Algorithms - fall 2020

Shortest paths (part 2)

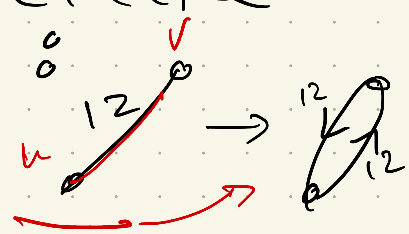
Recap:

- HW: due in 1 week
- Next HW - possibly will be 2^{nd} oral HW.
  (Stay tuned)

Next problem: Shortest paths

Goal: Find shortest path from s to v.

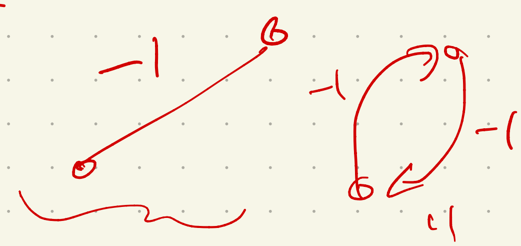
We'll think directed, but really could do undirected w/no negative edges :



Motivation:

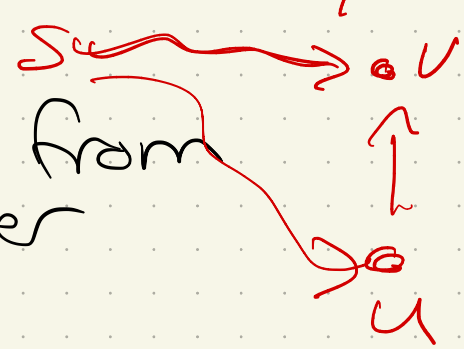
- maps
- routing

abstract but weighted



introduced neg. cycles

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

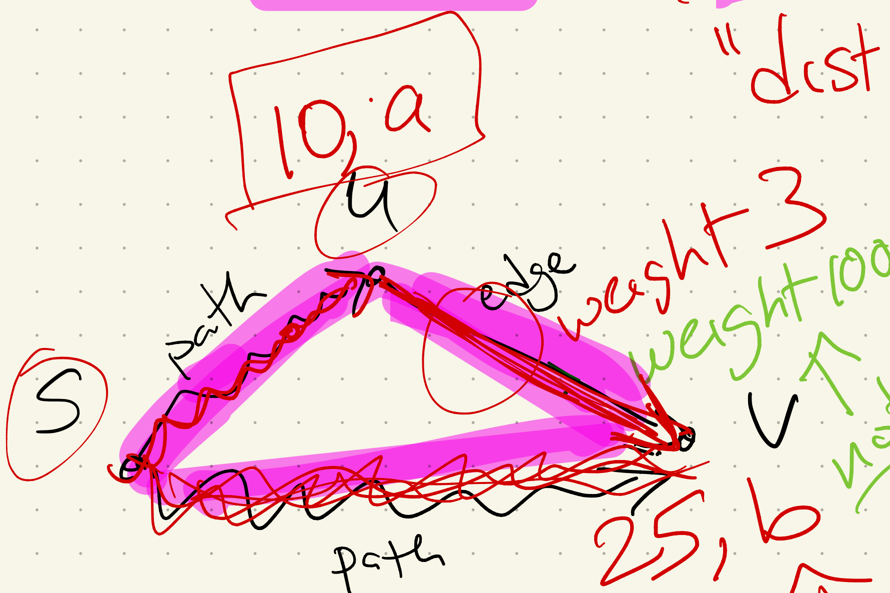


Find shortest paths from s to every other vertex.

Called the single-source Shortest path Tree.

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

"dist" is that best guess so far



better path to v!

v is bad guess, could improve

If $u \to v$ is tense:

update w/ better path!

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

+ update previous node

So, relax:

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

take uv edge (plus u's distance) as v's new "guess"

Computing a SSSP:

(Ford 1956 & Dantzig 1957)

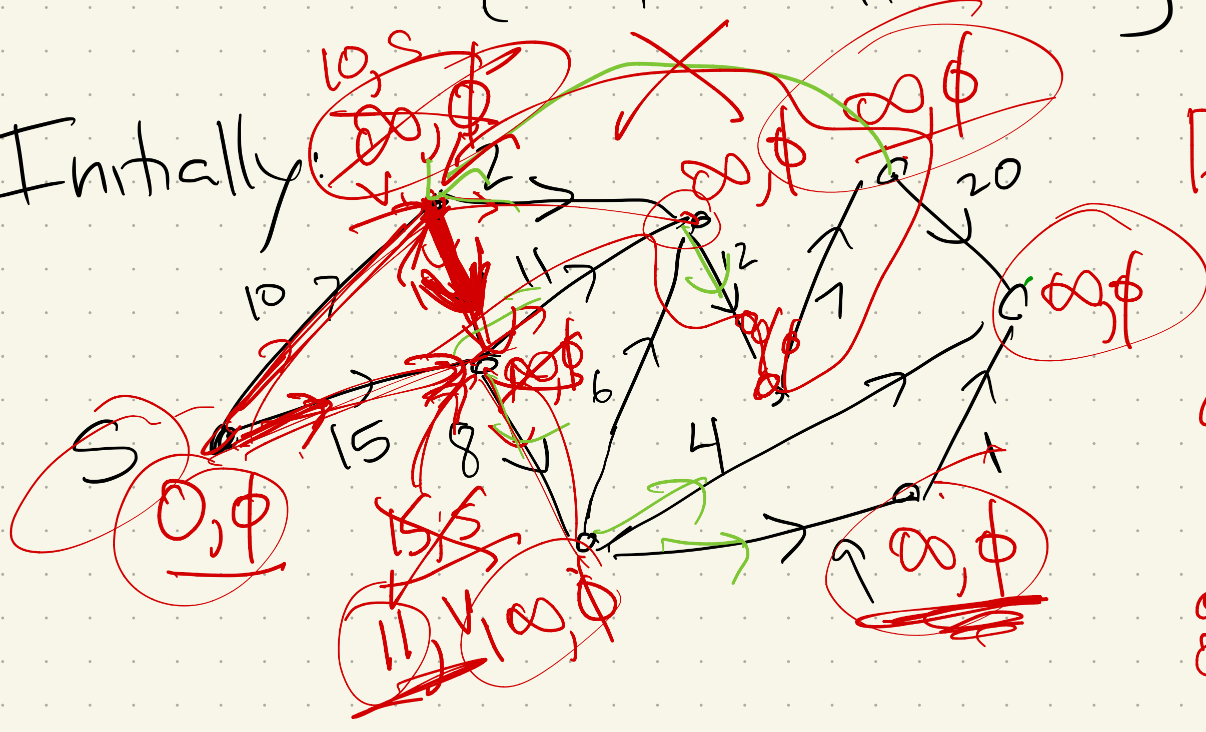
Each vertex will store 2 values.

(Think of these as tentative shortest paths)

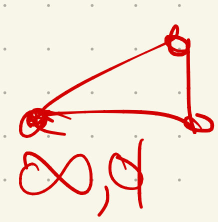
- dist(v) is length of tentative shortest s⤳v path
  (or ∞ if don't have an option yet)

- pred(v) is the predecessor of v on that tentative path s⤳v
  (or NULL if none)

Initially:



like DFS/BFS tree → use array



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 + ∞)

While $S \neq V$:

grow S

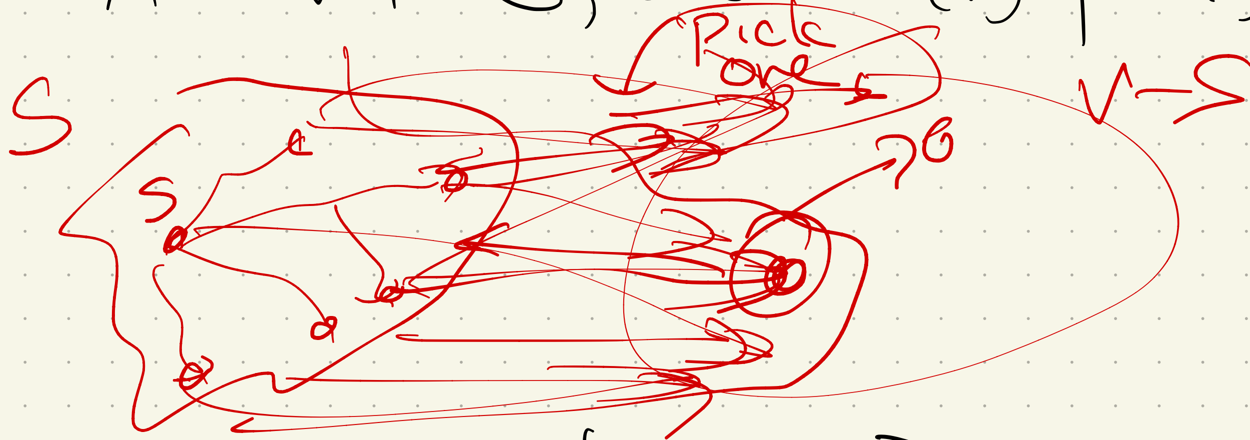
Select node $v \notin S$ with one edge from S to v with:

greedy!

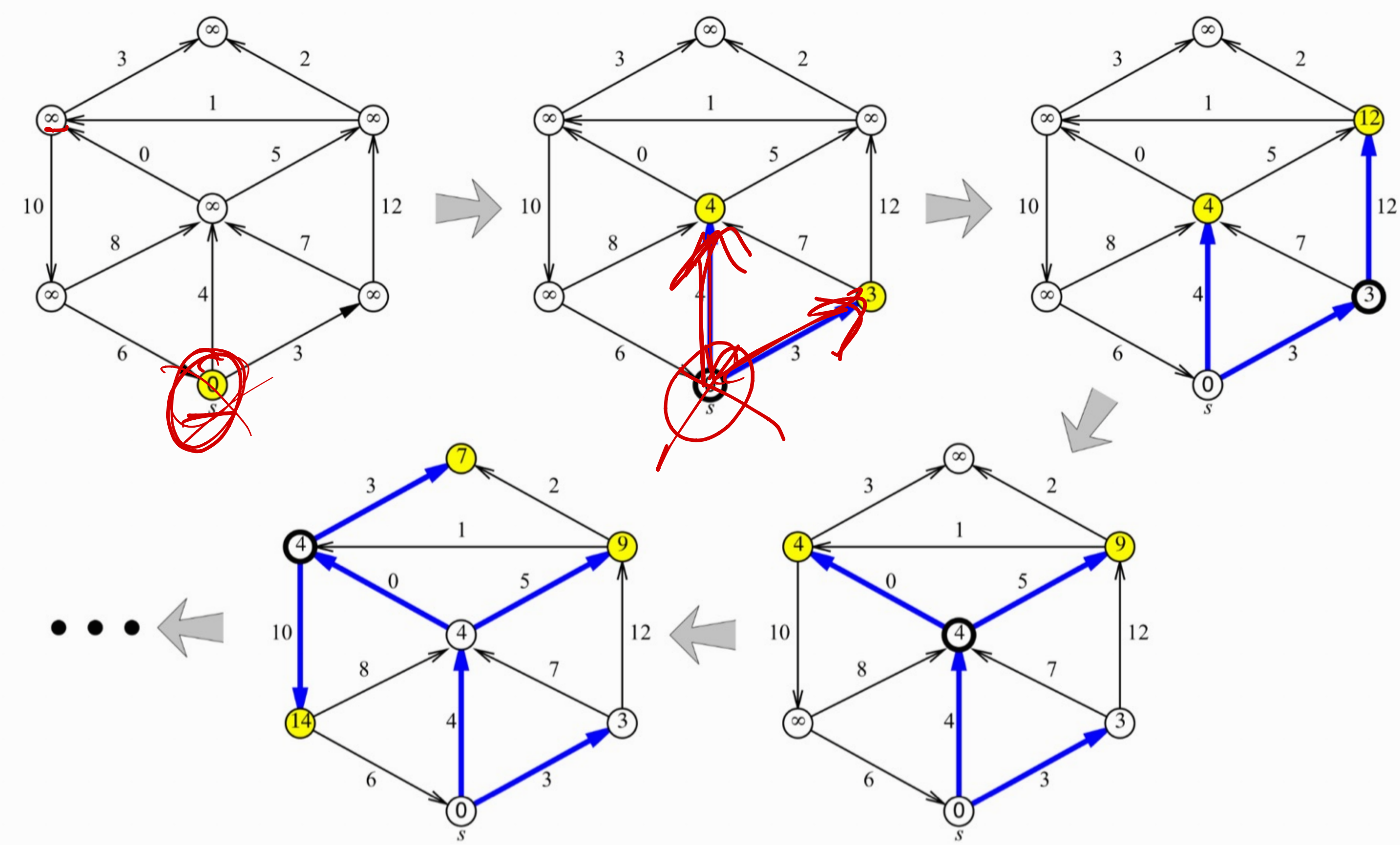
$$\min_{e=(u,v),\, u \in S} \text{dist}(u) + w(u \to v)$$

"lightest" edge

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.

PQ: vertices, + best guesses (as priority)

sets s's value to (0, ∅) + all others to (∞, ∅)

```
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))
```

Figure 8.11. Dijkstra's algorithm.

insert
removeMin
getMin
changeKey

relax - found better path

at end, (v, dist) pairs encode SP-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: if $|S| = 1$:

then $S = \{s\}$ (one vertex)

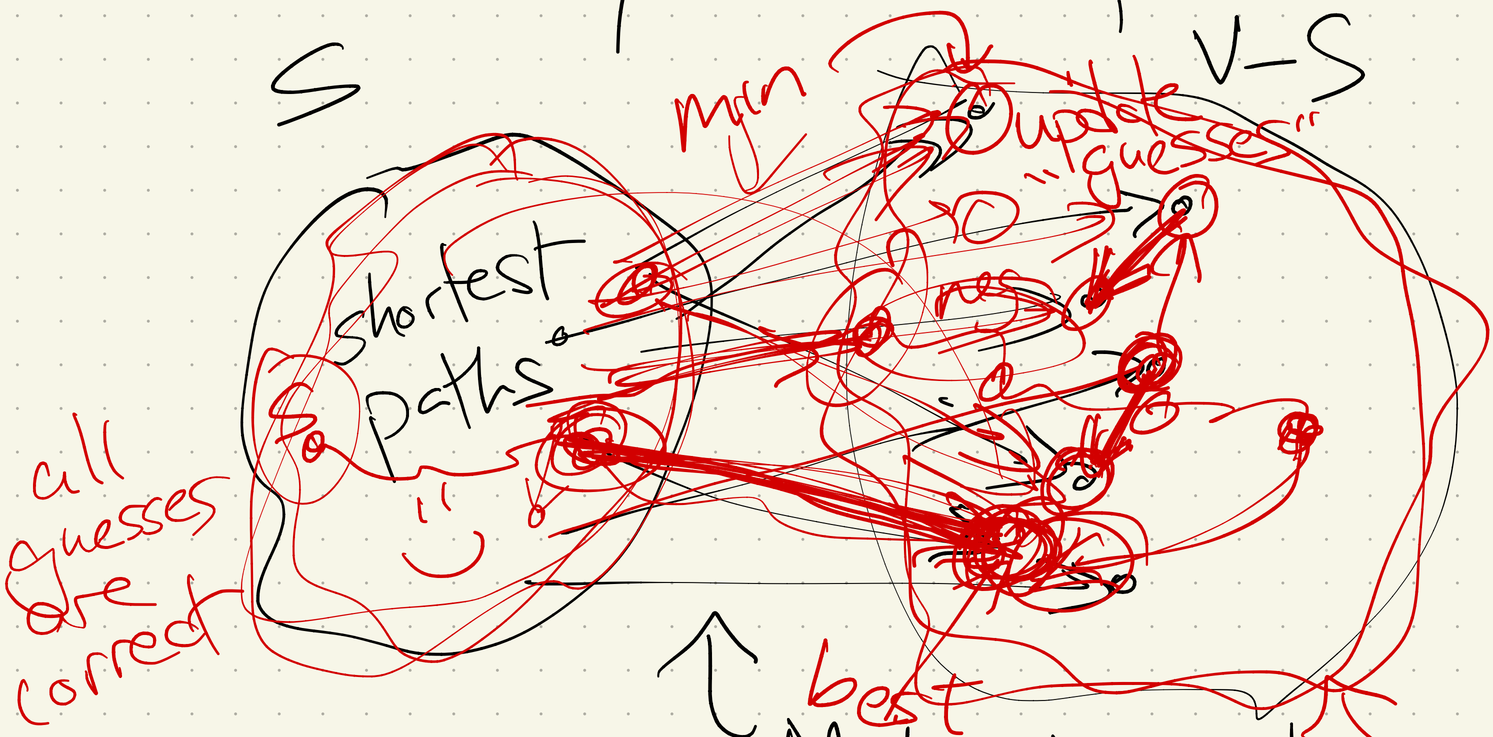
s has distance 0 to itself!

so $(s, \emptyset)$ is correct

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

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

algorithm is adding some v to S, which (by IH) we know has only shortest paths



Not tense!

(b/c alg relaxed)

Claim: min is correct.

If no negative edges, no other path can beat one in S.

Back to implementation + run time:

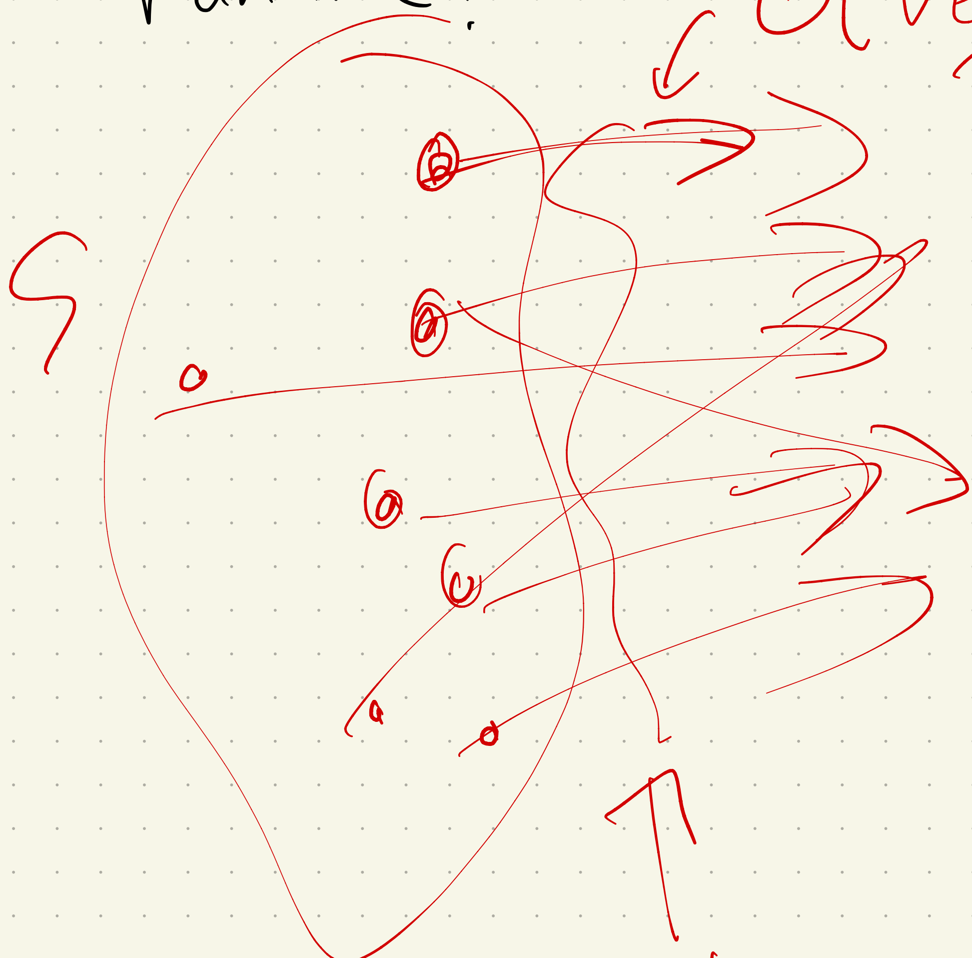
For each $v \in S$, could check each edge + compute $D[v] + w(e)$

runtime?

\# reps

$O(VE)$

time to search all edge lists

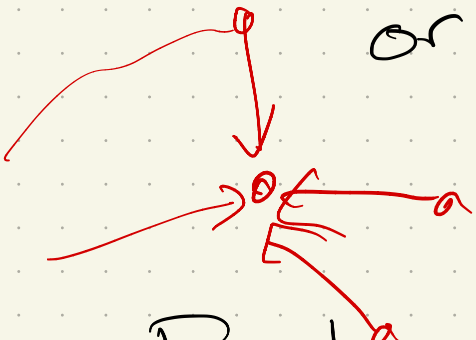


only want smallest

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



$d^+(v)$ edges out

Runtime:

- at most E ChangeKey operations in heap O
- at most E inserts/removes

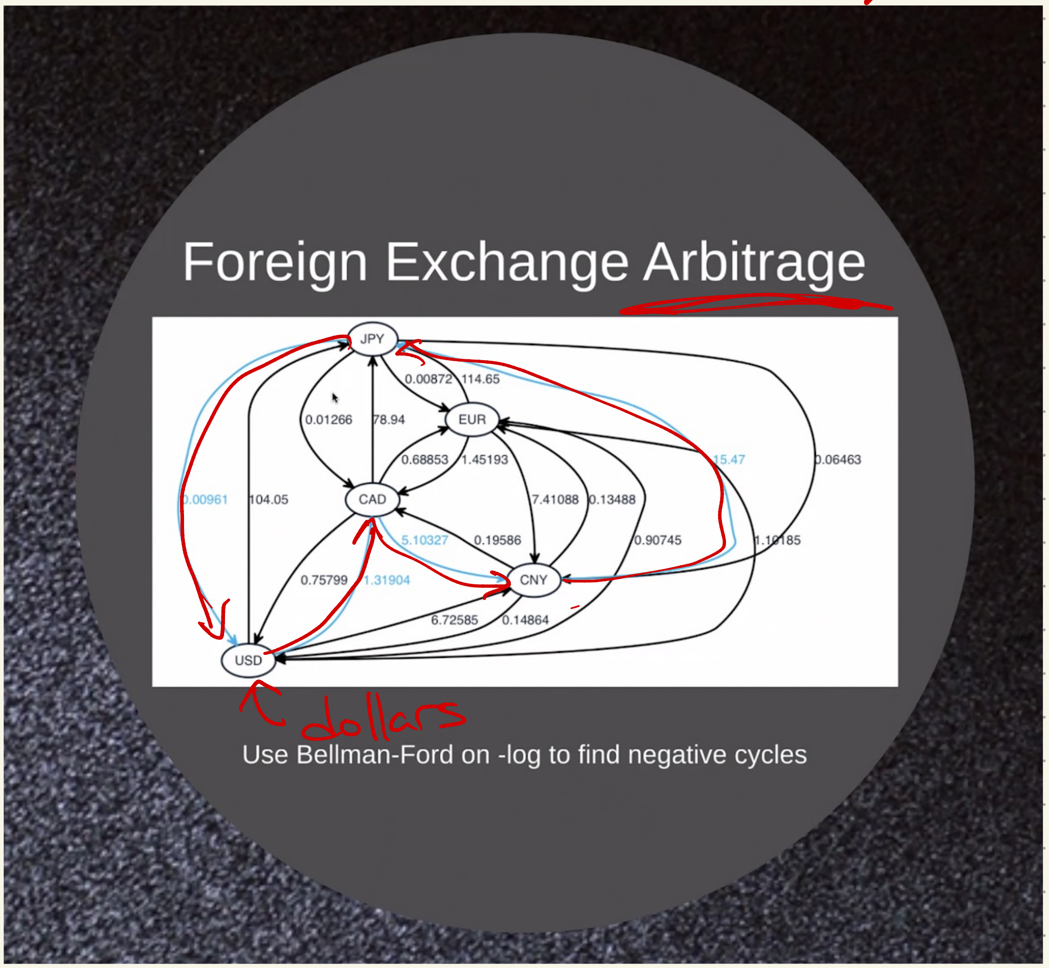
in a heap

Total: $O(E \log V)$

(no negative edges)

Negative edges:
Can happen!

Example: 2 weeks ago in colloquium

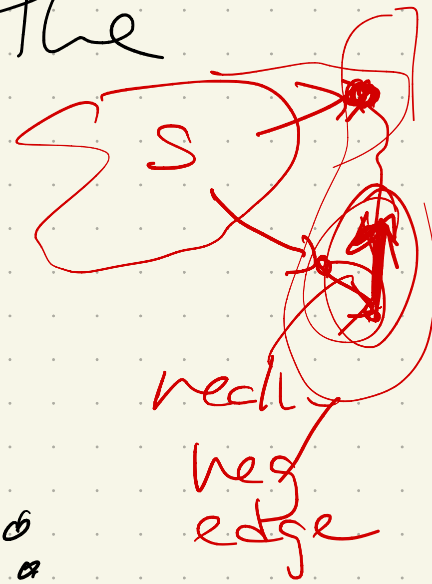


in finance
(not so much in roads...)

What happens with negative edges? (in Dijkstra)

Well, for one thing, the induction breaks!

How bad is it?

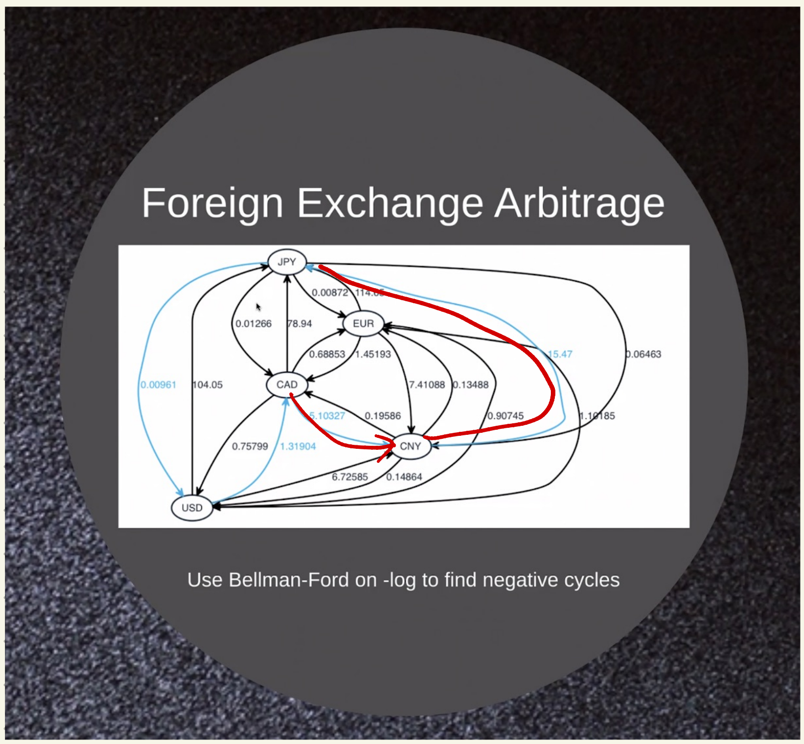


If negative cycles:
completely fails → runs forever!

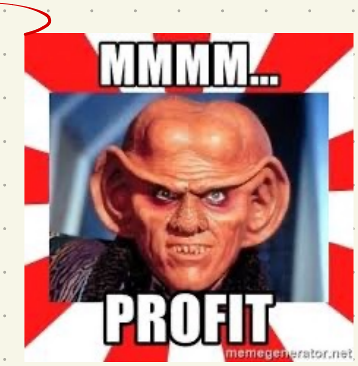
If negative edges but no negative cycles:
exponential but will stop eventually

What do we do with negative cycles?

Note: sometimes the entire goal is to find these!



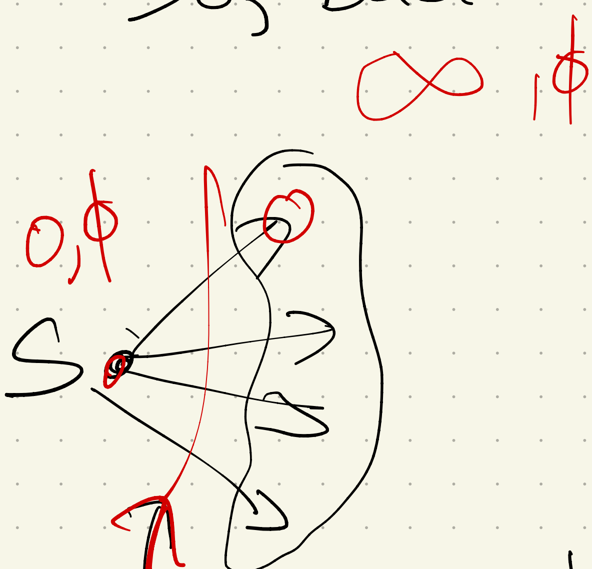
Here, cycle = profit!



Well, can't be "greedy" + take the minimum each time.

Might repeat, so need to
- compute SP-tree
- or find a negative cycle

So, back to BFS style.




```
BELLMANFORD(s)
  INITSSSP(s)
  while there is at least one tense edge
     for every edge u→v
        if u→v is tense
           RELAX(u→v)
```

while loop

all tense initially, so relax them all.

Repeat - how long?? (problem)

if neg cycle, repeat forever

Result: how long can a path be? (shortcut infinite loop)

```
BellmanFord(s)
    InitSSSP(s)
    repeat V − 1 times
        for every edge u→v
            if u→v is tense
                Relax(u→v)
    for every edge u→v
        if u→v is tense
            return "Negative cycle!"
```

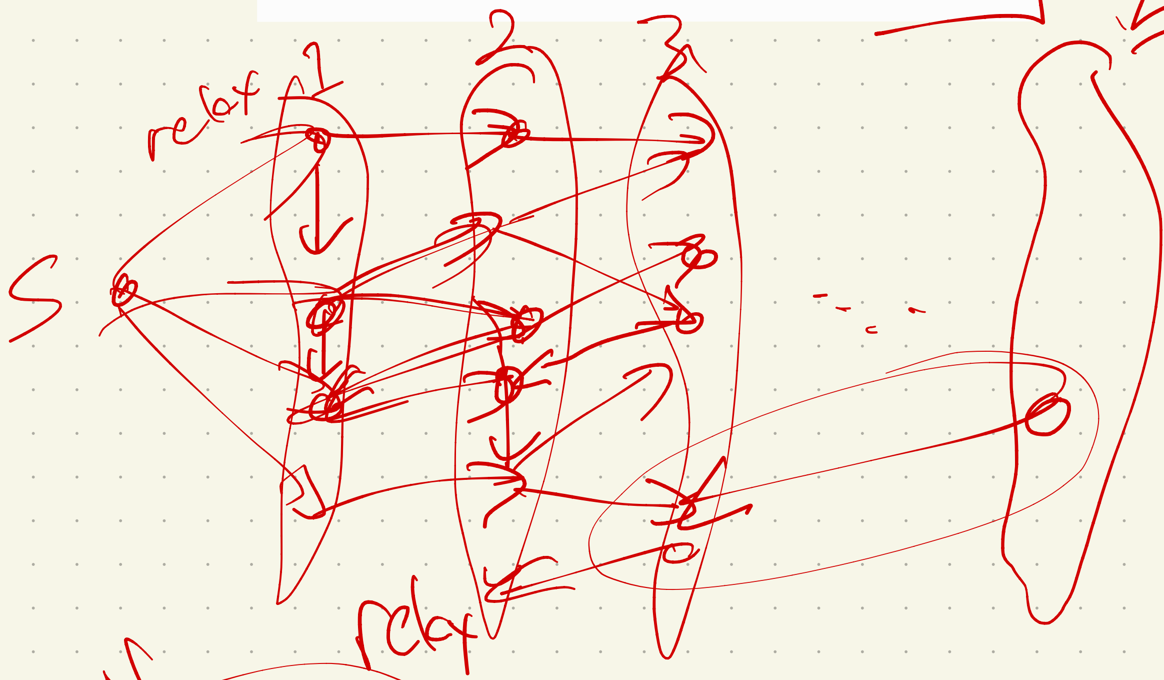
$O(V)$

not ∞ → v times

relax all $O(E)$

$O(E)$

this is on cycle! $\leq V-1$



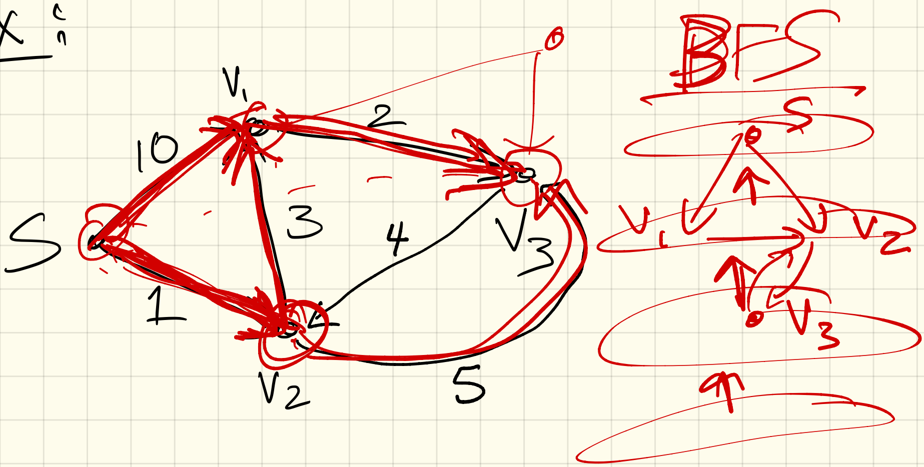
tense anywhere!

Runtime: $V-1$ rounds guarantees every vertex + edge have been "hit" $\geq$ once $O(VE)$

Notation:

Let $\boxed{\text{dist}_{\le i}(v)} :=$ length of the shortest s-to-v path using at most i edges

Ex:



S:

$d_{\le 0}(s) = 0$

$d_{\le 1}(s) = 1$

⋮

all 0

V_1:

$d_{\le 0}(v_1) = \infty$

$d_{\le 1}(v_1) = 10$

$d_{\le 2}(v_1) = 4$

$d_{\le 3}(v_1) = 4$

V_2:

$d_{\le 0}(v_2) = \infty$

$d_{\le 1}(v_2) = 1$

$d_{\le 2}(v_2) = 1$

$d_{\le 3}(v_2) = 1$

V_3:

$d_{\le 0}(v_3) = \infty$

$d_{\le 1}(v_3) = \infty$

$d_{\le 2}(v_3) = 6$

$d_{\le 3}(v_3) = 6$

So: $\mathrm{dist}_{\le 0}(s) = 0$ & for all $v \neq s$, $\mathrm{dist}_{\le 0}(v) = \infty$ BC - Init SSSP

Claim: $\forall v$ & i, after i iterations of B-F, $\mathrm{dist}(v) \le \mathrm{dist}_{\le i}(v)$

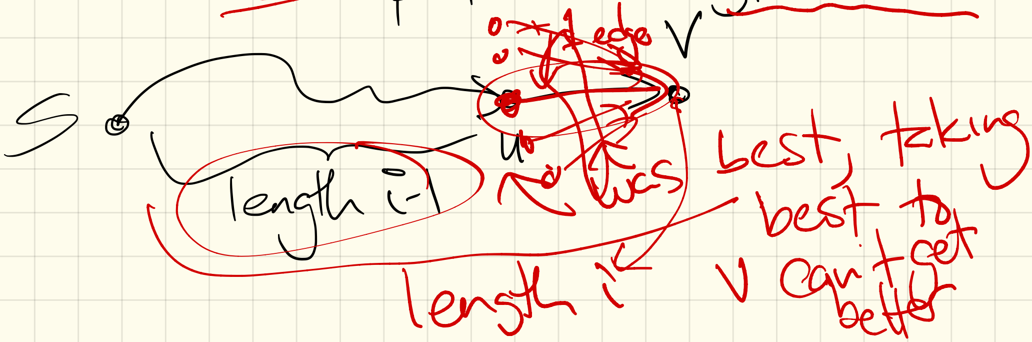
Why?

Induction on i:

BC: $d_{\le 0}(s)$ & $d_{\le 0}(v)$ are good

IH: $d_{\le i-1}(v)$ is $\ge \mathrm{dist}(v)$

IS: $d_{\le i}(v)$: Take shortest path of length i to v:



Either $\underline{u \to v \text{ is tense}}$:

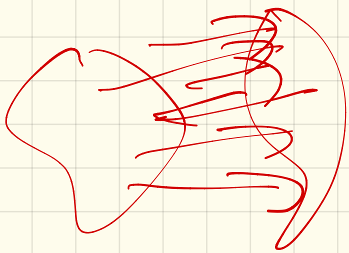
$$dist(v) > dist_{\le i-1}(u) + w(u \to v)$$

→ so relax: better v guess

$$d(v) \leftarrow dist_{\le i-1}(u) + w(u \to v)$$

OR: not

$$dist(v) < \underbrace{dist_{\le i-1}(u) + w(u \to v)}_{\text{worse, so keep old } i-1 \text{ path}}$$

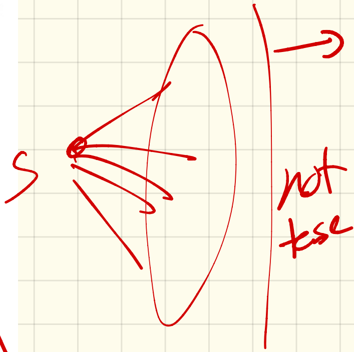


Takeaway

Since any path has length $\leq V-1$, don't need to repeat more than that!

```
BellmanFord(s)
    InitSSSP(s)
    repeat V − 1 times
        for every edge u→v
            if u→v is tense
                Relax(u→v)
    for every edge u→v
        if u→v is tense
            return "Negative cycle!"
```

look at every edge



Runtime: $O(VE)$

Why is B-F in practice slower?

Dijkstra $O(E \log V)$ cool data structure: 1 edge

B-F : $O(EV)$ looks at all edges

really 3 possibilities:

- G has positive weights
  use Dijkstra

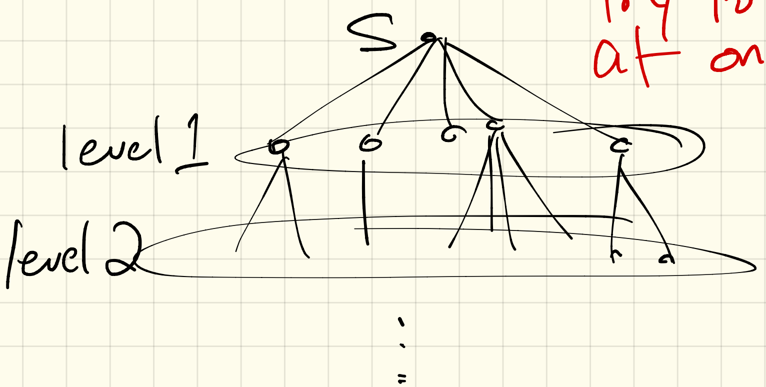
- G has negative edges
  how many / how big?
  use B-F (Dijkstra works but is slow)

- G has negative cycles.
  use B-F

The rest: an (in practice) speed-up

Think of a BFS tree:

try to look at only tense edges




```
MOORE(s):
  INITSSSP(s)
  PUSH(s)
  PUSH(✠)                         ⟨⟨start the first phase⟩⟩
  while the queue contains at least one vertex
      u ← PULL()
      if u = ✠
          PUSH(✠)                 ⟨⟨start the next phase⟩⟩
      else
          for all edges u→v
              if u→v is tense
                  RELAX(u→v)
                  if v is not already in the queue
                      PUSH(v)
```

queue:

s ✠

more next time

Final version: Bellman's!

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

Why??

Using i again as # of edges in the path!

Since all paths are $\leq V-1$,

$dist_{V-1}(v)$ <u>is</u> $dist(v)$

(assuming no negative cycles)

Nicer:

```
BELLMANFORDDP(s)
  dist[0, s] ← 0
  for every vertex v ≠ s
      dist[0, v] ← ∞
  for i ← 1 to V − 1
      for every vertex v
          dist[i, v] ← dist[i − 1, v]
          for every edge u→v
              if dist[i, v] > dist[i − 1, u] + w(u→v)
                  dist[i, v] ← dist[i − 1, u] + w(u→v)
```

Later observation:

Really don't need the i.

Just update those "tentative" distances, & trust it'll halt.

```
BELLMANFORDFINAL(s)
  dist[s] ← 0
  for every vertex v ≠ s
      dist[v] ← ∞
  for i ← 1 to V − 1
      for every edge u→v
          if dist[v] > dist[u] + w(u→v)
              dist[v] ← dist[u] + w(u→v)
```

Same runtime as prior BF
(just a bit more confusing!)