Dijkstra's Algorithm

Bellman-Ford tightens every edge $n-1$ times. Inefficient!

Dijkstra tightens each edge <u>once</u>. 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 $c()$.]

```
Dijkstra(G, s):
    c(u) = ∞   ∀u
    c(s) = 0
    S = {}
    While S ≠ V:
        find u ∈ (V − S) minimizing c(u).
        S ← S + {u}
        For each edge (u,v) from u in E:
            c(v) ← min(c(v), c(u) + cost(u->v))
```

"visit u"

tighten(u, v)

Correctness

For simplicity, suppose $c^*(u)$ all unique
[Full proof on Piazza]

Can order $u \in V$ by distance from s:

$$s = u_1, u_2, \ldots, u_n$$
$$c(u_1) < c(u_2) < c(u_3) < \ldots < c(u_n)$$

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

Pf 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 $s \to u_k$ path.

$$c^*(u') = c^*(u_k) - \text{cost}(u' \to u_k)$$
$$\leq c^*(u_k)$$

because nonnegative edges.
uniqueness assumption $\Rightarrow c^*(u') < c^*(u_k)$
$\Rightarrow u'$ before u_k in order
inductive hypo $\Rightarrow$ already visited u', and
$c(u') = c^*(u')$ when it was visited
$\Rightarrow$ when visited u' we set

$$c(u_k) \leftarrow \min(c(u_k), \underbrace{c^*(u') + \text{cost}(u' \to u_k)}_{c^*(u_k)})$$
$$= c^*(u_k).$$

So we have $c(u_k) = c^*(u_k)$
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 = $O($ time to tighten m edges
+ time to find the n vertices
to visit$)$

Simplest approach:
Look through all V to decide
node to visit.
$\Rightarrow O(1)$ tighten, $O(n)$ time to find each u.
$\Rightarrow O(m + n^2) = O(n^2)$ running time.
Better than Bellman-Ford!

Better approach: store $V - S$ in a binary heap keyed by $c()$

node to visit = delete-min on heap
= $O(\log n)$ time.

but now tighten() changes a $c()$
⇒ need to bubble up that node in heap.
"decrease-key operation"
= $O(\log n)$ time.

⇒ total time = $O(m \log n + n \log n)$
= $O(m \log n)$.

Fanciest approach: use a Fibonacci heap
delete-min: $O(\log n)$
decrease-key: $O(1)$
(amortized)
⇒ $O(m + n \log n)$ time.

[in practice, use a binary heap]