Algorithms

Shortest paths 2:
Dijkstra

Recap

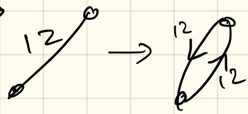
- HW due Friday
- Next HW: due Friday, Nov. 22
- Then expect 2 more, due
  - Dec. 2
  - Dec. 9

Tentative!

~~Next~~ Our 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.

Computing a SSSP:

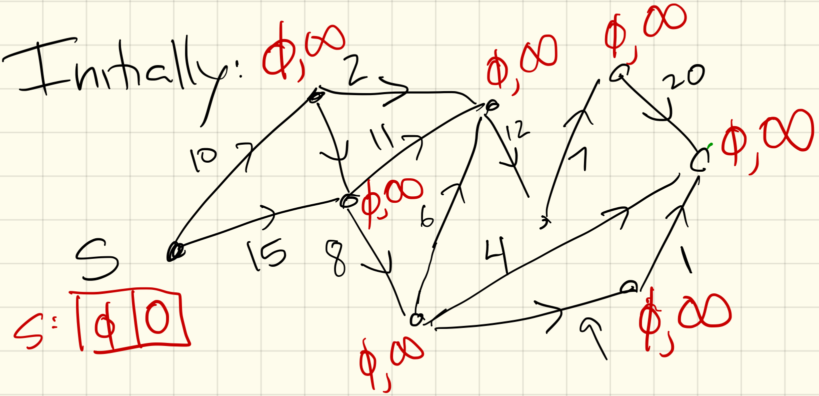
(Ford 1956 & Dantzig 1957)

Each vertex will store 2 values.

(Think of these as tentative shortest paths)

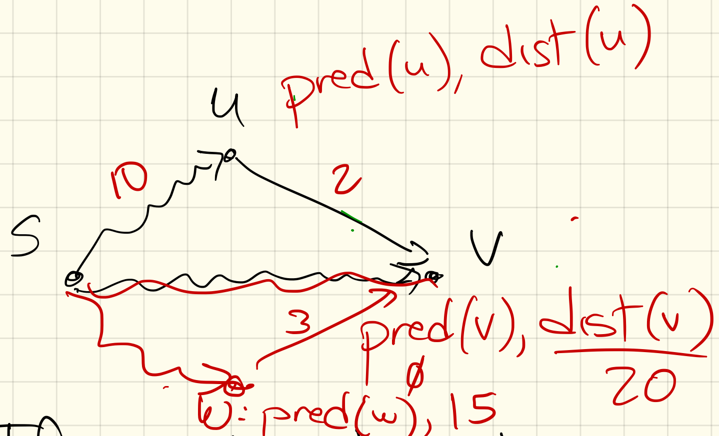
- dist(v) is length of tentative shortest $s \rightsquigarrow v$ path
  (or ∞ if don't have an option yet)
- pred(v) is the predecessor of v on that tentative path $s \rightsquigarrow v$
  (or NULL if none)

Initially:



We say an edge $\overrightarrow{uv}$ is <u>tense</u>

if $dist(u) + w(u\to v) < dist(v)$



If $u \to v$ is tense:

path via u is better

so: $pred(v) = u$

$dist(v) = dist(u) + w(u\to v)$

So, relax!

RELAX$(u\to v)$:

$dist(v) \leftarrow dist(u) + w(u\to v)$

$pred(v) \leftarrow u$

Algorithm:

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"?

(+ proof)

In DAGs: top layout

Easier! Can lay out
so all edge / are forward

Then: for $i = 2$ to n

find SP to v_i

How?

for $j = 1$ to $i-1$

try $dist(v_j) + w(v_j \to v_i)$ (if edge exists)

keep best one

already know SP tree up to v_i

Dijkstra ('59)

(actually Leyzorek et al '57, "plus more")

Make the bag a priority queue:

Keep "explored" part of the graph, S.

Initially, $S = \{s\}$ & $dist(s) = 0$
(all others NULL & ∞)

While $S \neq V$:

find best vertex ~~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 $dist(v)$ & $pred(v)$ accordingly

Claim: v belongs in SP tree w/ dist = $dist(v)$

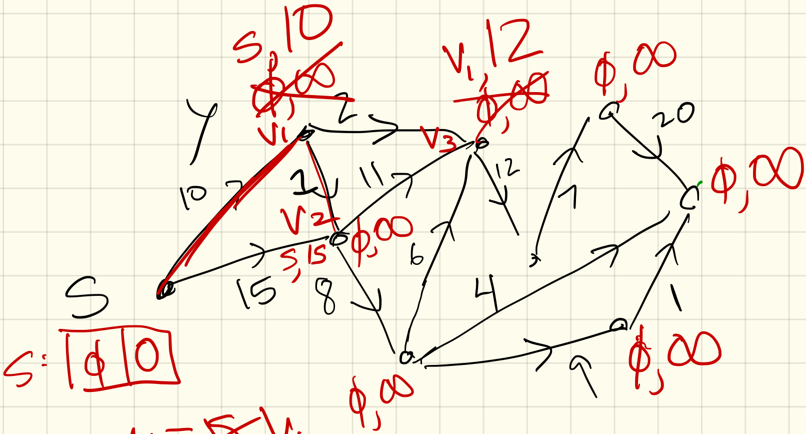
Nicer version →

sets ∅, ∞ on all vertices but s

```
DIJKSTRA(s):
    INITSSSP(s)
    INSERT(s, 0)
    while the priority queue is not empty
        u ← EXTRACTMIN()
        for all edges u→v
            if u→v is tense
                RELAX(u→v)
                if v is in the priority queue
                    DECREASEKEY(v, dist(v))
                else
                    INSERT(v, dist(v))
```

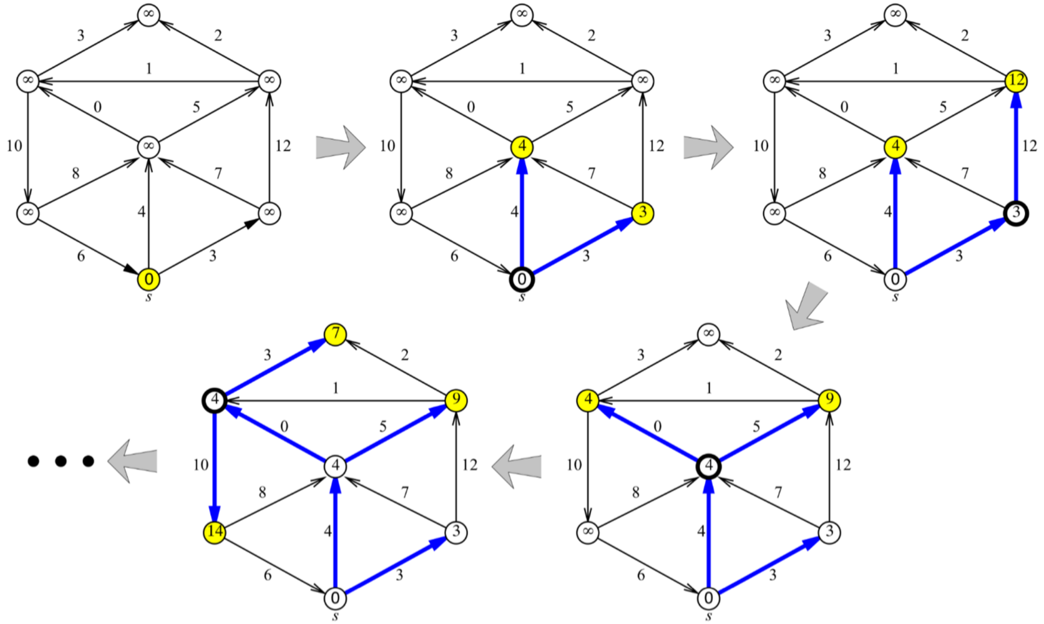
pop, then re-insert

Figure 8.11. Dijkstra's algorithm.



PQ ~~s, 0~~ $u = \cancel{s}\ v_1$ ~~v_1, 10~~ ~~v_2, 11~~

~~v_2~~, 15 v_3, 12



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.

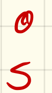
Correctness (if no negative edges)

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: $|S| = 1$

$S = \{s\}$



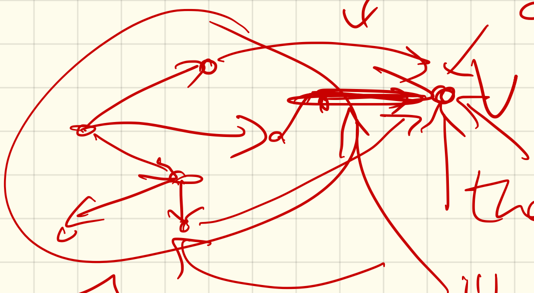
distance in SP tree is 0

IH: Spps claim holds when $|S| = k$.

IS: Consider $|S| = k+1$:

algorithm is adding some $\underline{v}$ to S

S



none of these are tense (b/c I relaxed them already)

"best" guess dist(v) is actually shortest path.

Assume: have correct SP-tree

If no negative edges, then no other path can beat this one (or else S wasn't SP tree)

Back to implementation + run time:

For each $v \in S$, could check each edge + compute $D[v] + w(e)$

runtime? $O(E)$

(ick)

↳ think data structures

Better: a heap!

When v is added to S:

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

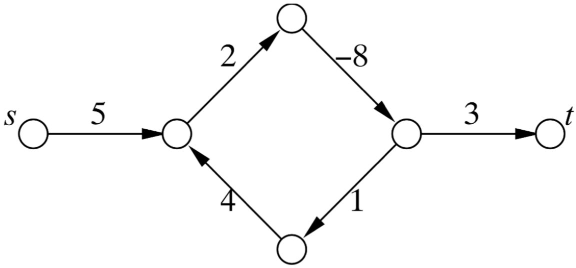
Runtime:

- at most ~~n~~ E ChangeKey operations in heap
- at most ~~n~~ V inserts/removes

each $\log V$

$$\Rightarrow O(E \log V) \text{ (ish)}$$

What about negative edges again?



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.

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

pre-hint for reading

Next time:

Finish SSSP

Friday: NP-Hardness