Dijkstra's Algorithm

Bellman-Ford tightens every edge \( n-1 \) times. Inefficient!

Dijkstra tightens each edge once. Works harder to find the right edge to tighten.

In each round, Dijkstra "visits" a vertex, tightening all edges out of the vertex.

Chooses the unvisited vertex closest to \( S \).

[But we don't know all distances yet! So it picks the vertex of minimum \( c() \).]

\[
\text{Dijkstra}(G, s):
\begin{align*}
    c(w) &= \infty \quad \forall w \\
    c(s) &= 0 \\
    S &= \emptyset
\end{align*}
\]

While \( S \neq V \):

- Find \( u \in (V - S) \) minimizing \( c(u) \).
- \( S \leftarrow S + \{u\} \)

For each edge \( (u, v) \) from \( u \) in \( E \):

\[
    c(v) \leftarrow \min(c(v), c(u) + \text{cost}(u \to v))
\]

"Visit u"

 Tighten \( (u, v) \)
Correctness

For simplicity, suppose \( c^*(u) \) all unique

Full proof on Piazza

Can ordered \( u \in V \) by distance flow as:
\[
S = u_1, u_2, \ldots, u_n
\]
\[
c(u_1) < c(u_2) < c(u_3) < \ldots < c(u_n)
\]

**Lemma:**
The \( k^{th} \) node visited = \( u_k \)
and \( c(u_k) = c^*(u_k) \) when it is visited.

**Proof Trivial for \( k = 1 \).**
If true for all \( k < k \)
then consider the state just before
choosing the \( k^{th} \) node to visit.

**Claim:** \( c(u_k) = c^*(u_k) \).

Let \( u' = \text{pred}(u_k) \) = previous node to \( u_k \)
in shortest \( S \rightarrow u_k \) path.
\[
c^*(u') = c^*(u_k) - \text{cost}(u' \rightarrow u_k)
\leq c^*(u_k)
\]

because nonnegative edges.

Uniqueness assumption \( \Rightarrow c^*(u') < c^*(u_k) \)
\( \Rightarrow u' \) before \( u_k \) in order
\( \Rightarrow \) already visited \( u' \) and
\( \Rightarrow \) when visited \( u' \) we set
\[
c(u_k) \leq \min(c(u_k), c^*(u') + \text{cost}(u' \rightarrow u_k))
= c^*(u_k).
\]
So we have $c(u_i) \geq c^*(u_i) \geq c^*(u_k) = c(u_k)$
when deciding on the $k^{th}$ node to visit.
We've already visited $u_i \forall i < k$
and all other $i$ have
$c(u_i) \geq c^*(u_i) > c^*(u_k) = c(u_k)$.

Hence Dijkstra will choose $u_k$ in the $k^{th}$ round
with $c(u_k) = c^*(u_k)$.

Since we visit every node and $c(u) = c^*(u)$
when it is visited, Dijkstra eventually
gets each $c(u) = c^*(u)$, proving correctness.

**Running time**

Time = $O(\text{time to tighten } M \text{ edges} + \text{time to find the } N \text{ vertices to visit})$

Simplest approach:
Look through all $V$ to decide
node to visit.
$\Rightarrow O(1) \text{ tighten, } O(n) \text{ time to find each } u.$
$\Rightarrow O(m + n^2) = O(n^2) \text{ running time.}$
Better than Bellman-Ford!
Better approach: Store $V - S$ in a binary heap keyed by $c()$

\[ \text{Node to visit} = \text{delete-min on heap} = O(\log n) \text{ time.} \]

but now `tighten()` changes a $c()$

\[ \Rightarrow \text{need to bubble up that node in heap} \]

"decrease-key operation"

\[ = O(\log n) \text{ time.} \]

\[ \Rightarrow \text{total time} = O(m \log n + n \log n) \]

\[ = O(m \log n). \]

Fanciest approach: use a Fibonacci heap

\[ \text{delete-min: } O(\log n) \]

\[ \text{decrease-key: } O(1) \text{ (amortized)} \]

\[ \Rightarrow O(m + n \log n) \text{ time.} \]

[In practice, use a binary heap]