1 Proof of Bellman-Ford Correctness

Bellman-Ford solves the problems that Dijkstra's algorithm has with negative weight edges, by employing a different relaxation ordering on the edges. The pseudocode for the Bellman-Ford algorithm can be found below.

```
for v ∈ V: d[v] ← ∞
d[S] ← 0
for i = 1 to | V | −1
    for each edge (u, v) ∈ E:
        Relax(u, v)
for each edge (u, v) ∈ E
    do if d[v] > d[u] + w(u, v)
        then report a negative-weight cycle exists
```

In plain English, this algorithm relaxes every edge in the graph $|V|$ times and at the end checks for a negative cycle. The gist of the correctness is as follows: Consider a source node s and its immediate neighbors, $v \in N(s)$. As we know from Dijkstra's algorithm, provided that there are no negative weight cycles, some edge, call it (s, v) with weight w, is the shortest path from the source to a neighbor. By virtue of that edge being a shortest path, relaxing any other edges before or after (s, v) will not change the distance reported at the end of the first iteration, w. In other words, all shortest paths consisting of a single edge away from the source node will be explored and correctly assigned at the end of the first iteration of Bellman-Ford. Likewise, at the k^{th} iteration, Bellman-Ford will have found all paths of k edges away. Clearly, after $|V| - 1$ iterations, the shortest path from s to any other node has been discovered. The only situation that breaks this assumption is the existence of negative-weight cycles. If such a cycle exists, then after $|V| - 1$ iterations, Bellman-Ford still hasn't converged, and in turn some node's distance to s can be decreased even further.

The following lemmas and theorems establish the correctness of the Bellman-Ford algorithm. We do not cover these results in as detailed depth in recitation as depicted here, but understanding lemmas 2 and 3 go a long way towards cementing the intuition for Bellman-Ford.

Lemma 1 ***Upper-Bound Property*** *We always have $d[v] \geq \delta(s, v)$ and if we ever find $d[v] = \delta(s, v)$, $d[v]$ never changes.*

Proof. We show that $d[v] \geq \delta(s, v)$ by induction on the number of relaxation steps k.

Base Case: For $k = 0$, we have $d[s] = 0$ and $d[v] = \infty$ for all $v \in V - \{v\}$. If there is no negative cycle, then $\delta(s, s) = 0$; otherwise $\delta(s, s) = -\infty$ so $d[s] \geq \delta(s, s)$.

Induction Step: Assume after $k - 1$ iterations, $d[v] \geq \delta(s, v)$ for all $v \in V$. Now consider the kth relaxation on edge (u, v). Firstly, for $w \in V - \{v\}$, $d[w]$ does not change, so by induction $d[w] \geq \delta(s, w)$. Now, consider if we set $d[v] = d[u] + w(u, v) \geq \delta(s, u) + w(u, v) \geq \delta(s, v)$ (otherwise $\delta(s, v)$ would not be the shortest path from s to v).

If we ever achieve $d[v] = \delta(s, v)$, we cannot decrease $d[v]$ since we have just shown that $d[v] \geq \delta(s, v)$. Moreover, we cannot increase $d[v]$ because we only change $d[v]$ during relaxations and relaxations cannot increase $d[v]$. Therefore, once we have $d[v] = \delta(s, v)$ it cannot change.

Lemma 2 ***Path Relaxation***

Assume we have a graph G with no negative cycles. Let $p =< v_0, v_1, ..., v_j >$ be a shortest path from v_0 to v_j. Any sequence of edge relaxations that includes in order the relaxations of (v_0, v_1), (v_1, v_2), ..., (v_{j-1}, v_j) produces $d[v_j] = \delta(v_0, v_j)$ after all these relaxations and at all time afterwards. Note that this property holds regardless of what other relaxation calls are made before, during, or after these relaxations.

Proof. We proceed by induction on the kth vertex in p, showing that after relaxations (v_0, v_1), ..., (v_{k-1}, v_k) have occured that $d[v_k] = \delta(v_0, v_k)$.

Base Case: $k = 0$. After initialization we have that $d[v_0] = 0 = \delta(v_0, v_0)$.

Induction Step: Assume that we have relaxed, in order, edges (v_0, v_1), ..., (v_{k-2}, v_{k-1}) and $d[v_{k-1}] = \delta(v_0, v_{k-1})$. Then eventually we will make a relaxation call to (v_{k-1}, v_k) since we assume this call happens at least once after the call to (v_{k-2}, v_{k-1}). By the upper-bound property, at the time of this call $d[v_k] \geq \delta(v_0, v_k)$. In addition, we have that $\delta(v_0, v_k) = \delta(v_0, v_{k-1}) + w(v_{k-1}, v_k)$ because p is a shortest path. Therefore $d[v_k] \geq d[v_{k-1}] + w(v_{k-1}, v_k)$ and after this relaxation call we will have $d[v_k] = d[v_{k-1}] + w(v_{k-1}, v_k) = \delta(v_0, v_k)$. By the upper-bound property this is maintained ever after.

Lemma 3 ***Bellman-Ford Corectness 1:*** *Assume we have a graph G with no negative cycles reachable by s. Then after running Bellman-Ford we have, $d[v] = \delta(s, v)$ for all $v \in V$ reachable from s.*

Proof. If $v \in V$ is reachable from s then there must exist an acyclic shortest path $p =< s, v_1, ..., v_j >$ where $v_j = v$. Now, since p is acyclic, p can contain no more than V vertices and therefore no more than $|V| - 1$ edges. At every step of the Bellman-Ford algorithm we relax every edge. Therefore, we surely relax (s, v_1) on the first iteration, (v_1, v_2) on the second iteration and so on (we also relax (v_1, v_2) on the first iteration but we can not guarantee that it is relaxed after the relaxation of (s, v_1)). By the time we have reached the $|V| - 1$ iteration, we must have relaxed every edge in p in order. Therefore, by the path relaxation property, we have $d[v] = \delta(s, v)$ after $|V| - 1$ all edge relaxation steps.

Corollary 4 ***Bellman-Ford Correctness 2:*** *For vertex $v \in V$, Bellman-Ford terminates with $d[v] = \infty$ if and only if v is not reachable from s.*

Proof. Let $d[v] = \infty$ and assume v is reachable from s. Then from Bellman-Ford correctness 1 we have that $d[v] = \delta(s, v) = \infty$ which is a contradiction.

Assume v is not reachable from s. By the upper bound property, we have that $d[v] \geq \delta(s, v)$ at all times. Since $\delta(s, v) = \infty$ we must have that $d[v] = \infty$ at all times.

Theorem 5 ***Correctness of Bellman-Ford:*** *If G contains no negative cycles reachable from s, the algorithm finds no negative cycles and $d[v] = \delta(s, v)$ for all $v \in V$. If G does contain a negative-weight cycle, the algorithm will return that there is a negative cycle in the graph.*

Proof. If G contains no negative-weight cycles, by the lemmas above, $d[v] = \delta(s, v)$ for all $v \in V$ after the termination of the algorithm. Therefore $d[v] = \delta(s, v) \leq \delta(s, u) + w(u, v) \leq d[u] + w(u, v)$ by the triangle inquality for all edges (u, v). Thus no edge can still be relaxed.

Now assume G contains a negative weight cycle that is reachable from s. Let this cycle be $c =< v_0, v_1, ..., v_j >$ with $v_0 = v_j$, such that $\sum_{i=1}^{j} w(v_{i-1}, w_i) < 0$. We proceed by contradiction. Assume that the last check in Bellman-Ford does not return a negative-weight cycle. This means that for any edge (u, v) in the graph $d[u] \leq d[v] + w(u, v)$. Summing these up for all edges in the cycle we get that $\sum_{i=1}^{j} d[v_i] \leq \sum_{i=0}^{j-1} d[v_i] + w(v_i, v_{i+1})$. But $\sum_{i=1}^{j} d[v_i] = \sum_{i=0}^{j-1} d[v_i]$ since $v_0 = v_j$, so we get that $\sum_{i=0}^{j-1} w(v_i, v_{i+1} = \sum_{i=1}^{j} w(v_{i-1}, w_i) \geq 0$ which is a contradiction with the fact that the cycle has negative weight. Thus, Bellman-Ford does return that the graph has a negative-weight cycle if such a cycle exists.

2 Summary of Running Times of SSSP Algorithms

- **DAG**: For directed acyclic graphs we can consruct the topological order of the nodes in the graph. Traversing all nodes in the topological order and relaxing all outgoing edges of the current vertex finds all shortest paths in the graph. The running time of this algorithm is $O(V + E)$.

- **Non-negative weight edges**: For graphs non-negative weight edges we can use Dijkstra's algorithm to find all shortest paths from a source vertex. Running time is $O(V \log(V) + E \log(V))$, but does not work for negative edge weights.

- **Negative-weight edges**: For graphs that might have negative weight edges we use Bellman-Ford's algorithm to find all shortest paths from a source vertex. Bellman-Ford relaxes all $O(E)$ edges of a graph exactly $O(V)$ times, so the running time of the algorithm is $O(VE)$. Since the relaxation time is $O(1)$ per relaxation, the runtime of the Bellman-Ford algorithm is $O(VE)$. Bellman-Ford can also detect if a graph has a negative weight cycle reachable from the source vertex or not.

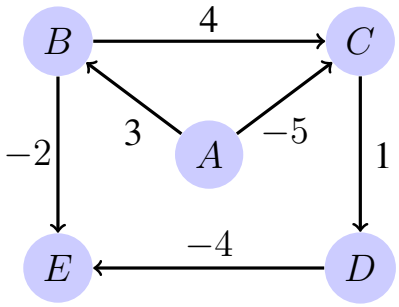
3 Example Problems Using Bellman-Ford

- Let $G = (V, E)$ be a weighted, directed graph with arbitrary (possibly negative) edge weights, but no negative cycles. Suppose we wish to find the shortest path P that starts at a source vertex s, ends at a target vertex t, and also passes through two additional detour vertices u and v. P is permitted to contain cycles, which might be necessary when visiting u and v. Provide an algorithm that does this in $O(VE)$ time.

Solution: Run Bellman-Ford 3 different times, with sources s, u and v. Now, we have all the shortest paths that we need. Compare $s_1 = \delta(s,u) + \delta(u,v) + \delta(v,t)$ and $s_2 = \delta(s,v) + \delta(v,u) + \delta(u,t)$. The smallest between s_1 and s_2 is our shortest path. This takes $O(VE)$, since we run BF three times and then we only do one simple comparison.

- In the worst-case, the Bellman Ford algorithm runs for $|V| - 1$ iterations, where $|V|$ is the number of nodes in the graph. However, for the graph G below, with A as the start node, there is an order of edge relaxations for which the Bellman Ford algorithm will have discovered all the shortest paths after a single iteration.

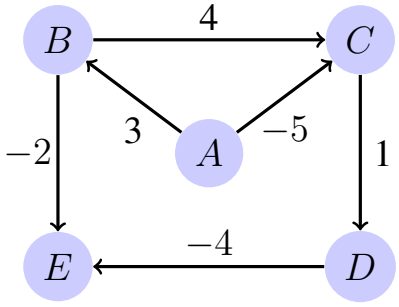
  Demonstrate such an edge ordering, and briefly explain why it works.



Solution: Relax the edges in a topologically sorted fashion.

Change the direction (but not the weight) of *a single* edge of the graph in part (a) such that for any order of edge relaxations, Bellman Ford takes more than one iteration to converge.

(The graph is reproduced below for your convenience.)



Solution: Change the orientation of the (B, E) edge, creating a negative cycle. Bellman-Ford will not converge.

Note that Bellman Ford will converge in one iteration for *some* ordering of edges for any graph if there are no negative cycles. In particular, if there are no negative weight cycles,

there is a well-defined "shortest path tree" obtained by running Bellman Ford to completion. Now, imagine re-running Bellman Ford where you relax the edges in the topologically sorted order along the shortest path tree.