CSE 331: Introduction to Algorithm Analysis and Design

Dynamic Programming

1 Shortest Path With Negative Edges

1.1 Problem

Input: A weighted graph $G = (V, E)$ (directed or undirected) and an ending node $t \in V$. ~~Each edge $e \in E$ has a weight $l_e > 0$.~~ Each edge $e \in E$ has a weight $l_e \in \mathbb{R}$. Assume that G has no negative cycles.

Output: The length of the shortest path from s to t for all $s \in V$.

1.2 Definitions

Negative Cycle: A cycle where the sum of all the edge weights in the cycle is negative

$OPT(i, u)$**:** The length of the shortest path from u to t with at most i edges.

1.3 Bellman-Ford Algorithm

1. $OPT(t, 0) = 0$
2. $\forall_{u \neq t} OPT(u, 0) = \infty$
3. For $i = 0, ..., n-1$
   (a) For each $v \in V$
      i. $OPT(v, i) = min(OPT(v, i-1), min(l_e + OPT(w, i-1))$ where $e \in E$ and $e = (v, w)$
4. return $OPT(s, n-1)$ for all nodes $s \in V$

1.4 Correctness

Lemma 1. *If G has no negative cycles and nodes s and t are connected then there exists a shortest path from s to t that is simple (no cycles).*

Proof. Assume that there is a shortest path from s to t that contains a cycle. Since a cycle starts and ends with the same node the cycle can be removed and the path still connect s to t. Since the weight of the cycle is ≥ 0 the weight of the new path is less than or equal to the weight of the original path. By removing all the cycles we will have a simple path that is also the a shortest path since the weight did not increase. Therefore, there exists a shortest path connecting s and t that is simple. □

Proof. We want to prove $OPT(v, i)$ is in fact the weight of the shortest path from v to t using at most i edges. We will prove this by induction over i.

Base case: For $i = 0$ the shortest path from t to t is 0. If there were a shorter path from t to t then that path would be a negative cycle. There is no path from any other node to t that takes 0 edges. The algorithm correctly sets these values.

Induction step: We will assume that the algorithm correctly computes $OPT(v, k)$ for all $v \in V$ and show that it correctly computes all $OPT(v, k + 1)$. Let's only consider the first edge in the shortest path from v to t that uses at most $k + 1$ edges. The possibilities for this first edge are all of the outgoing edges e from v to any other node u. Regardless of which edge is chosen the length of each path to t is $l_e + OPT(u, k)$. Whichever node u leads to the minimum weight path is the shortest path from v to t with at most $k + 1$ edges. It is also possible that none of these edges lead to a shorter path than we've already found in which case the shortest path is the same as $OPT(v, k)$. All these cases can be computed since they only depend on $OPT(u, k)$ being correct which is true from the inductive hypothesis. Since these options cover every case for the shortest path, taking the minimum weight path of all these options will be the correct minimum weight. Since our algorithm exactly checks these options and takes the minimum of them it is correctly computing $OPT(v, k + 1)$ for all $v \in V$

Since the algorithm correctly computes $OPT(v, k)$ for all $v \in V$ and all k and by Lemma 1 there exists a shortest path from any s to t that is simple, then $OPT(v, n - 1)$ must be the length of the shortest path from v to t which is what the algorithm returns.

□

1.5 Runtime

1. $OPT(t, 0) = 0$ $O(1)$
2. $\forall_{u \neq t} OPT(u, 0) = \infty$ $O(n)$
3. For $i = 0, ..., n - 1$ $O(n)$ iterations
   (a) For each $v \in V$ $O(n)$ iterations
      i. $OPT(v, i) = min(OPT(v, i-1), min(l_e + OPT(w, i-1))$ where $e \in E$ and $e = (v, w)$ $O(m)$ total to check every edge in the graph while iterating over all the nodes (Amortized $O(m)$)
4. return $OPT(s, n - 1)$ for all nodes $s \in V$ $O(n)$

Total runtime is $O(1) + O(n) + O(n) \cdot ((O(n) + O(m)) + O(n))$ making the overall runtime of $O(n \cdot m)$.