1 Single Source Shortest Paths (SSSP) Recap from last time

Last lecture we defined the SSSP problem in weighted graphs:

Single Source Shortest Paths (SSSP) in weighted graphs: Given a directed or undirected $G = (V, E)$ weighted by $w : E \to \mathbb{R}$, and a source vertex $s \in V$, either detect a negative weight cycle (and throw an exception) or compute for every vertex v, the distance $d(s, v)$ and a predecessor vertex $\pi(v)$ so that $(\pi(v), v) \in E$ and $d(s, v) = d(s, \pi(v)) + w(\pi(v), v)$.

We noted that when negative cycles are present in the graph, the notion of distance we presented (minimum weight of a path) is not well-defined, so we said that SSSP algorithms wouldn't have to return the distances if a negative cycle is detected. If there are no negative cycles reachable from the source, however, we do want to return all distances and predecessor vertices so that we can build a shortest paths tree that contains all shortest paths we return.

We presented a framework for weighted shortest paths algorithms in which we maintain an estimate $d[t]$ of the distance $d(s, t)$ for each t so that at all times $d[t] \geq d(s, t)$. Initially all estimates are ∞, and for SSSP from s we also set $d[s] = 0$. This is actually $d(s, s)$ as long as there is no negative cycle through s.

During the algorithms we scan edges (u, v) and **"relax"** them. This corresponds to setting

$$d[v] = \min\{d[v], d[u] + w(u, v)\}.$$

Relaxation maintains the **invariant** that $d[v] \geq d(s, v)$.

We then proceeded to give a brainless relax algorithm that does return the correct distances if there are no negative cycles but does not detect negative cycles, and we also showed how to solve SSSP in weighted DAGs by relaxing in topological order.

Today we will see an algorithm that solves SSSP in weighted graphs: it either detects a negative cycle reachable from the source or returns the correct distances.

2 The Bellman-Ford Algorithm

The Bellman-Ford Algorithm solves the Single Source Shortest Paths (SSSP) problem i.e. given $G = (V, E)$ with weights $w : E \to \mathbb{R}$ and $s \in V$, compute $d(s, t), \forall\ t \in V$, or detect a negative cycle reachable from s, if one exists.

Recall that as usual, $|V| = n, |E| = m$.

The algorithm is simple to describe: First, initialize the distance estimates as usual: $d[s] = 0, d[v] = \infty, \forall v \neq s$. Then, repeat $n - 1$ times: relax the edges one by one in arbitrary order. At the end, if there is an edge whose relaxation would yield an improved estimate, return that a negative cycle is reachable from s. (Pseudocode is in Algorithm 1.)

To see the running time of the proposed algorithm, note that we repeatedly update our distance estimates $n - 1$ times on all m edges in time $O(mn)$, and then run through all m edges to check for negative cycles in time of $O(m)$. Thus, the total running time for the Bellman-Ford algorithm is $O(mn)$.

A priori, it probably isn't obvious why this algorithm is correct and why it detects negative cycles. However, if we knew that negative cycles will be detected correctly, then from what we proved last lecture, it would be easy to see that when there are no negative cycles, the distances returned are correct.

Claim 1. *Assume that Bellman-Ford(G, s) always detects a negative cycle reachable from s. Then, if G does not contain any negative cycles reachable from s, the distances returned by Bellman-Ford are correct.*

Proof. If the algorithm terminates without detecting a negative cycle, then at the end there is no edge whose relaxation leads to an improved distance estimate. Last time we showed that when no improvements are

Algorithm 1: Bellman Ford Algorithm with source s

```
∀v ∈ V, d[v] ← ∞ // set initial distance estimates
//optional: set π(v) ←nil for all v, π(v) represents the predecessor of v
d[s] ← 0 // set distance to start node trivially as 0
for i from 1 → n − 1 do
    for (u, v) ∈ E do
        d[v] ← min{d[v], d[u] + w(u, v)} // update estimate of v
        // optional - if d[v] changes, then π(v) ← u
// Negative Cycle Step
for (u, v) ∈ E do
    if d[v] > d[u] + w(u, v) then
        return "Negative Cycle"; // negative cycle detected
return d[v] ∀ v ∈ V
```

possible via relaxation, the distance estimates are correct, and so we get that Bellman-Ford will terminate and return the right answer as long as it detects negative cycles correctly. □

Now we will prove a few claims that will lead us to the full correctness of the Bellman-Ford algorithm.

Claim 2. *After iteration k of the Bellman-Ford algorithm, for every $v \in V$, $d[v]$ is at most the minimum weight of an s-v path that has at most k edges.*

Proof. Let $d_k[v]$ denote the value of $d[v]$ right after iteration k. We will prove by induction on k, that $d_k(v) \leq$ minimum weight of a path from s to v on $\leq k$ edges.
Base Case: $k = 0$ Here, $d_0[v] = \infty$ if $v \neq s$ which is the weight of a 0-edge path from s to v, and $d_0(s) = 0 =$ minimum weight of length 0 path from s to s. The claim is trivially satisfied in the base case.
Inductive Step: Assume that $d_{k-1}[v] \leq$ minimum weight of $s \to v$ path on $\leq k-1$ edges $\forall v$.

Let P be a shortest $s \to v$ path on $\leq k$ edges. If P contains no edges, then $v = s$, and $d_k[s] \leq d_0[s] = 0 =$ the weight of P. So let's assume P has at least one edge. Let u be node just before v on P, and let Q be the subpath of P from s to u . The path Q has $\leq k-1$ edges and is a shortest path on $\leq k-1$ edges from s to u as subpaths of shortest paths are also shortest. By the inductive hypothesis, Q has weight $\geq d_{k-1}[u]$.

Now, in iteration k, when we relax edge (u, v), we update $d[v]$ to get

$$d_k[v] \leq d[v] \leq d[u] + w(u, v) \leq d_{k-1}[u] + w(u, v) \leq w(Q) + w(u, v) = w(P).$$

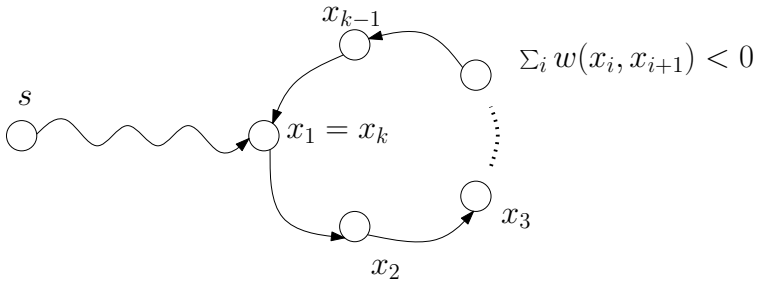
(The first and third inequalities hold since the kth iteration starts with $d_{k-1}[\cdot] = d[\cdot]$ and ends with $d_k[\cdot] = d[\cdot]$ and $d[\cdot]$ is nonincreasing. The second inequality holds from the definition of relaxation.)

This proves the induction step for iteration k. □

Now from last time we also have the following claim:

Claim 3. *If there is no negative cycle C such that s can reach C and t can be reached from C (via directed paths), then $d(s, t) > -\infty$ and there is a shortest s-t path that is simple, i.e. visits no vertex more than once and hence has at most $n-1$ edges.*

Proof. We essentially proved this last time, but let's do it again. We know that if there is no negative cycle C such that s can reach C and t can be reached from C (via directed paths), then $d(s, t) > -\infty$, and there is a shortest path from s to t on a finite number of edges. Take any such s-t path P. If P goes through some vertex x more than once, it contains a cycle C through x. We assumed that C cannot have negative weight, and hence its weight $w(C)$ is nonnegative. If we remove C from P, we get a path from s to t of weight $d(s, t) - w(C) \leq d(s, t)$, i.e. the new path is still a shortest path. We can keep removing cycles until we get a simple path. □



Because of Claims **??** and **??**, we get that after the $n-1$ relaxation stages of the Bellman-Ford algorithm, for every t that is not reachable from s via a negative cycle, $d[t] = d(s,t)$. In other words, all distances that are not $-\infty$ are computed correctly.

Now we only need to show that the algorithm detects negative cycles correctly, and that it can be modified to find the $-\infty$ distances as well.

We prove two claims that will show that negative cycles are correctly detected.

Claim 4. *If after the $n-1$ relaxation stages we have $d[v] > d[u] + w(u,v)$, then $d(s,v) = -\infty$, i.e. v can be reached from s through a negative cycle.*

Proof. We already proved that if $d(s,v) > -\infty$ (there's no negative cycle through which s can reach v), then after the $n-1$ relaxation stages, $d[v] = d(s,v)$. Since $d[u] + w(u,v)$ is the weight of some s-v path (or ∞), if $d(s,v) > -\infty$ it must be that $d[u] + w(u,v) \geq d(s,v) = d[v]$. Hence in order for $d[v] > d[u] + w(u,v)$ to hold, $d(s,v)$ must be $-\infty$. □

Claim 5. *If there is a negative cycle C reachable from s so that t can be reached from C (i.e. $d(s,t) = -\infty$), then for some $(u,v) \in E$ (in fact $(u,v) \in C$) s.t. v can reach t, after the $n-1$ relaxation stages, $d[v] > d[u] + w(u,v)$.*

This claim implies that if there is a negative cycle reachable from s, Bellman-Ford will detect it.

Proof. Let C be a negative cycle reachable from s, where C contains the nodes $x_k = x_1, x_2, \ldots, x_{k-1}, x_k$ with edges (x_i, x_{i+1}) for $i \in \{1, \ldots, k-1\}$ (see Figure **??**). Assume also that C can reach t

The sum of the weights of edges in the cycle C is negative, i.e. $\sum_{i=1}^{k-1} w(x_i, x_{i+1}) < 0$. Now, consider $\sum_{i=1}^{k-1} d[x_i]$. As x_1 is the same as x_k, we have that $\sum_{i=1}^{k-1} d[x_i] = \sum_{i=1}^{k-1} d[x_{i+1}]$.

We will now attempt to prove the Claim by contradiction. Suppose that $\forall i \in \{1, \ldots, k-1\}$, $d(x_{i+1}) \leq d(x_i) + w(x_i, x_{i+1})$. Adding over inequalities formed over all i values, $\sum_{i=1}^{k-1} d[x_{i+1}] \leq \sum_{i=1}^{k-1} d[x_i] + \sum_{i=1}^{k-1} w(x_i, x_{i+1})$. As shown above, we can replace $\sum_{i=1}^{k-1} d[x_{i+1}]$ by $\sum_{i=1}^{k-1} d[x_i]$. The inequality thus becomes $\sum_{i=1}^{k-1} d[x_i] \leq \sum_{i=1}^{k-1} d[x_i] + \sum_{i=1}^{k-1} w(x_i, x_{i+1})$. Reducing the inequality, $0 \leq \sum_{i=1}^{k-1} w(x_i, x_{i+1})$. However, the weight of the negative cycle $\sum_{i=1}^{k-1} w(x_i, x_{i+1})$ should be less than 0, which is a contradiction. □

Combining Claims **??** and **??** we get the following:

Lemma 2.1. *After the first $n-1$ relaxation stages, a vertex t has $d(s,t) = -\infty$ (equivalently it's reachable from s through a negative cycle) if and only if there is some edge (u,v) such that $d[v] > d[u] + w(u,v)$ and such that v can reach t.*

We also get that Bellman-Ford returns there is a negative cycle if and only if there is a negative cycle.

3 Finding the vertices with $-\infty$ distance from s.

Recall that the distance from s to a vertex t is $-\infty$ if and only if there is some negative cycle C so that s can reach C and the vertices of C can reach t. We showed that for all other vertices in the graph, Bellman-Ford

will compute the correct distances. We also showed in Lemma **??** that $d(s,t) = -\infty$ if and only if there is an edge (u,v) such that v can reach t and $d[v] > d[u] + w(u,v)$ after the $n-1$ relaxation stages of the algorithm.

Hence to determine the $-\infty$ distances it suffices to:

1. For every edge (u,v) for which $d[v] > d[u] + w(u,v)$ after the $n-1$ relaxation stages of the algorithm, insert v into a set L.

2. Find all vertices reachable from any vertex L.

Step 1 is easy to add to the algorithm. Step 2 is not hard to accomplish using BFS or DFS: after determining the set of vertices L, run DFS (or BFS) as if L were a single vertex. While there are some small details to figure out, this can still be accomplished in linear time.

Here is the final Bellman-Ford algorithm.

Algorithm 2: Bellman Ford Algorithm with source s

```
∀v ∈ V, d[v] ← ∞ // set initial distance estimates
//optional: set π(v) ←nil for all v, π(v) represents the predecessor of v
d[s] ← 0 // set distance to start node as 0
for i from 1 → n − 1 do
    for (u, v) ∈ E do
        d[v] ← min{d[v], d[u] + w(u, v)} // update estimate of v
        // optional - if d[v] changes, then π(v) ← u
L ← empty list
// Negative Cycle Step
for (u, v) ∈ E do
    if d[v] > d[u] + w(u, v) then
        negcycle ← true
        L.insert(v)
X ← MultiSource-DFS(L)
for v ∈ X do
    d[v] ← −∞
    // optional - set π(v) ← nil
return (negcycle, (d[v], π(v)), ∀ v ∈ V)
```