

Analysis of Algorithms I: Single-Source Shortest Paths

Xi Chen

Columbia University

We start with some notation. Let $G = (V, E)$ denote a weighted directed graph. The weight of $(u, v) \in E$ is $w(u, v)$. The weight of a path $p = \langle v_0, v_1, \dots, v_k \rangle$ is the sum of the weights of its edges:

$$w(p) = \sum_{i=1}^k w(v_{i-1}, v_i)$$

Given $u, v \in V$, we define the shortest-path weight from u to v :

$$\delta(u, v) = \min \{ w(p) : \text{any path from } u \text{ to } v \}$$

and $\delta(u, v) = +\infty$ if v is not reachable from u . Usually we simply refer to $\delta(u, v)$ as the distance from u to v . In this class, we focus on the single-source shortest-paths problem: Given a weighted directed graph $G = (V, E)$ and a source vertex $s \in V$, compute $\delta(s, v)$ and find a shortest path from s to v , for all $v \in V$. In the next class, we discuss the all-pairs shortest-paths problems. While the latter can be solved by running a single-source algorithm once for each vertex, usually it can be solved faster.

Some basic properties of $\delta(s, v)$:

- 1 Triangle inequality:

$$\delta(u, v) \leq \delta(u, y) + \delta(y, v), \quad \text{for all } u, y, v \in V$$

Implies $\delta(s, v) \leq \delta(s, u) + w(u, v)$ for any $(u, v) \in E$

- 2 * Subpath property *: If $p = \langle v_0, v_1, \dots, v_k \rangle$ is a shortest path from v_0 to v_k , then for any $i, j : 0 \leq i \leq j \leq k$,

$$p_{i,j} = \langle v_i, v_{i+1}, \dots, v_j \rangle$$

must be a shortest path from v_i to v_j .

We start by discussing the case when all weights are nonnegative (e.g., distances between cities). Dijkstra's algorithm: Very very similar to Prim's algorithm for minimum spanning trees. Let $G = (V, E)$ be a weighted directed graph. Note: If G is undirected, just replace each undirected edge by two directed edges with opposite directions with the same weight. For convenience, we also assume that all vertices are reachable from s , though this assumption is not necessary.

Dijkstra's algorithm maintains a set of vertices S , with $S = \{s\}$ at the beginning. For each round, we pick a vertex from $V - S$ and add it to S . When a vertex v is picked and added into S , the distance $\delta(s, v)$ is computed correctly and stored in $v.d$. Since we assumed that all vertices are reachable from s , the algorithm stops when $S = V$. In addition to $v.d$, each vertex $v \in V$ also has an attribute $v.\pi$, a pointer to another vertex in the graph. Edges from

$$E_\pi = \{(v.\pi, v) : v \in V - \{s\}\} \subseteq E$$

form a shortest-paths tree: For every $v \in V - \{s\}$, the unique path from s to v in E_π must be a shortest path from s to v . In the class, we only focus on the $v.d$ attribute.

Before describing the algorithm, we present the key lemma to Dijkstra's algorithm. Let S be a set of vertices with $s \in S$. We say p is an S -path from s to $v \in V - S$ if all vertices of p lie in S except v itself (so all the edges on the path p have both endpoints in S except the last edge (u, v) , with $u \in S$ and $v \in V - S$.) Quick question: If we know the distance $\delta(s, u)$ for all $u \in S$, how can we compute the weight of the shortest S -path from s to $v \in V - S$? We denote the latter by $\delta(s, S, v)$. Use the following formula:

$$\delta(s, S, v) = \min_{u \in S} \{ \delta(s, u) + w(u, v) \} \quad (1)$$

Prove its correctness. Here comes the lemma:

Lemma

Let S be a set of vertices with $s \in S$. If $v \in V - S$ has the minimum $\delta(s, S, v)$ among all vertices $v \in V - S$, then we must have $\delta(s, v) = \delta(s, S, v)$.

Assume this is not the case, then we must have $\delta(s, v) < \delta(s, S, v)$ because $\delta(s, v) \leq \delta(s, S, v)$ by definition. This means there is a shortest path p from s to v such that

$$w(p) < \delta(s, S, v)$$

Let y denote the first vertex not in S on the path p . If $y = v$ then p is indeed an S -path and thus,

$$w(p) \geq \delta(s, S, v)$$

contradiction. So $y \neq v$ is a predecessor of v in p . Let p' denote the subpath of p from s to y , then p' is clearly an S -path (why?). As a result, we have

$$\delta(s, S, y) \leq w(p') \leq w(p) < \delta(s, S, v)$$

contradicting with the assumption that v has the minimum $\delta(s, S, v)$ among all vertices in $V - S$ (since $y \in V - S$).

This suggests the following naive but correct algorithm: Start with $S = \{s\}$ and $s.d = 0$. At any time every $v \in S$ has $v.d = \delta(s, v)$. For each round (when $S \neq V$ yet), use formula (1) to compute $\delta(s, S, v)$ for each $v \in V$, which takes time $|V - S| \cdot |S|$. Find a vertex $v \in V$ that has the minimum $\delta(s, S, v)$. Set

$$v.d = \delta(s, S, v)$$

and add it into S . But ... too slow!

Instead, we keep the following invariant: Prior to each round

- 1 For every $u \in S$, $u.d = \delta(s, u)$. For every $v \in S$,

$$v.d = \delta(s, S, v)$$

which is set to be $+\infty$ if currently there is no S -path from s to v (may happen even if all vertices are reachable from s)

- 2 We also maintain a priority queue Q of vertices in $V - S$, sorted based on the $v.d$ attribute. So to find a vertex $v \in V$ with the minimum $\delta(s, S, v)$, it suffices to make a call to Extract-Min. However (similar to Prim's algorithm), after adding v to S (note that there is no need to change $v.d$, why?) we need to update $w.d$ for every w remains in Q .

Now we present Dijkstra's algorithm:

- 1 set $S = \{s\}$, $s.d = 0$ and $s.\pi = \text{nil}$ (root)
- 2 for each $v \in V - \{s\}$ (check that the invariant holds)
- 3 if $(s, v) \in E$: set $v.d = w(s, v)$ and $v.\pi = s$
- 4 else: set $v.d = +\infty$ and $v.\pi = \text{nil}$
- 5 Priority-Queue-Init($Q, V - \{s\}$)
- 6 while $Q \neq \emptyset$ ($S \neq V$) do
- 7 $u = \text{Extract-Min}(Q)$
- 8 for each $v \in \text{adj}[u]$ do
- 9 if $v \in Q$ and $v.d > u.d + w(u, v)$ then
- 10 Decrease-Key($Q, v, u.d + w(u, v)$) and $v.\pi = u$

To prove its correctness, it suffices to show that after adding u to S at the beginning a while-loop, by the end of the loop we still have

$$v.d = \delta(s, S, v)$$

for every vertex v in Q . Running time of Dijkstra: Initialization of Q plus $n - 1$ Extract-Min plus m Decrease-Key. If we use Heap (or Red-Black tree) to implement Q : $O(m \lg n)$. By using a Fibonacci heap (Chapter 19), the total running time is $O(m + n \lg n)$.

Now we work on the more general case when the weights can be negative. Again, we assume that all vertices $v \in V$ are reachable from s . The trouble of having negative weights is that sometimes $\delta(s, v)$ is not well defined. How can this happen? It happens when there is a cycle c in G such that the total weight $w(c)$ of edges in c is negative. For example, if $(s, a), (a, b), (b, c), (c, a), (a, d) \in E$ and the weight of the cycle $abca$ is negative, then we can go from s to d by cycling around $abca$ for as many times as we want so that the total weight of the path approaches $-\infty$. So no matter what path from s to d you pick, I can always find you in this (kind of stupid) way a path with even smaller total weight. Show that if there is no negative-weight cycle in G , then $\delta(s, v)$ is well-defined and there always exists a shortest “simple” path from s to v .

The Bellman-Ford algorithm solves the single-source shortest-paths problem when the weights may be negative. (See the details below.) The input is a weighted directed graph $G = (V, E)$ in which the weights may be negative, as well as a source vertex $s \in V$. Output: Either indicate that G has a negative-weight cycle; or if no negative-weight cycle exists in G (for which case $\delta(s, v)$ is well-defined for all $v \in V$), compute $\delta(s, v)$ and a shortest path from s to v for all $v \in V$. For the latter, again we mean that E_π forms a shortest-paths tree.

- 1 set $s.d = 0$ and $s.\pi = \text{nil}$ (root)
- 2 for each $v \in V - \{s\}$ do
- 3 set $v.d = \infty$ and $v.\pi = \text{nil}$
- 4 repeat $n - 1$ times
- 5 for each edge $(u, v) \in E$ do
- 6 if $v.d > u.d + w(u, v)$ then
- 7 set $v.d = u.d + w(u, v)$ and $v.\pi = u$
- 8 for each edge $(u, v) \in E$ do
- 9 if $v.d > u.d + w(u, v)$ then
- 10 return “negative cycle”
- 11 return “no reachable negative cycle”

The running time of Bellman-Ford is $\Theta(nm)$. Now we prove its correctness. First of all, if there is a negative-weight cycle, say

$$c = \langle v_0, v_1, \dots, v_k, v_0 \rangle$$

in G , then the algorithm must return “negative cycle”. To see this, assume for contradiction that line 10 is not executed.

Because $(v_0, v_1), (v_1, v_2), \dots, (v_k, v_0) \in E$, we have

$$v_1.d \leq v_0.d + w(v_0, v_1)$$

$$v_2.d \leq v_1.d + w(v_1, v_2)$$

...

$$v_0.d \leq v_k.d + w(v_k, v_0)$$

Summing up all these $k + 1$ inequalities gives us

$$0 \leq w(v_0, v_1) + w(v_1, v_2) + \dots + w(v_k, v_0)$$

contradicting with our assumption of c being a negative cycle.

Finally we show that if there is no negative-weight cycle in G , then $v.d = \delta(s, v)$ for all v before the first for-loop of line 8; and the algorithm outputs “no reachable negative cycle” by the end. We prove the second part first. If $v.d = \delta(s, v)$ for all $v \in V$, then for any $(u, v) \in E$, we have the following simple inequality

$$\delta(s, v) \leq \delta(s, u) + w(u, v)$$

(why?) and thus,

$$v.d \leq u.d + w(u, v)$$

for all $(u, v) \in E$. So it outputs “no reachable negative cycle”.

We prove $v.d = \delta(s, v)$ for all $v \in V$. First, it is easy to prove, using induction, that $v.d \geq \delta(s, v)$ during any time of the algorithm. Also $v.d$ is nonincreasing during the execution of Bellman-Ford because we only change $v.d$ on line 7, which only makes it smaller. These two properties imply that if $v.d$ is set to be $\delta(s, v)$ at some time during the execution, then it remains to be $\delta(s, v)$ ever after! Now we start the proof.

Pick any vertex $v \in V$. We show that $v.d = \delta(s, v)$ by the end of the $(n - 1)$ iterations of line 4. If there is no negative-weight cycle, then $\delta(s, v)$ is well-defined and there is a “simple” path p from s to v with $w(p) = \delta(s, v)$. Because

$$p = \langle v_0, v_1, \dots, v_{k-1}, v_k \rangle, \quad \text{where } s = v_0 \text{ and } v = v_k$$

is simple, we have $k \leq n - 1$. It suffices to prove by induction:

By the end of the i th iteration of line 4, $v_i.d = \delta(s, v_i)$.

Because it implies that by the end of the $(k \leq n - 1)$ th iteration, we have $v.d = \delta(s, v)$ and it remains so ever after.

The basis is trivial. Induction step: Assume that by the end of the $(i - 1)$ th iteration (or at the beginning of the i th iteration), for some $i \leq k$, $v_{i-1}.d = \delta(s, v_{i-1})$ and remains so ever after. We show that by the end of the i th iteration, it must be the case that $v_i.d = \delta(s, v_i)$. This is because in the for-loop of line 5, after the edge $(v_{i-1}, v_i) \in E$ is processed, we must have

$$v_i.d \leq v_{i-1}.d + w(v_{i-1}, v_i) = \delta(s, v_{i-1}) + w(v_{i-1}, v_i) = \delta(s, v_i)$$

The second equation uses $v_{i-1}.d = \delta(s, v_{i-1})$ by the inductive hypothesis. The last equation uses the * subpath property *.

Read Section 24.2: How to solve the single-source shortest paths problem efficiently when G is a DAG:

Topological sort + dynamic programming