Lecture 11 Dijkstra's Algorithm
Scribes: Himanshu Bhandoh (2015), Virginia Williams, and Anthony Kim (2016), G. Valiant (2017), M. Wootters (2017) **Date:** May 15, 2017

Before moving on to the next lecture notes, here's some stuff we didn't get to on Monday May 8. In the interest of time, in lecture we will not cover all the details of the proof; but we will give the outline in lecture and the proof is in these notes.

1 Negative Edge Weights

Note that Dijkstra's algorithm solves the single source shortest paths problem when there are no edges with negative weights. While Dijkstra's algorithm may fail on certain graphs with negative edge weights, having a negative cycle (i.e., a cycle in the graph for which the sum of edge weights is negative) is a bigger problem for any shortest path algorithm. When computing a shortest path between two vertices, each additional traversal along the cycle lowers the overall cost incurred and an arbitrarily small distance can be reached after looping around the cycle multiple times. In this case, the shortest path to a node on the cycle is not well defined since it is (negatively) infinite.

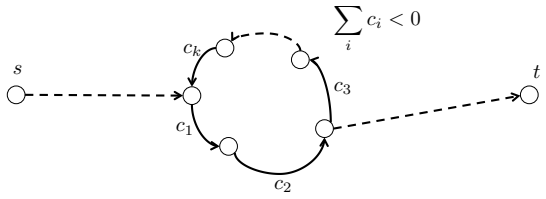

Figure 1: Assume there is a negative cycle along the $s - t$ path. The distance between s and t is not well-defined.

For example, consider the graph in Figure 1. The shortest path from s to t would start from the node s, loop around the negative cycle an infinite number of times and eventually reach destination t. The shortest path would, hence, be of infinite length and is not well-defined.

Besides the negative cycles, there are no problems in computing the shortest paths in a graph with negative edge weights. In fact, there are many applications where allowing negative edge weights is important.

2 Bellman-Ford Algorithm

In this section, we study the Bellman-Ford algorithm that solves the single source shortest paths problem on graphs with edges with potentially negative weights. Given a directed graph $G = (V, E)$ with edge weights given by $c(x, y)$ for $(x, y) \in E$, we want to compute the shortest path distances $d(s, v)$ from source s for all $v \in V$. More specifically, the Bellman-Ford algorithm:

- Detects a negative cycle if it exists and is reachable from s, or
- Computes the shortest path distances $d(s, v)$ for all $v \in V$.

Note $\pi(\cdot)$ is used to store the shortest paths found and $\pi(v)$ represents the predecessor of v on the shortest path from s to v.

For an example run of the Bellman-Ford algorithm, please refer to the lecture slides or CLRS.

Algorithm 1: Bellman-Ford Algorithm

$d[v] \leftarrow \infty, \forall v \in V$ // set initial distance estimates
// to maintain paths: set $\pi(v) \leftarrow$nil for all v, $\pi(v)$ represents the predecessor of v
$d[s] \leftarrow 0$ // set distance to start node trivially as 0
for i *from* $1 \rightarrow n-1$ **do**
  for $(u, v) \in E$ **do**
    $d[v] \leftarrow \min\{d[v], d[u] + w(u, v)\}$ // update the distance estimate for v
    // to maintain paths - if $d[v]$ changes, then $\pi(v) \leftarrow u$
// Negative Cycle Step
for $(u, v) \in E$ **do**
  if $d[v] > d[u] + w(u, v)$ **then**
    return "Negative Cycle"; // negative cycle detected
return $d[v]$ $\forall$ $v \in V$

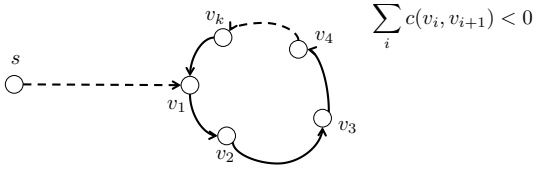

Figure 2: A negative cycle reachable from source s

The total runtime of the Bellman-Ford algorithm is $O(mn)$. In the first for loop, we repeatedly update the distance estimates $n-1$ times on all m edges in time $O(mn)$. In the second for loop, we go through all m edges to check for negative cycles in time of $O(m)$.

We prove the correctness of the Bellman-Ford algorithm in two steps:

Claim 1. *If there is a negative cycle reachable from s, then the Bellman-Ford algorithm detects and reports "Negative Cycles".*

Proof. For the sake of contradiction, suppose there exists a negative cycle C reachable from the source s and the Bellman-Ford algorithm does not report "Negative Cycles". Assume C contains nodes $v_1, v_2, \ldots, v_k$ with edges (v_i, v_{i+1}) for $i = 1, \ldots, k$ such that $\sum_{i=1}^{k} c(v_i, v_{i+1}) < 0$, where $v_{k+1} = v_1$. See Figure 2. Let $d[\cdot]$ be the distance estimates determined in the first for loop of the algorithm.

Since C is reachable from s, there is a path from s to v_1 and to all nodes on C. In particular, there exist simple paths, i.e., paths without cycles, of at most $n-1$ edges to the nodes of C. In the first for loop, the edges on each such simply path get relaxed in order and consequently, $d[v_i]$ will be some finite number less than ∞ for $i = 1, \ldots, k$. Since the Bellman-Ford algorithm does not report "Negative Cycles" in the second for loop, it must be that $d[v_{i+1}] \leq d[v_i] + c(v_i, v_{i+1})$ for $i = 1, \ldots, k$. Adding the inequalities, we obtain

$$\sum_{i=1}^{k} d[v_{i+1}] \leq \sum_{i=1}^{k} d[v_i] + \sum_{i=1}^{k} c(v_i, v_{i+1}) \ .$$

As we are summing over the cycle C, the terms $\sum_{i=1}^{k} d[v_{i+1}]$ and $\sum_{i=1}^{k} d[v_i]$ are equal and can be cancelled. It follows that $0 \leq \sum_{i=1}^{k} c(v_i, v_{i+1})$. This contradicts that C is a negative cycle.

□

In the next claim, we show that if the graph has no negative cycles reachable from the source, then the Bellman-Ford algorithm returns the correct shortest path distances.

Claim 2. *If G has no negative cycles reachable from s, then $d[v] = d(s, v), \forall v \in V$.*

Proof. Let $d_k(v)$ be the value of $d[v]$ after k iterations of the first for loop. We prove by induction the statement that $d_k(v)$ is equal to the minimum distance of a path from s to v with at most k edges. Then, we will have $d_{n-1}(v) = d[v]$ for all node v at termination. Since we can assume that shortest paths have at most $n - 1$ edges without loss of generality, the claim follows.

We argue that if there is a path from s to v, then there exists a shortest path from s to v has at most $n - 1$ edges. If a shortest path has a cycle, the cycle cannot be negative and we can remove it and improve its total distance. If the cycle has a positive weight, removing the cycle will strictly improve the shortest path's distance. If the cycle has zero weight, we can ignore the cycle. Hence, we can assume that shortest paths are simple, that is, do not have cycles.

Base Case: When $k = 0$, the distance estimates have been just initialized. So, $d_0(v) = \infty$ if $v \neq s$. Furthermore, $d_0(s) = 0 = d(s, s)$, which is the minimum distance of length-0 paths from s to s. The statement is satisfied for the base case.

Inductive Step: Assume that $d_{k-1}(v)$ is equal to the minimum distance of a $s \to v$ path on at most $k - 1$ edges for all v.

Consider $v \neq s$. Let P be a shortest simple $s \to v$ path on at most k edges. Let u be the node just before v on P, and let Q be the sub-path of P from s to u. The path Q would have at most $k - 1$ edges and is a shortest path from s to u with at most $k-1$ edges, since sub-paths of shortest paths are also shortest paths. By the inductive hypothesis, Q has distance $d_{k-1}(u)$.

In the k-th iteration, we update $d_k(v)$ such that $d_k(v) \leq d_{k-1}(u) + w(u, v) = w(Q) + w(u, v) = w(P)$. Since whenever $d_k(v)$ is finite, it actually corresponds to the distance of some path from s to v on at most k edges, in particular, it has to be at least as large as the distance of the shortest path from s to v on at most k edges. Thus, $d_k(v) \geq w(P)$. After the k-th iteration, $d_k(v) = w(P)$, and the inductive step follows.

The induction is complete, and the claim is proved. □