Design and Analysis of Algorithms Lecture 5

Shortest Paths

Lecturer: Daniel A. Spielman January 31, 2017

5.1 Why

I am writing these notes to explain my proof of the correctness of Dijkstra's algorithm, as my proof is slightly different from the one in the book. These notes are intended to supplement the book, not replace it.

5.2 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. We represent a directed from u to v edge by a pair (u, v). The length of this edge will be written $l(u, v)$. In today's lecture, all lengths are positive.

The goal of the algorithm is twofold: to determine the length of the shortest path from s to every other vertex, and to return a 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, `pred`, which stands for predecessor. A predecessor of a node v is a node right before v on a shortest path from s to v. If u is a predecessor of v, then

$$d(v) = d(u) + l(u, v).$$

The node $\texttt{pred}(v)$ computed by the algorithm will be a predecessor of v. If you want the shortest path from s to v, you first look at $\texttt{pred}(v)$, and this will give you the node on the path before v. Of course, you then look at $\texttt{pred}(\texttt{pred}(v))$ 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 and $\texttt{pred}(u) = \emptyset$ for all $u \neq s$.
1. While there exists a $u \notin W$ for which $\delta(u) < \infty$. (In the first iteration, $\delta(s) < \infty$ and $s \notin W$)
    a. Let $u \notin W$ minimize $\delta(u)$.

b. 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 $\texttt{pred}(v) = u$.

c. Set $W = Wu$.

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

This algorithm is greedy in the sense that W is always a set of vertices that is closest to s. This fact might not be obvious, so we will prove it later. If all of the distances from s were distinct, we could say this by re-numbering the vertices by order of distance from s. So, vertex 1 would be s, and so on. Then, when $W = k$, we would have $W = 1, \ldots, k$. But, this approach does not work if there are ties among the distances. So, we instead will show that for all $x \in W$ and all $y \notin W$, $d(x) \leq d(y)$.

5.3 Analysis

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. □

The idea behind the algorithm is that we always add the predecessor of v before we add v. And, when we add the predecessor of v, we set $\delta(v) = d(v)$. While this fact alone does not give a proof of correctness of the algorithm, it is a good first step. We prove it now.

If u is a predecessor of v, $u \in W$ and $\delta(u) = d(u)$, then $\delta(v) = d(v)$.

Proof. As u is a predecessor of v, $d(v) = d(u) + l(u, v)$. Before u is added to W, if $\delta(v)$ is more than $\delta(u) + l(u, v)$, it is set to $\delta(u) + l(u, v)$. As we have assumed that $d(u) = \delta(u)$, we now know that $\delta(v) \leq d(v)$. By Lemma 5.3, we can conclude that $\delta(v) = d(v)$. □

The reason that the above argument does not directly prove the correctness of the algorithm is that we have not proved that a predecessor of v is added to W before v. We will do that now.

At the end of every iteration of the while loop,

a. $\delta(x) = d(x)$ for all $x \in W$, and

b. for all $x \in W$ and $y \notin W$, $d(x) \leq d(y)$.

Part b of the Lemma tells us that W does in fact contain the vertices closest to s, and that is in fact adds vertices to W in order of their distance from s.

Proof. We will prove this by induction on the number of times the loop has been executed. That is, by induction on time.

In the first iteration through the loop, The vertex u selected in step 1a will be s, W will be set to s, and $\delta(s) = d(s) = 0$. So, the inductive hypothesis holds after the first execution of the loop.

For the induction, assume that the inductive hypothesis is satisfied. We will show that the vertex u selected at step 1a minimizes $d(u)$ among those vertices not in W. To this end, we first reason about such a vertex[1].

Let y be a vertex not in W that minimizes $d(y)$. I say "a vertex" because there could be more than one. We need to show that y or a vertex at the same distance from s is added to W. Let x be a predecessor of y. As $d(x) < d(y)$, and y minimizes d over vertices not in W, $x \in W$. By the induction hypothesis, we know that $\delta(x) = d(x)$. Lemma 5.3 now allows us to conclude that $\delta(y) = d(y)$.

We now show that the vertex u selected in step 1a satisfies $d(u) = d(y)$. We know that $\delta(u) \leq \delta(y) = d(y)$. Combining this with Lemma 5.3, which says that $d(u) \leq \delta(u)$, we may conclude $d(u) \leq d(y)$. As y minimizes $d(y)$ among vertices not in W, and u is not in W, $d(y) \leq d(u)$, which implies that $d(u) = d(y)$ and thus u also minimizes $d(u)$ among vertices not in W. Let $\widehat{W}$ be the set W at the end of the iteration and W be the set at the beginning. As $\widehat{W} = Wu$, we have proved part b for this iteration. The above inequalities also imply that $d(u) = \delta(u)$, which establishes part a for this iteration. □

At the end of the algorithm, $\delta(u) = d(u)$ for every vertex u.

Proof. We know that $\delta(u) = d(u)$ for every vertex that is added to W. It remains to show that $d(u) = \infty$ for those that are not added to W. That is, these are the vertices that are not reachable from s.

I claim that when the algorithm terminates, there are no edges (u, v) with $u \in W$ and $v \notin W$. If there were such an edge, then $\delta(v)$ would be less than ∞. However, then the algorithm would not terminate.

This means that there is no path from s to v for every vertex v such that $\delta(v) = \infty$, which implies that $d(v) = \infty$ for all such vertices. To make that more clear, imagine what such a path would look like. As it starts inside W and ends outside W, it would have to contain some edge (x, y) where $x \in W$ and $y \notin W$. But, we just argued that such an edge can not exist. This finishes the proof, as we have shown that $d(v) = \infty$ for all $v \notin W$.

[1] One might first be inclined to prove a statement about u, the vertex chosen in step 1a. Note that this is not what we are doing, and it would lead to an awkward proof. Instead, we reason about a vertex y that has the properties we wish to show are possessed by u.

□

5.4 Implementation

Of course, we implement this algorithm using a priority queue. 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 $O((n+m)\log n)$.

5.5 Shortest Path Tree

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