

Shortest Paths

Lecturer: Daniel A. Spielman

January 28, 2010

6.1 Why

I am writing these notes to explain my perspective on Dijkstra's algorithm.

6.2 Dijkstra's algorithm

The inputs to Dijkstra's algorithm are a directed graph with lengths on the edges, $G = (V, E, l)$, and a "start" vertex s . The goal of the algorithm is twofold: to determine the length of the shortest path from s to every other vertex, and to return data structure from which one can easily compute this path. Our algorithm will return the lengths in an array that I will call δ , and it will return the data structure in the form an array of pointers, `prev`. If you want the shortest path from s to v , you first look at `prev(v)`, and this will give you the node on the path before v . Of course, you then look at `prev(prev(v))` to figure out how to get there, and so on.

Let $d(v)$ denote the length of the shortest path from s to v . The algorithm will maintain estimates of $d(v)$ that we will call $\delta(v)$. This estimate is the length of the shortest path from s to v using only the edges that the algorithm has encountered so far, so it will always be the case that $\delta(v) \geq d(v)$. The algorithm will also establish a set of vertices $W \subseteq V$ for which $\delta(v) = d(v)$, and will grow this set at each iteration. The set W will always contain the set of vertices closest to s .

Here is the algorithm.

0. Set $\delta(s) = 0$, $W = \emptyset$, and $\delta(u) = \infty$ and `prev(u) = \emptyset` for all $u \neq s$.
1. While there exists a $u \notin W$ for which $\delta(u) < \infty$.
 - a. Let $u \notin W$ minimize $\delta(u)$.
 - b. Set $W = W \cup \{u\}$.
 - c. For $(u, v) \in E$, $v \notin W$,
if $\delta(u) + l(u, v) < \delta(v)$,
set $\delta(v) = \delta(u) + l(u, v)$ and `prev(v) = u` .

When the algorithm finishes, any vertex u for which $\delta(u) = \infty$ is unreachable from s .

6.3 Analysis

Lemma 1. *Throughout the course of the algorithm $\delta(v) \geq d(v)$ for all v .*

Proof. We prove this by induction on time. Initially, we have $\delta(s) = d(s) = 0$, and $\delta(v) = \infty \geq d(v)$ for all $v \neq s$, so it is true initially. We just need to verify that it remains true whenever $\delta(v)$ is updated. When $\delta(v)$ is updated, it is set to $\delta(u) + l(u, v)$. Now, there is a path from s to v of length $d(u) + l(u, v)$, so

$$d(v) \leq d(u) + l(u, v) \leq \delta(u) + l(u, v) = \delta(v),$$

where the second inequality follows from the induction hypothesis. So, $d(v) \leq \delta(v)$ throughout the algorithm. \square

Lemma 2. *When a vertex u is added to W , we have $d(u) = \delta(u)$ and*

$$d(u) = d(\text{prev}(u)) + l(\text{prev}(u), u).$$

Proof. Again, we will prove this by induction on time. During the first execution of the while loop, we will have $u = s$, and set $W = \{s\}$. Thus, the statement will be true at this time, as $\delta(s) = d(s) = 0$. We now prove that it remains true when we add an element u to W , assuming that for all $w \in W$, $d(w) = \delta(w)$. (you should verify that $\delta(w)$ is never changed after w is added to W .)

Forget about u for a moment, and define

$$S = \arg \min \{d(x) : x \notin W\}.$$

That is, let S be the set of vertices x not in W whose value of $d(x)$ is minimum. I claim that for all $x \in S$, $d(x) = \delta(x)$. To prove this let x be any element of S , and let y be the previous vertex on a shortest path from s to x , so

$$d(x) = d(y) + l(y, x).$$

Now, $l(y, x) > 0$, so $d(y) < d(x)$, which implies $y \in W$. Let y be the first such vertex that was added to W . By our inductive hypothesis, when we added y to W we had $\delta(y) = d(y)$. By the assumption that y is the first vertex for which $d(x) = d(y) + l(y, x)$ that we added to W , and Lemma 1, we know that when we added y to W we set

$$\delta(x) = \delta(y) + l(y, x) = d(y) + l(y, x) = d(x),$$

and

$$\text{prev}(x) = y.$$

Now, to finish the proof, we just need to show that $u \in S$. To show this, let $x \in S$ and observe

$$\begin{aligned} \delta(u) &\leq \delta(x), && \text{by the choice of } u \text{ in the algorithm} \\ &= d(x), && \text{by the claim we just proved} \\ &\leq d(u), && \text{by the definition of } S \\ &\leq \delta(u), && \text{by Lemma 1.} \end{aligned}$$

So $d(u) = d(x)$, and $u \in S$. \square

6.4 Implementation

Of course, we implement this algorithm using a priority queue, just as we did in Prim's algorithm. A simple analysis shows that the algorithm will add elements to the priority queue n times, change keys at most m times, and extract min at most n times (where n is the number of vertices and m is the number of edges). So, the running time of the algorithm is bounded by $O((n + m) \log n)$.

6.5 Shortest Path Tree

We now observe that there is a tree in the graph so that the shortest path from s to each u is the unique path in that tree from s to u . In fact, this tree is the union of the edges $(\text{prev}(u), u)$ for $u \neq s$.