : i>j\}$ which is in contradiction to the definition of E_f which is the edge set for G_f. Therefore, we have a contradiction with the definition of G_f, so the cyclic edge that causes contradiction must not exist. Thus G_f is acyclic.

Proof by contradiction for G_b: Let us consider that G_b does indeed have a cycle. A cycle would indicate that at least one edge (u,v) would have u with a lower index than v. However, [$v_{|V|}$, $v_{|V|-1}$, ..., v_1] is a topological sort for G_b because from the definition of E_b, the edges are directed from smaller to larger indices. In other words, this edge would have $\{(u_i, v_j) \in E : i<j\}$ which is in contradiction to the definition of E_b which is the edge set for G_f. Therefore, we have a contradiction with the definition of G_b, so the cyclic edge that causes contradiction must not exist. Thus G_b is acyclic.

24-1B

Suppose that we implement each pass of the Bellman-Ford algorithm in the following way. We visit each vertex in the order v_1, v_2, …, $v_{|V|}$, relaxing edges of E_f that leave the vertex. We then visit each vertex in the order $v_{|V|}$, $v_{|V|-1}$, ..., v_1, relaxing edges of E_b that leave the vertex.

Assume edges are partitioned into two directed cyclic graphs E_f and E_b. E_f consists of edges that go from lower number vertex to higher [v_1, v_2,…, $v_{|v|}$] and E_b is vice versa [$v_{|v|}$, $v_{|v|-1}$,…, v_1]. First loop through the edges in E_f in topological ordering of E_f. Then loop through the edges in E_b in topological ordering of E_b. In the first part of the loop, any path that is part of the shortest path tree that starts at a vertex with correct labeling and only goes through edges of E_f becomes labeled correctly. In the second part of the loop, any path that starts at a vertex with correct labeling and only goes through edges of E_b becomes labeled correctly. Thus, total number of iterations is the number of E_f-E_b alternations on any given path. This value is at most $|v|/2$ and at this point $v.d = \delta(s,v)$ for all $v \in V$.

24-1C

We would expect to see no improvement to the asymptotic runtime of the Bellman-Ford algorithm attributed by this scheme because even with performing only an upper bound of $|V|/2$ passes, the improvement will still have $O(V)$ passes. The original Bellman-Ford algorithm of $|V|-1$ passes also takes $O(V)$ time to complete. Each pass still takes $\Theta(E)$ time, so the total asymptotic runtime for Yen's improvement and the Bellman-Ford algorithm remains at $O(VE)$.