CSCI 3100

SSSP (cont)

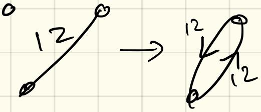
Today

- No office hours today
- Monday: sign up for Friday HW slot

Next problem: Shortest paths

Goal: Find shortest path from s to v.

We'll think directed, but really could do undirected w/no negative edges:



Motivation:

- maps
- routing

Usually, to solve this, need to solve a more general problem:

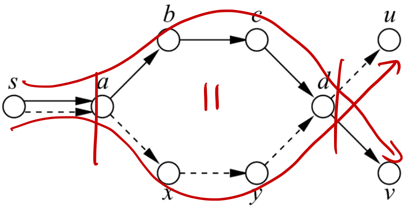
Find shortest paths from s to *every* other vertex.

Called the single-source shortest path tree.

SSSP

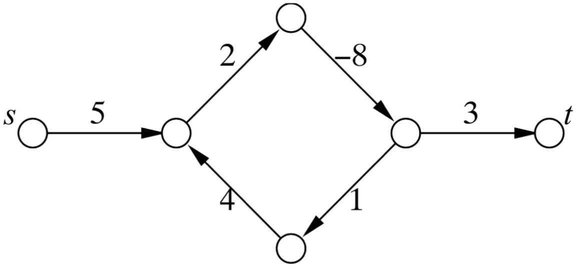
Some notes:

- Why a tree?



If $s \to a \to b \to c \to d \to v$ and $s \to a \to x \to y \to d \to u$ are shortest paths, then $s \to a \to b \to c \to d \to u$ is also a shortest path.

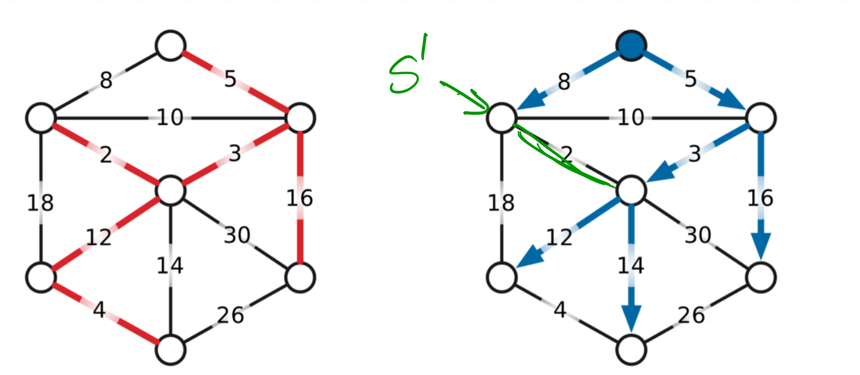
- Negative edges? (initially, assume none)



There is no shortest path from s to t.

Important to realize:

MST ≠ SSSP



Why? SSSP is rooted & directed

- SSSP for every vertex (these are different!)

Computing a SSSP:

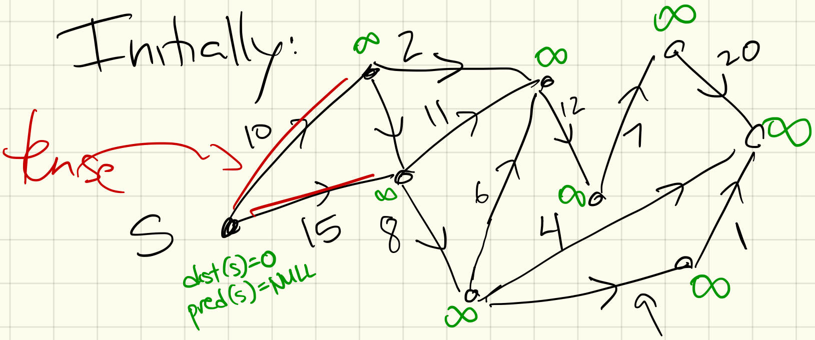
(Ford 1956 & Dantzig 1957)

Each vertex will store 2 values.

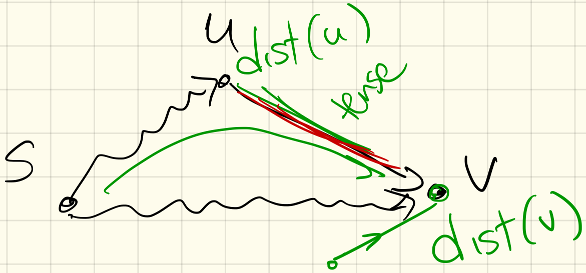
(Think of these as tentative shortest paths)

- dist(v) is length of tentative shortest s⇝v path
  (or ∞ if don't have an option yet)
- pred(v) is the predecessor of v on that tentative path s⇝v
  (or NULL if none)

Initially:



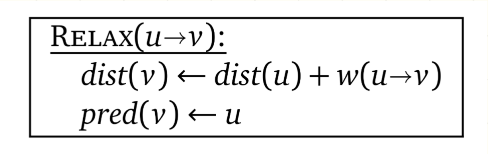
We say an edge $\overrightarrow{uv}$ is tense
if $dist(u) + w(u \to v) < dist(v)$



If $u \to v$ is tense:

use the better path!

So, relax:

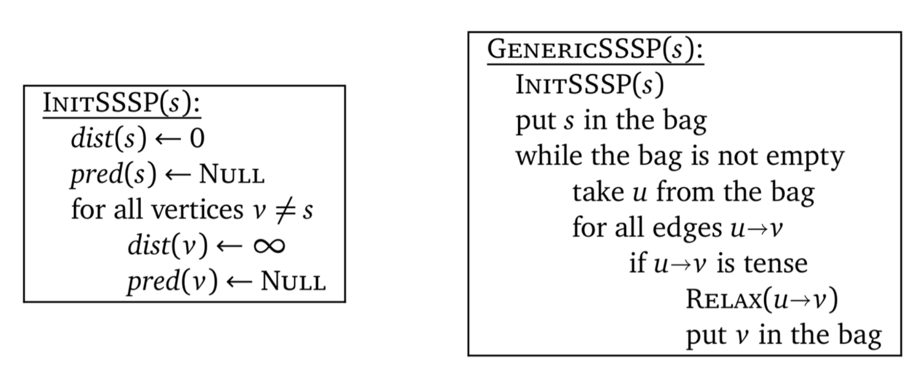


```
RELAX(u→v):
    $dist(v) \leftarrow dist(u) + w(u \to v)$
    $pred(v) \leftarrow u$
```

Algorithm: (Dantzig

Repeatedly find tense edges & relax them.

When none remain, the pred(v) edges form the SSSP tree.



```
INITSSSP(s):
    dist(s) ← 0
    pred(s) ← NULL
    for all vertices v ≠ s
        dist(v) ← ∞
        pred(v) ← NULL
```

```
GENERICSSSP(s):
    INITSSSP(s)
    put s in the bag
    while the bag is not empty
        take u from the bag
        for all edges u→v
            if u→v is tense
                RELAX(u→v)
                put v in the bag
```

To do: which "bag"?

Dijkstra ('59)

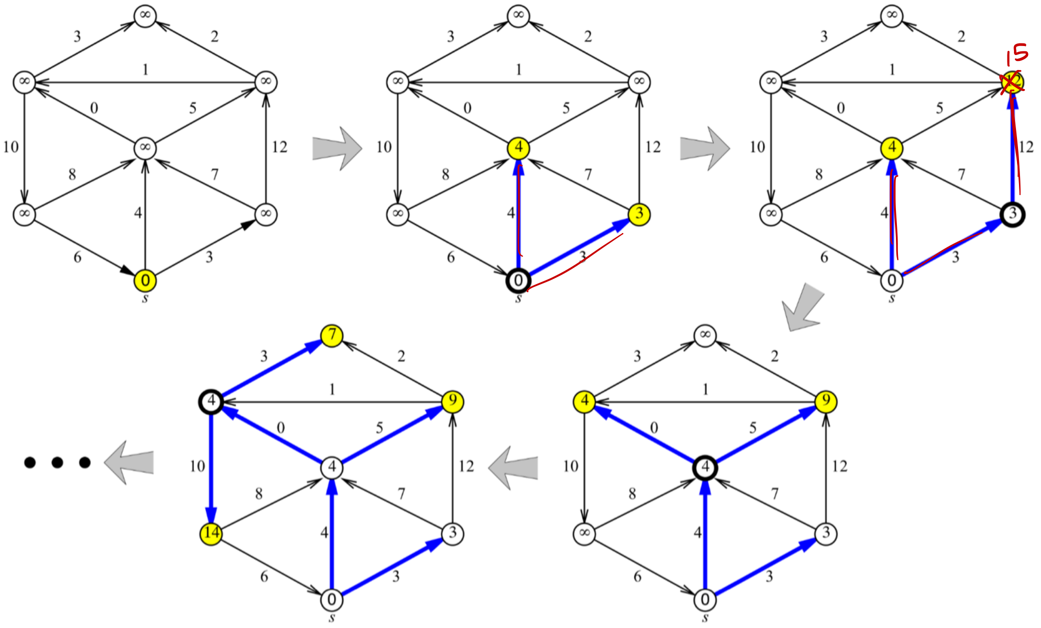
(actually Leyzorek et al '57, Dantzig '58)

Make the bag a priority queue:

Keep "explored" part of the graph, S.

```
Initially, $S = \{s\}$ + $dist(s) = 0$
    (all others NULL + $\infty$)
While $S \neq V$:
    Select node $v \notin S$ with one edge from $S$ to $v$ with:
        $\min_{e=(u,v),\, u \in S} dist(u) + w(u \to v)$
    Add $v$ to $S$, set $dist(v)$ + $pred(v)$
```

Picture →



Four phases of Dijkstra's algorithm run on a graph with no negative edges.
At each phase, the shaded vertices are in the heap, and the bold vertex has just been scanned.
The bold edges describe the evolving shortest path tree.

Correctness

Thm: Consider the set S at any point in the algorithm. For each $u \in S$, the distance $dist(u)$ is the shortest path distance (so $pred(u)$ traces a shortest path).

Pf: Induction on $|S|$:

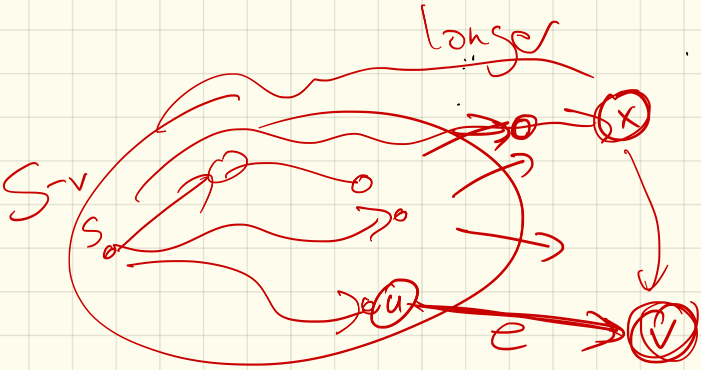
Base case: $|S| = 1$ ✓

$d(s) = 0$.

IH: Claim holds when $|S| = k-1$.

IS: Consider when $|S| = k$, & $\underline{v}$ was added to get there.

Let $e = u \to v$ into v getting us to v.



Claim: any other $s \rightsquigarrow x$ path where $x \notin S-v$ is longer than $s \rightsquigarrow u \xrightarrow[e]{} v$

On any $s \rightsquigarrow x$ path, some edge left set S!

Portion of $s \rightsquigarrow x$ path up to 1st edge leaving *was* considered when I added $u \to v$.

$\Rightarrow$ that portion of $s \rightsquigarrow x$ wasn't chosen, so it was $> dist(u) + w(e)$.

So no other paths to v *could* be shorter

$\Rightarrow$ S is best.

Back to implementation + run time:

For each $v \in S$, could check each edge + compute $\mathrm{dist}(v) + w(e)$

$w(\cancel{v} \to x)$

runtime?

$\underline{O(mn)}$

(or worse)

Better: a heap! of vertices

When $\underline{v}$ is added to S:

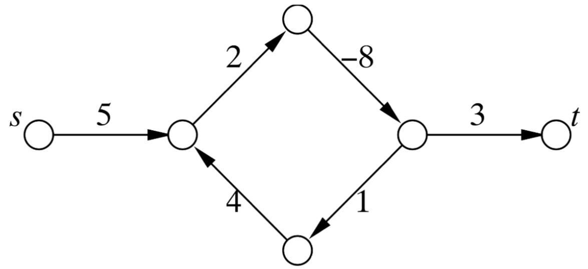
- look at v's edges and
  either insert $O(\log n)$ w with key
  $\mathrm{dist}(v) + w(v \to w)$
  or update w's key
  if $\mathrm{dist}(v) + w(v \to w)$ beats current one
  $O(\log n)$

Runtime:

- at most m ChangeKey operations in heap
- at most n inserts / removes

$$\underline{O(m \log n)}$$

What about negative edges?



There is no shortest path from *s* to *t*.

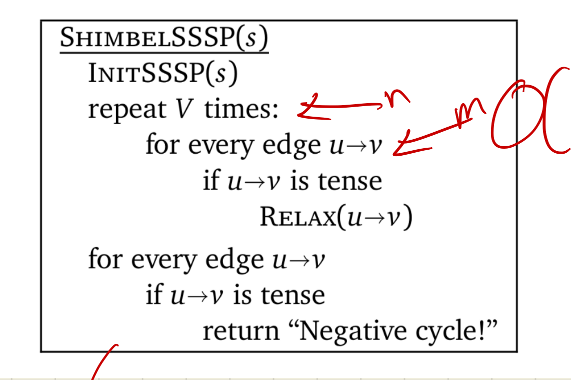
Bellman-Ford ('58)

(Actually, Shimbel '55)

Key: use dynamic programming to force a path to use each edge at most once.

$$dist_i(v) = \begin{cases} 0 & \text{if } i = 0 \text{ and } v = s \\ \infty & \text{if } i = 0 \text{ and } v \neq s \\ \min \left\{ \begin{array}{c} dist_{i-1}(v), \\ \min\limits_{u \to v \in E} (dist_{i-1}(u) + w(u \to v)) \end{array} \right\} & \text{otherwise} \end{cases}$$

Notes cover 2 ways to formalize this:



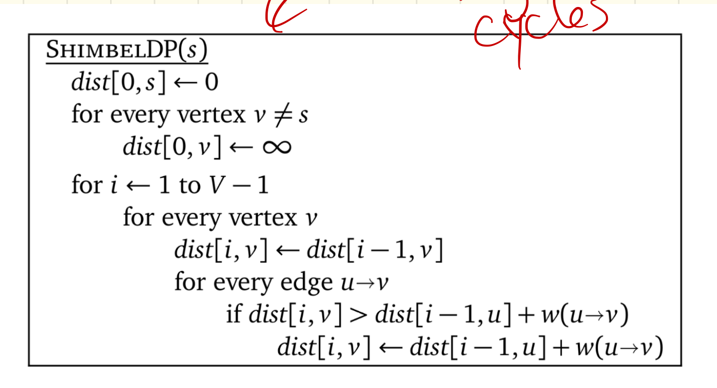
```
ShimbelSSSP($s$)
    InitSSSP($s$)
    repeat $V$ times:
        for every edge $u \to v$
            if $u \to v$ is tense
                Relax($u \to v$)
    for every edge $u \to v$
        if $u \to v$ is tense
            return "Negative cycle!"
```

← n

← m $O(mn)$

↳ detects (but doesn't work) negative cycles

work w/ negative cycles



```
ShimbelDP($s$)
    $dist[0,s] \leftarrow 0$
    for every vertex $v \neq s$
        $dist[0,v] \leftarrow \infty$
    for $i \leftarrow 1$ to $V-1$
        for every vertex $v$
            $dist[i,v] \leftarrow dist[i-1,v]$
            for every edge $u \to v$
                if $dist[i,v] > dist[i-1,u] + w(u \to v)$
                    $dist[i,v] \leftarrow dist[i-1,u] + w(u \to v)$
```

(more in notes...)