Shortest paths in edge-weighted graphs. Given an edge-weighted directed graph $G = (V, E)$, the *shortest-path distance* from vertex v to vertex w is the minimum weight of any $v \rightsquigarrow w$ path:

$$\mathsf{dist}(v, w) = \min\{\mathsf{wt}(P) : P \text{ is a path from } v \text{ to } w \text{ in } G\}.$$

We think of a path P as a set of edges, and use $\mathsf{wt}(P)$ to denote the sum of the edges' weights, so $\mathsf{wt}(P) = \sum_{e \in P} \mathsf{wt}(e)$. If there is no $v \rightsquigarrow w$ path, then $\mathsf{dist}(v, w) = \infty$.

Consider some examples to get intuition for shortest paths.

Observation 1. *Let $G = (V, E)$ be any edge-weighted directed graph. Suppose the weight of each cycle C in G is non-negative (that is $\sum_{e \in C} \mathsf{wt}(e) \geq 0$). Then, for every two vertices $v, w \in V$, the shortest-path distance from v to w is well-defined. Also, if there is any $v \rightsquigarrow w$ path, then there is a shortest $v \rightsquigarrow w$ path that is* simple *(has no cycles).*

Lemma 1. *Let $G = (V, E)$ be any edge-weighted directed graph. Let $s \in V$ be any vertex. Assume the weight of each cycle in G is non-negative. Then $\mathsf{dist}(s, s) = 0$, and, for every vertex $w \in V \setminus \{s\}$,*

$$\mathsf{dist}(s, w) = \min\{\mathsf{dist}(s, u) + \mathsf{wt}(u, w) : (u, w) \in E\}.$$

Proof (long form).

1. The only simple path from s to s is the empty path of weight 0, so, by Obs. 1, $\mathsf{dist}(s, s) = 0$.
2.1. Consider any other vertex $w \in V \setminus \{s\}$.
2.2. First we show $\mathsf{dist}(s, w) \leq \min\{\mathsf{dist}(s, u) + \mathsf{wt}(u, w) : (u, w) \in E\}$.
2.3.1. Consider any edge $(u, w) \in E$.
2.3.2.1. <u>Case 1.</u> First consider the case that $\mathsf{dist}(s, u) < \infty$.
2.3.2.2. There is a path P_u from s to u of weight $\mathsf{dist}(s, u)$.
2.3.2.3. Path P_u and edge (u, w) form a path P_w from s to w of weight $\mathsf{dist}(s, u) + \mathsf{wt}(u, w)$.
2.3.2.4. So $\mathsf{dist}(s, w) \leq \mathsf{wt}(P_w) = \mathsf{dist}(s, u) + \mathsf{wt}(u, w)$.
2.3.3.1. <u>Case 2.</u> Otherwise $\mathsf{dist}(s, u) = \infty$.
2.3.3.2. Then $\mathsf{dist}(s, w) \leq \infty = \mathsf{dist}(s, u) + \mathsf{wt}(u, w)$.
2.3.4. By Blocks 2.3.2 and 2.3.3, $\mathsf{dist}(s, w) \leq \mathsf{dist}(s, u) + \mathsf{wt}(u, w)$.
2.4. By Block 2.3, $\mathsf{dist}(s, w) \leq \min\{\mathsf{dist}(s, u) + \mathsf{wt}(u, w) : (u, w) \in E\}$.
2.5. To finish we show $\mathsf{dist}(s, w) \geq \min\{\mathsf{dist}(s, u) + \mathsf{wt}(u, w) : (u, w) \in E\}$.
2.6.1. <u>Case 1.</u> First consider the case that $\mathsf{dist}(s, w) < \infty$.
2.6.2. Let P_w be any path from s to w of weight $\mathsf{dist}(s, w)$.
2.6.3. Let (u', w) be the last edge on P_w.
2.6.4. Let $P_{u'} = P_w \setminus \{(u', w)\}$ be the prefix of P_w from s to u'. Then

$$\begin{aligned}
\mathsf{dist}(s, w) &= \mathsf{wt}(P_w) && \textit{(by the definition of } P_w\textit{)} \\
&= \mathsf{wt}(P_{u'}) + \mathsf{wt}(u', w) && \textit{(as } P_w = P_{u'} \cup \{(u', w)\}\textit{)} \\
&\geq \mathsf{dist}(s, u') + \mathsf{wt}(u', w) && \textit{(as } P_{u'} \textit{ is an } s \rightsquigarrow u' \textit{ path)} \\
&\geq \min\{\mathsf{dist}(s, u) + \mathsf{wt}(u, w) : (u, w) \in E\} && \textit{(because } (u', w) \in E\textit{).}
\end{aligned}$$

2.7.1. <u>Case 2.</u> Otherwise $\mathsf{dist}(s, w) = \infty$.
2.7.2. Then $\mathsf{dist}(s, w) = \infty \geq \min\{\mathsf{dist}(s, u) + \mathsf{wt}(u, w) : (u, w) \in E\}$.
2.8. By Blocks 2.6 and 2.7, $\mathsf{dist}(s, w) \geq \min\{\mathsf{dist}(s, u) + \mathsf{wt}(u, w) : (u, w) \in E\}$.
2.9. By this and Step 2.4, $\mathsf{dist}(s, w) = \min\{\mathsf{dist}(s, u) + \mathsf{wt}(u, w) : (u, w) \in E\}$.
3. By Block 2, $\forall w \in V \setminus \{s\}$, $\mathsf{dist}(s, w) = \min\{\mathsf{dist}(s, u) + \mathsf{wt}(u, w) : (u, w) \in E\}$. □

Single-Source Shortest Paths

input: Directed graph $G = (V, E)$ with non-negative edge weights and source vertex $s \in V$.

output: For each vertex $v \in V$, the shortest-path distance from s to v.

Greedy design pattern: the first step. We start by finding a rule for easily determining just *part* of a correct solution. By Lemma 1, $\mathsf{dist}(s, s) = 0$. Now what about out-neighbors of s? Is there always an out-neighbor w of s whose shortest-path distance $\mathsf{dist}(s, w)$ we can easily determine? Consider examples to get intuition.

Lemma 2. *Among edges leaving s, let (s, w') be one of minimum weight. Then the edge (s, w') is a shortest path from s to w'. That is, $\mathsf{dist}(s, w') = \mathsf{wt}(s, w')$.*

Proof (short form). We first show $\mathsf{dist}(s, w') \le \mathsf{wt}(s, w')$, then show $\mathsf{dist}(s, w') \ge \mathsf{wt}(s, w')$.

Part (i). The edge (s, w') is one possible path from s to w'. So by definition, $\mathsf{dist}(s, w') \le \mathsf{wt}(s, w')$.

Part (ii). To show $\mathsf{dist}(s, w') \ge \mathsf{wt}(s, w')$, we show that every path from s to w' must have weight at least $\mathsf{wt}(s, w')$. The idea is that any path from s to w' must leave s, and leaving s (on any edge) costs at least $\mathsf{wt}(s, w')$. And all remaining edges on the path have non-negative weight.

Consider an arbitrary path $P_{w'}$ from s to w'. Let (s, x) be the first edge on $P_{w'}$. Let $P_x = P_{w'} \setminus \{(s, x)\}$ be the remaining path from x to w'. Then

$$\begin{aligned}
\mathsf{wt}(P_{w'}) &= \mathsf{wt}(s, x) + \mathsf{wt}(P_x) && \textit{(since } P_{w'} = \{(s,x)\} \cup P_x\textit{)} \\
&\ge \mathsf{wt}(s, x) + 0 && \textit{(since } \mathsf{wt}(P_x) \ge 0\textit{)} \\
&\ge \mathsf{wt}(s, w') && \textit{(by the choice of } w'\textit{).}
\end{aligned}$$

So $\mathsf{wt}(P_{w'}) \ge \mathsf{wt}(s, w')$. This holds for an arbitrary $s \rightsquigarrow w'$ path $P_{w'}$, so every $s \rightsquigarrow w'$ path has weight at least $\mathsf{wt}(s, w')$. That is, $\mathsf{dist}(s, w') \ge \mathsf{wt}(s, w')$.

By Parts (i) and (ii) above, $\mathsf{dist}(s, w') = \mathsf{wt}(s, w')$. □

Extending the first step. The algorithm will maintain an array d to keep track of the shortest-path distances that it has determined so far. It can set $d[s] = 0$ immediately, and, for w' as chosen in the lemma, it could set $d[w'] = \mathsf{wt}(s, w')$. How to determine the distances to the remaining vertices?

Consider some more examples for intuition. Is there some neighbor w of s or w' whose shortest-path distance we can easily compute? Consider just the edges leaving the set $\{s, w'\}$... (In class we considered several examples...)

Let's summarize the structure for the algorithm that we have in mind so far. As the algorithm proceeds, at any point in time, it will have correctly determined the distances to some subset of the vertices. We'll let K denote that subset of vertices. So, the algorithm has somehow set $d[v] = \mathsf{dist}(s, v)$ for each vertex $v \in K$. In each iteration, it should determine the distance to one new vertex, and add that vertex to K. (Initially $K = \{s\}$ and $d[s] = 0$. After the first step, it will have $K = \{s, w'\}$, and $d[s]$ and $d[w']$ will be set.) Then the algorithm should follow this template:

```
Dijkstra$(G = (V, E), s)$:
1. initialize $K \leftarrow \{s\}$ and $d[s] = 0$
2. while ...
3.1. somehow select a vertex $w' \in V \setminus K$
3.2. somehow compute $d[w'] = \mathsf{dist}(s, w')$
3.3. add $w'$ to $K$
4. return the array $d$
```

It should maintain the greedy invariant, that the partial solution so far can be extended to a correct solution:

greedy invariant: *Each vertex $v \in K$ satisfies $d[v] = \mathsf{dist}(s, v)$.*

We need to design the loop to maintain this invariant. Imagine the start of some iteration, where the invariant holds just before some iteration. The iteration should make progress, while maintaining the invariant. Specifically, it should determine the distance $\mathsf{dist}(s, w')$ to some vertex w' that is not in K. Then it will set $d[w']$ correctly and add w' to K, maintaining the invariant.

Say that a path P *is a path out of* K if P starts at s and ends at a vertex that is not in K. The idea will be to find, *among all paths out of K, one of minimum weight, say, P^*.* That path must be a shortest path from s to its destination vertex, say w', because otherwise the shortest path would be a path out of K of less weight.

And one can find P^* as follows: consider just the edges leaving K, that is, $(u, w) \in E$ such that $u \in K$ and $w \notin K$. Among these edges, choose (u', w') that minimizes $d[u'] + \mathsf{wt}(u', w') = \mathsf{dist}(s, u') + \mathsf{wt}(u', w')$. Consider the path $P_{w'}$ from s to w' that consists of a shortest path from s to u', followed by the edge (u', w'). That path $P_{w'}$ will be a minimum-weight path out of K. (We verify this later.)

So, in the iteration, the algorithm will find the edge (u', w') as described above. It will set $d[w'] = d[u'] + \mathsf{wt}(u', w')$ and add w' to K, maintaining the invariant. Here is the full algorithm:

```
Dijkstra$(G = (V, E), s)$:     — Dijkstra's algorithm for SINGLE-SOURCE SHORTEST PATHS —
1. initialize $K \leftarrow \{s\}$ and $d[s] = 0$
2. while some edge in $E$ leaves $K$ (that is, $\exists (u, w) \in E : u \in K,\ w \notin K$):
3.1. among edges leaving $K$, choose edge $(u', w')$ that minimizes $d[u'] + \mathsf{wt}(u', w')$
3.2. set $d[w'] = d[u'] + \mathsf{wt}(u', w')$
3.3. add $w'$ to $K$
4. set $d[w] = \infty$ for $w \in V \setminus K$
5. return $d$
```

We explain the termination condition of the loop, and Step 4, later.

First we show that the loop in Line 2 maintains the invariant:

Lemma 3. *In any execution, the loop maintains the invariant* $d[v] = \mathsf{dist}(s, v)$ for all $v \in K$.

Proof (long form).

1. Consider any execution of the algorithm on some input $(G = (V, E), s)$.
2. The invariant holds at the start of the first iteration, when $K = \{s\}$ and $d[s] = 0 = \mathsf{dist}(s, s)$. (Because G has no negative-weight cycles.)

3.1. Consider any iteration of the loop such that the invariant holds at the start of the iteration.
3.2. Let K and d denote the set K and the array d at the start of the iteration.
3.3. Let (u', w') be the edge chosen in the iteration.
3.4. To show that the iteration maintains the invariant, we show $\mathsf{dist}(s, w') = d[u'] + \mathsf{wt}(u', w')$.
3.5. First we show $\mathsf{dist}(s, w') \leq d[u'] + \mathsf{wt}(u', w')$.
3.6. Since $u' \in K$, by the invariant, $\mathsf{dist}(s, u') = d[u'] < \infty$.
3.7. So there exists a path from s to u' of weight $d[u']$.
3.8. This path plus the edge (u', w') form a path from s to w' of weight $d[u'] + \mathsf{wt}(u', w')$.
3.9. So there exists a path from s to u' of weight $d[u'] + \mathsf{wt}(u', w')$.
3.10. So $\mathsf{dist}(s, u') \leq d[u'] + \mathsf{wt}(u', w')$.
3.11. To finish we'll show $\mathsf{dist}(s, w') \geq d[u'] + \mathsf{wt}(u', w')$.
3.12.1. Let P be an arbitrary path from s to w'.
3.12.2. P starts in K, but ends outside of K, so some edge on P leaves K.
3.12.3. Let (x, y) be the first edge on P that leaves K (so $x \in K$, $y \notin K$).
3.12.4. Let P_x be the prefix of P that ends at x. Let P_y be the suffix of P that starts at y.
3.12.5. So P_x is a path from s to x, P_y is a path from y to w', and $P = P_x \cup \{(x, y)\} \cup P_y$. Then

$$\begin{aligned}
\mathsf{wt}(P) &= \mathsf{wt}(P_x) + \mathsf{wt}(x, y) + \mathsf{wt}(P_y) && \textit{(Since } P = P_x \cup \{(x,y)\} \cup P_y.\textit{)} \\
&\geq \mathsf{wt}(P_x) + \mathsf{wt}(x, y) && \textit{(Since } \mathsf{wt}(P_y) \geq 0.\textit{)} \\
&\geq \mathsf{dist}(s, x) + \mathsf{wt}(x, y) && \textit{(Since } P_x \textit{ is an } s \rightsquigarrow x \textit{ path.)} \\
&\geq d[x] + \mathsf{wt}(x, y) && \textit{(Since } x \in K \textit{ so } d[x] = \mathsf{dist}(s, x).\textit{)} \\
&\geq d[u'] + \mathsf{wt}(u', w') && \textit{(By the choice of } (u', w')\textit{, and } (x, y) \in E'.\textit{)}
\end{aligned}$$

3.13. By Block 3.12, every path from s to w' has weight at least $d[u'] + \mathsf{wt}(u', w')$.
3.14. That is, $\mathsf{dist}(s, w') \geq d[u'] + \mathsf{wt}(u', w')$.
3.15. By this and Step 3.10, $\mathsf{dist}(s, w') = d[u'] + \mathsf{wt}(u', w')$.
4. By Block 3, each iteration that starts with the invariant true preserves the invariant.
5. The invariant holds initially, so the lemma follows. □

Given that the invariant holds, correctness is easy to verify:

Theorem 1. Dijkstra *is correct.*

Proof (long form).

1. Consider the execution of the algorithm on an arbitrary input (G, s).
2. Consider the time just before Line 4 executes.
3. By Lemma 3, at that time, $d[v] = \mathsf{dist}(s, v)$ for each $v \in K$.
4. By the loop condition, no edges leave K, so vertices in $V \setminus K$ are not reachable from s.
5. So each remaining vertex $v \in V \setminus K$ has $\mathsf{dist}(s, v) = \infty$.
6. So, after Line 4 executes, each vertex $v \in V \setminus K$ also has $d[v] = \mathsf{dist}(s, v)$.
7. So the distances returned by the algorithm are correct. □

Shortest-path trees. Given an instance (G, s), a *shortest-path tree* T is a (directed) tree, rooted at s, whose vertices are the vertices in G that are reachable from s, such that, for each reachable vertex v, the path from s to v in T is a shortest path from s to v.

For any execution of the algorithm, the collection of edges (u', w') selected in any iteration form such a tree T, where the parent of w' in T is u'. So the algorithm can easily be augmented to compute a shortest-path tree T, if desired.

Efficient implementation. Let $n = |V|$ and $m = |E|$ denote the number of edges of a given instance. The algorithm has at most n iterations, as each iteration adds a vertex to K. A naive implementation will take linear time per iteration, giving overall run time of $O(n(m + n))$.

To improve this, maintain the edges leaving K (that is, $(u, w) \in E \cap (K \times (V \setminus K))$) in a heap, keyed by $d[u] + \mathsf{wt}(u, w)$. Then each iteration can be implemented as follows: to find the edge (u', w'), use a find-min operation on the heap. Then, when adding w' to K, maintain the heap by adding all edges from w' into $V \setminus K$ to the heap, and removing all edges into w'.

Each heap operation takes time $O(\log n)$, so the iteration takes time $O(\log n + \mathsf{degree}(w') \log n)$. Since $\sum_{w' \in V} \mathsf{degree}(w') = 2m$, the total time over all iterations is then $O((m + n) \log n)$.

Be aware that most texts describe the algorithm slightly differently, as maintaining a heap of the *vertices* in $V \setminus K$, where the key of each vertex $w \in V \setminus K$ is the minimum of the keys that we would store for its incoming edges. That is $\mathsf{key}(w) = \min\{d[u] + \mathsf{wt}(u, w) : (u, w) \in E, u \in K\}$.

External resources

- text: CLRS Chapter 24
- text: Kleinberg & Tardos Section 4.4
- text: Edmonds Lecture 21

  http://jeffe.cs.illinois.edu/teaching/algorithms/notes/21-sssp.pdf
- text: Dasgupta et al. Section 4.4
- MIT lecture (video):
  https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-006-introduction-to-algorithms-fall-2011/lecture-videos/lecture-15-single-source-shortest-paths-problem/