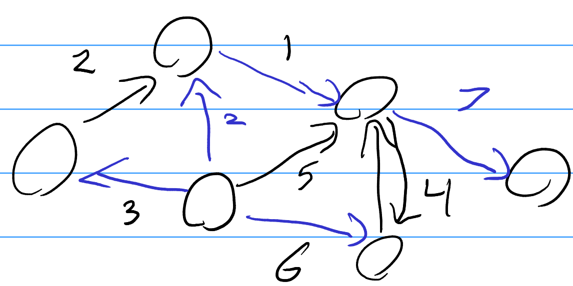
Shortest Paths

Given a directed graph G
edges have <u>costs</u>: $cost(u \to v)$

Path length = sum of individual edge costs

Want to find shortest paths in G.

Shortest paths from a source s
form a tree:



[If shortest $s \to t$ path is
$s = u_1 \to u_2 \to \dots \to u_{k-1} \to u_k = t$,

then shortest $s \to u_{k-1}$ path
is also $s = u_1 \to u_2 \to \dots \to u_{k-2} \to u_{k-1}$.]

Single - Source Shortest Paths (SSSP):

Find shortest path tree from s.

Point - to - point:

Find shortest $s \to t$ path

Algorithm = run SSSP from s

Find t in the tree

[nothing better known in general!]

All pairs shortest paths (APSP):

Find all shortest paths.

Algorithm = run SSSP for all s.

So how to solve SSSP?

Let $c^*(u)$ = true shortest path length to u

<u>Triangle Inequality</u> says:

For all $(u \to v)$ edges,

$$c^*(v) \le c^*(u) + \text{cost}(u \to v).$$

[can get to v by taking this edge]

Generic algorithm:

Start with upper bound $c()$ on $c^*()$

Repeatedly pick edges somehow

and apply triangle inequality.

More formally:

Generic (G, s):

Set $c(s) = 0, \quad c(u) = \infty \quad \forall u \neq s$

Repeatedly pick edges (u, v) somehow:

$$c(v) \Leftarrow \min\big(c(v),\ c(u) + \text{cost}(u \to v)\big)$$

"Relax"(u, v)

image: edge is spring of given length, go from stretched to relaxed

Lemma: No matter how edges are picked, $c(v) \geq c^*(v) \quad \forall v$ at all times.

Pf Starts true.

If it's true at any point, updates have

$$c(v) \Leftarrow \min\big(\underbrace{c(v)}_{\geq c^*(v)},\ \underbrace{c(u)}_{\geq c^*(u)} + \text{cost}(u \to v)\big)$$

$$\geq \min\big(c^*(v),\ c^*(u) + \text{cost}(u \to v)\big)$$

$$\geq c^*(v)$$

by the triangle inequality.

Hence it remains true. ∎

When does it get to the true answer?

Lemma

If $s = u_1 \to u_2 \to \dots \to u_k$ is true shortest $s \to u_k$ path, and relax is called on (u_1, u_2), $(u_2, u_3) \dots$ (u_{k-1}, u_k) in order — possibly w/ intervening calls, before, between and after — then $c(u_k) = c^*(u_k)$.

Pf

We induct on k.

$k = 1 \Rightarrow u_k = s$, so $c(u_k) = 0 = c^*(u_k)$ to start, and it never increases.

Otherwise, by induction
$c(u_{k-1}) = c^*(u_{k-1})$
when relax is called on (u_{k-1}, u_k).
Then this call sets

$$
\begin{aligned}
c(u_k) &= \min(c(u_k),\ c(u_{k-1}) + \mathrm{cost}(u_{k-1} \to u_k)) \\
&\le c(u_{k-1}) + \mathrm{cost}(u_{k-1} \to u_k) \\
&= c^*(u_{k-1}) + \mathrm{cost}(u_{k-1} \to u_k) \\
&= c^*(u_k).
\end{aligned}
$$

and later calls cannot increase it. ∎

So we need to relax every edge of the path in order.

Bellman-Ford Algorithm:

n-1 times:

relax every edge.

The i^{th} iteration relaxes every edge

$\Rightarrow$ relaxes the i^{th} edge on path

Paths have $\leq n-1$ edges

(Assuming no negative cycles!)

$\Rightarrow$ relaxed all edges in order after n-1 iterations

$\Rightarrow$ Correct.

Running time $O(mn)$

Best known algorithm for general graphs!

Can do better if edge lengths nonnegative by Dijkstra's algorithm.

Dijkstra's Algorithm:

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

Dijkstra relaxes each edge once. Works harder to find the right edge to relax.

In each round, Dijkstra "visits" a vertex, relaxing 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):

$c(u) = \infty \quad \forall u$

$c(s) = 0$

$S = \{\}$

While $S \neq V$:

Find $u \in (V - S)$ minimizing $c(u)$. 1

"visit u"

$S \leftarrow S + \{u\}$

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

$c(v) \leftarrow \min(c(v),\ c(u) + cost(u \to v))$

relax(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 $u' = pred(u_k)$ = previous node to u_k in shortest $s \to u_k$ path.
$c^*(u') = c^*(u_k) - cost(u' \to u_k)$
$\leq c^*(u_k)$
because nonnegative edges.
Uniquess 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

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

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 relax m edges
+ time to find the n vertices
to visit)

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

Better approach: store unvisited vertices, $V \setminus S$, in a binary heap keyed by $c()$

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

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

$\Rightarrow$ 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)
$\Rightarrow$ $O(m + n \log n)$ time.

[in practice, use a binary heap]