15 NOV 2021

immediately:

$$\boxed{dist(u,v,\ell) = \begin{cases} 0 & \text{if } \ell = 0 \text{ and } u = v \\ \infty & \text{if } \ell = 0 \text{ and } u \neq v \\ \min\left\{ \begin{array}{c} dist(u,v,\ell-1) \\ \min\limits_{x\to v}\left(dist(u,x,\ell-1) + w(x\to v)\right) \end{array} \right\} & \text{otherwise} \end{cases}}$$

recursive formula

Turning this recurrence into a dynamic programming algorithm is straightforward; the resulting algorithm runs in $O(V^2E) = O(V^4)$ time.

For each ℓ, we have a 2D array. $\ell = 0$ is base case. $\forall \ell$, as long as we have the table for $\ell - 1$, we can compute the table for ℓ.

Alg 1:

```
SHIMBELAPSP(V, E, w):
  for all vertices u
      for all vertices v
          if u = v
              dist[u, v, 0] ← 0
          else
              dist[u, v, 0] ← ∞        } initially, we know no paths (of length 0)
  for ℓ ← 1 to V − 1
      for all vertices u
          for all vertices v ≠ u
              dist[u, v, ℓ] ← dist[u, v, ℓ − 1]
              for all edges x→v        → trying to have x be the last stop before v.
                  if dist[u, v, ℓ] > dist[u, x, ℓ − 1] + w(x→v)
                      dist[u, v, ℓ] ← dist[u, x, ℓ − 1] + w(x→v)
```

This algorithm was first sketched by Alfonso Shimbel in 1954.[5] Just like Bellman's formulation of Bellman-Ford, we don't need the inner loop over vertices v or the iteration index ℓ. The modified algorithm is shown below.

Alg 2:

No ℓ here!

```
ALLPAIRSBELLMANFORD(V, E, w):
  for all vertices u
      for all vertices v
          if u = v
              dist[u, v] ← 0
          else
              dist[u, v] ← ∞
  for ℓ ← 1 to V − 1        ← ℓ is used to bound how many times we make updates.
      for all vertices u
          for all edges x→v
              if dist[u, v] > dist[u, x] + w(x→v)
                  dist[u, v] ← dist[u, x] + w(x→v)
```

[5]Shimbel assumed the input was a complete $V \times V$ matrix of distances, so his original algorithm actually runs in $O(V^4)$ time no matter how many edges the graph has.

In groups: How are these algorithms working? Try it out on a small example.

alg. 1 line 11, alg. 2 line 9

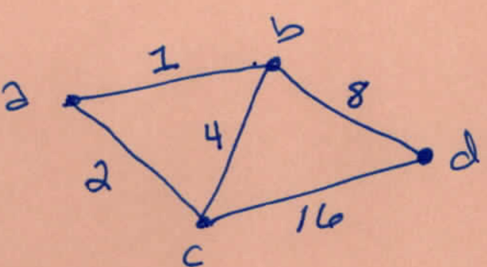
→ what exactly does this set of edges look like?

$l = 2$, $l-1 = 1$

I'm computing:

$l-1$ (1)	a	b	c	d
a	0	1	2	∞
b	1	0	4	8
c	2	4	0	16
d	∞	8	16	0

$l=2$	a	b	c	d
a		1		
b				
c				
d				



line 7: $l = 2$. I see my table for $l-1 = 1$ above (left).

line 8: $u = a$ (my options: a, b, c, d)

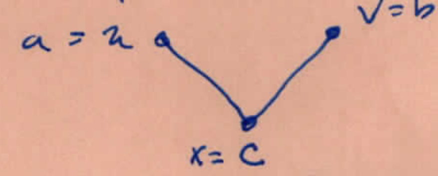
line 9: $v = b$ (my options were: b, c, d)

line 10: all edges $x \to b$: this set/my options are $\{a \to b, c \to b, d \to b\}$

lets' choose edge $c \to b$.

line 12: checking $\text{dist}[\overset{a}{u}, \overset{b}{v}, l] > \text{dist}[\overset{a}{u}, \overset{c}{x}, l-1] + w(\overset{c}{x} \to \overset{b}{v})$

$u \longrightarrow v = b$ compared to



$1 > 2 + 4$ $\boxed{\text{FALSE}}$

What is a flow?

Given: weighted directed graph $G=(V,E,c)$ ← capacity fcn.

source $s \in V$
sink $t \in V$

Def'n: A <u>(feasible) (s,t)-flow</u> in G is a fcn on the edges $f: E \to \mathbb{R}$ s.t.

① capacity constraint: $\forall e \in E$
$0 \le f(e) \le c(e)$ } ⊛

② conservation of flow: $\forall v \in V \setminus \{s,t\}$

$$\underbrace{\sum_{u \in V} f(u \to v)}_{\text{the flow into } v} = \underbrace{\sum_{w \in V} f(v \to w)}_{\text{the flow out of } v}$$

where if $a \to b \notin E$, $f(a \to b) := 0$.

⊛ w/out capacity constraint, we have a flow that is not feasible.

Def'n: If $e \in E$ and $f(e) = c(e)$, then we say that f <u>saturates</u> e.

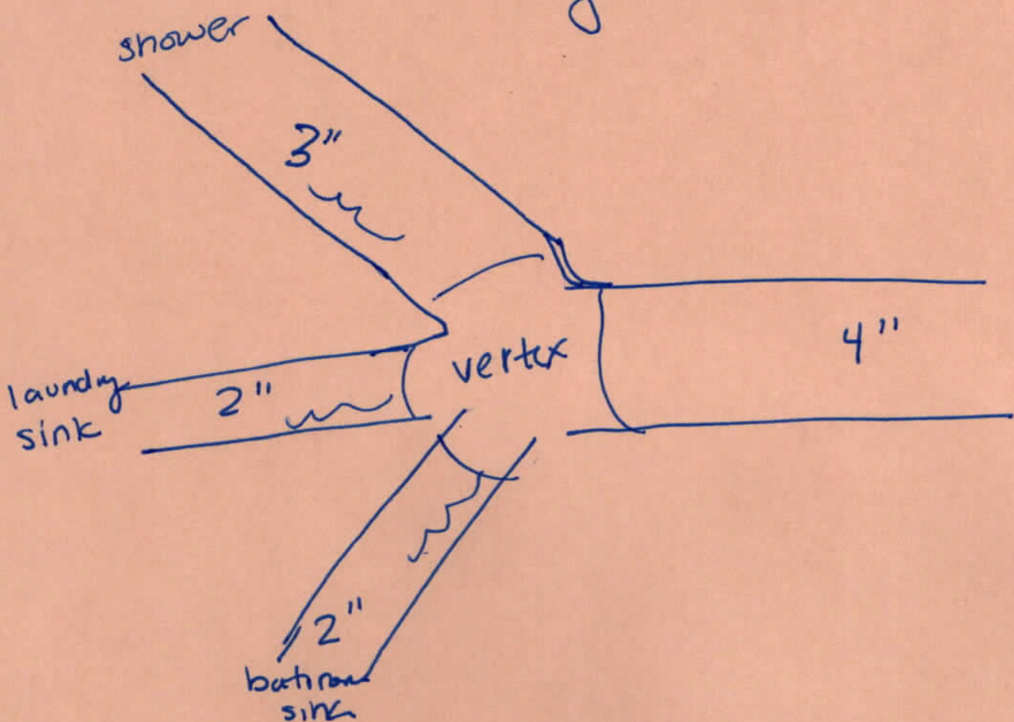
Def'n: The value of a flow $|f| := \sum_{u \in V} f(s \to u) - \sum_{w \in V} f(w \to s)$

③

Groups ① Draw a graph with at least 5 verts.

② Create at least 3 feasible flows

- identify value of flow
- identify saturated edges



* ok to send less
* not ok to send more

In computer networks, how fast can bits travel