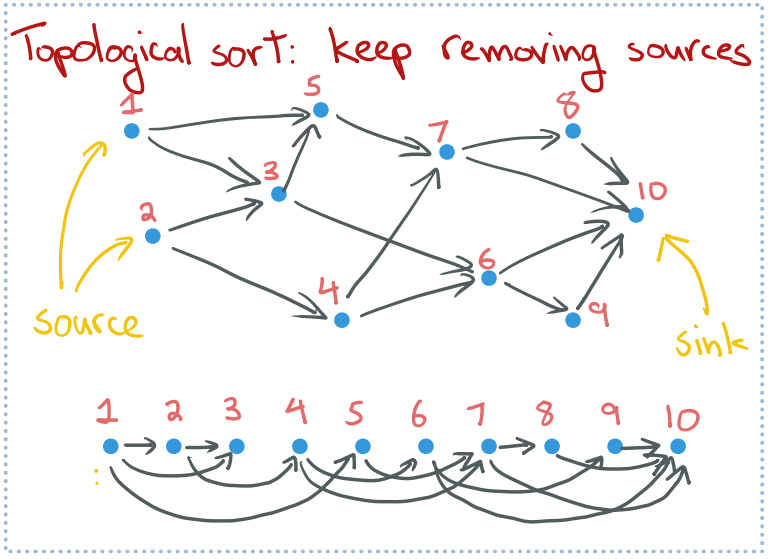
Negative Edge Lengths and All Pairs Shortest Paths

1 week ago

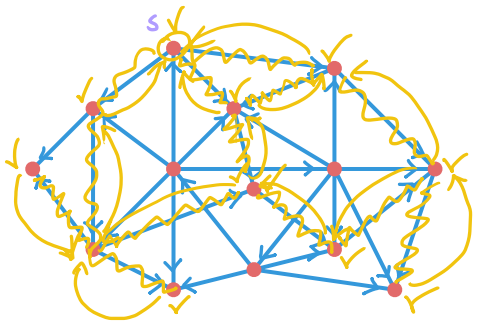
Connectivity in Directed Graphs



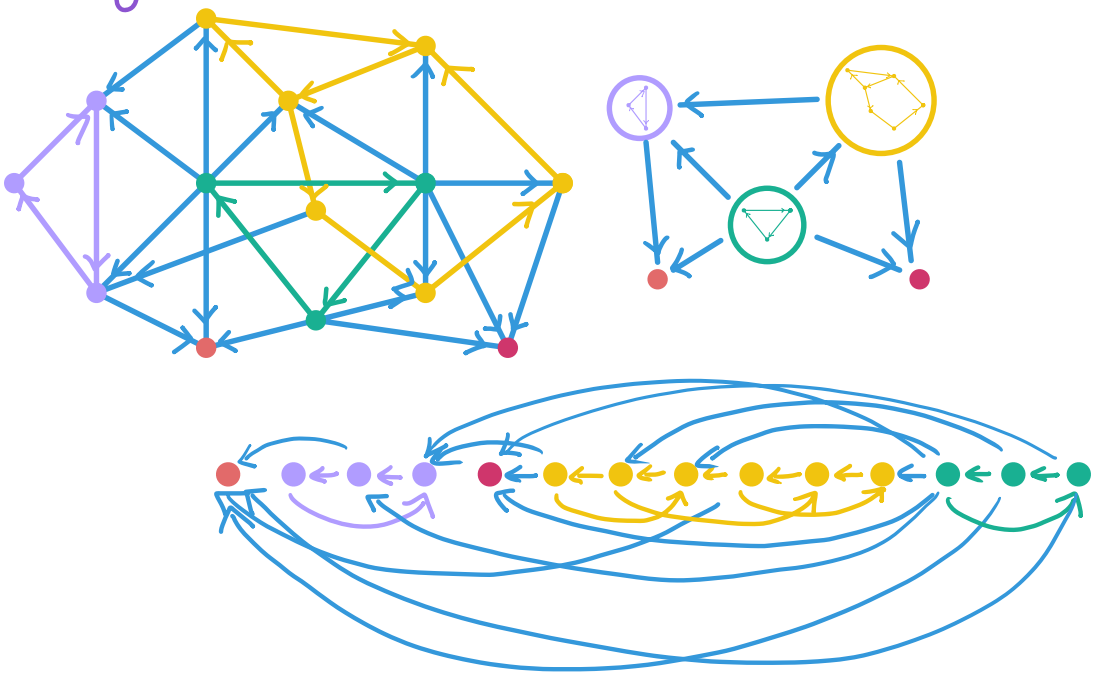
"depth-first search"

DFS(v):

① if v is unmarked
- ⓐ mark v
- ⓑ for each edge v→w
  - ① DFS(w)



strongly connected component (SCC)

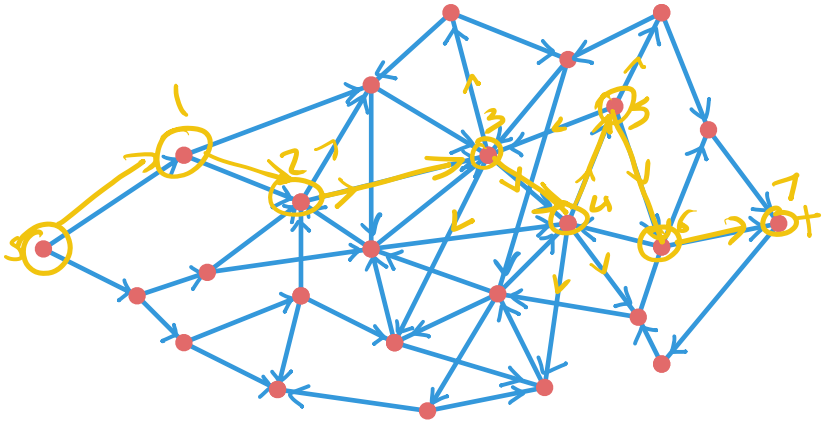


`sink-first-SCC(`$G = (V, E)$`)`

1. Compute a post-DFS ordering of V in the *reversed* graph G_r
2. Until $V = \emptyset$
   - A. Let v be the *last* (remaining) vertex in V in the post-DFS order of G_r
   - *// v is in a sink component of G.*
   - B. $C \leftarrow$ all vertices marked by `dfs(`v`)`
   - C. Append C to the output
   - D. Remove C and all incident edges from G

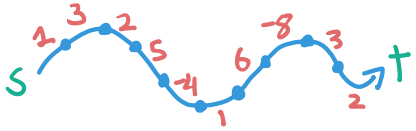
All SCC's in $O(mtn)$ time

last time: distances w/ positive weights



shortest way from s to t?

for walk w, $\bar{\ell}(w) = \sum_{e \in w} \ell(e)$



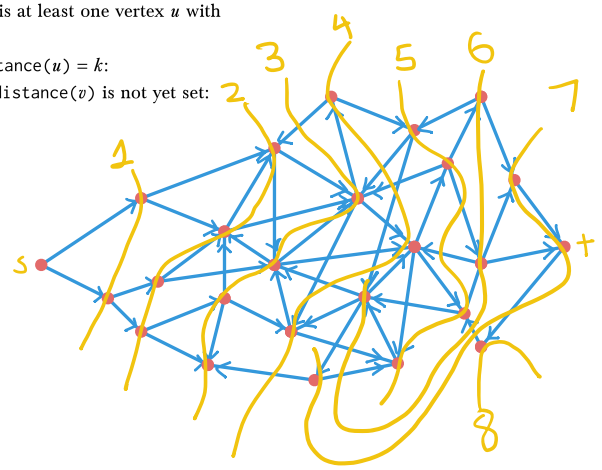
Distance $d(s,t) = \inf \{ \bar{\ell}(w) : w \text{ is an } (s,t)\text{-walk} \}$

(infimum)
(greatest lower bound)

- $d(s,t) = +\infty$ if s cannot reach t
- $d(s,t) = -\infty$ if there are arbitrarily negative length (s,t)-walks

breadth-first-search (BFS)$(G = (V,E),\ s \in V)$

1. Set distance$(s) = 0$.
2. For $k = 0,1,2,\ldots,$ as long as there is at least one vertex u with distance$(u) = k$:
   A. For all vertices u such that distance$(u) = k$:
      1. For all arcs (u,v) for which distance(v) is not yet set:
         a. set $d(v) = k+1$.



O(mtn) time

(w/ simple bookkeeping)

Dijkstra$(G = (V,E),\ \ell : E \to \mathbb{R}_{>0},\ s \in V)$

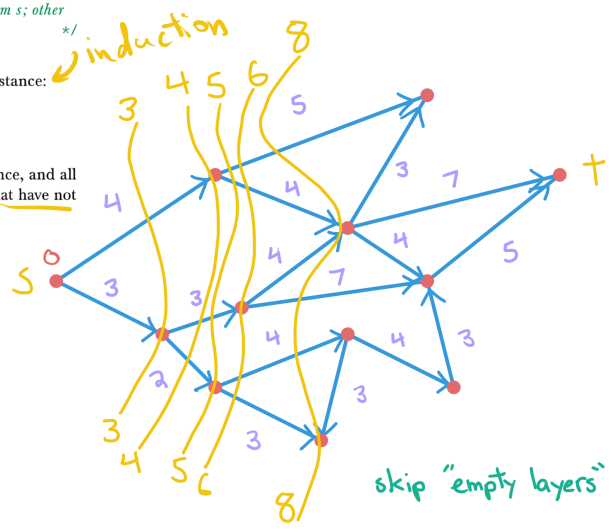
/* *We assume for simplicity that all vertices are reachable from s; other vertices would be ignored.* */

1. Set distance$(s) = 0$.
2. Until all vertices (reachable from s) are assigned a distance:
   A. Let $u, v \in V$ minimize

      $$\text{distance}(u) + \ell(u,v)$$

      over all vertices u that have been assigned a distance, and all edges (u,v) leaving u, restricting to endpoints v that have not yet been assigned a distance.
   B. Set distance$(v) =$ distance$(u) + \ell(u,v)$.
3. Return all the distance labels.

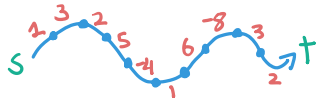
induction



skip "empty layers"

today: Negative edges

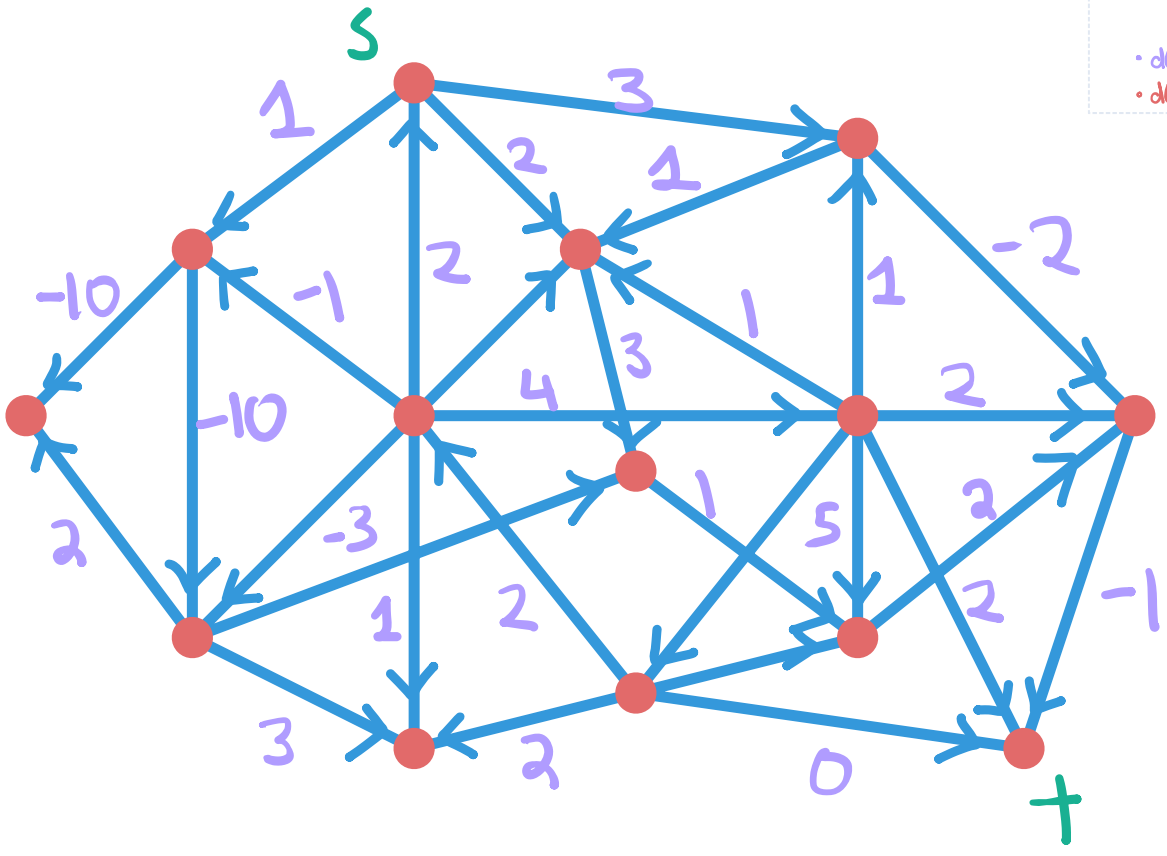
for walk w, $\ell(w) = \sum_{e \in w} \ell(e)$



Distance $d(s,t) = \inf \{ \ell(w) : w \text{ is an } (s,t)\text{-walk} \}$

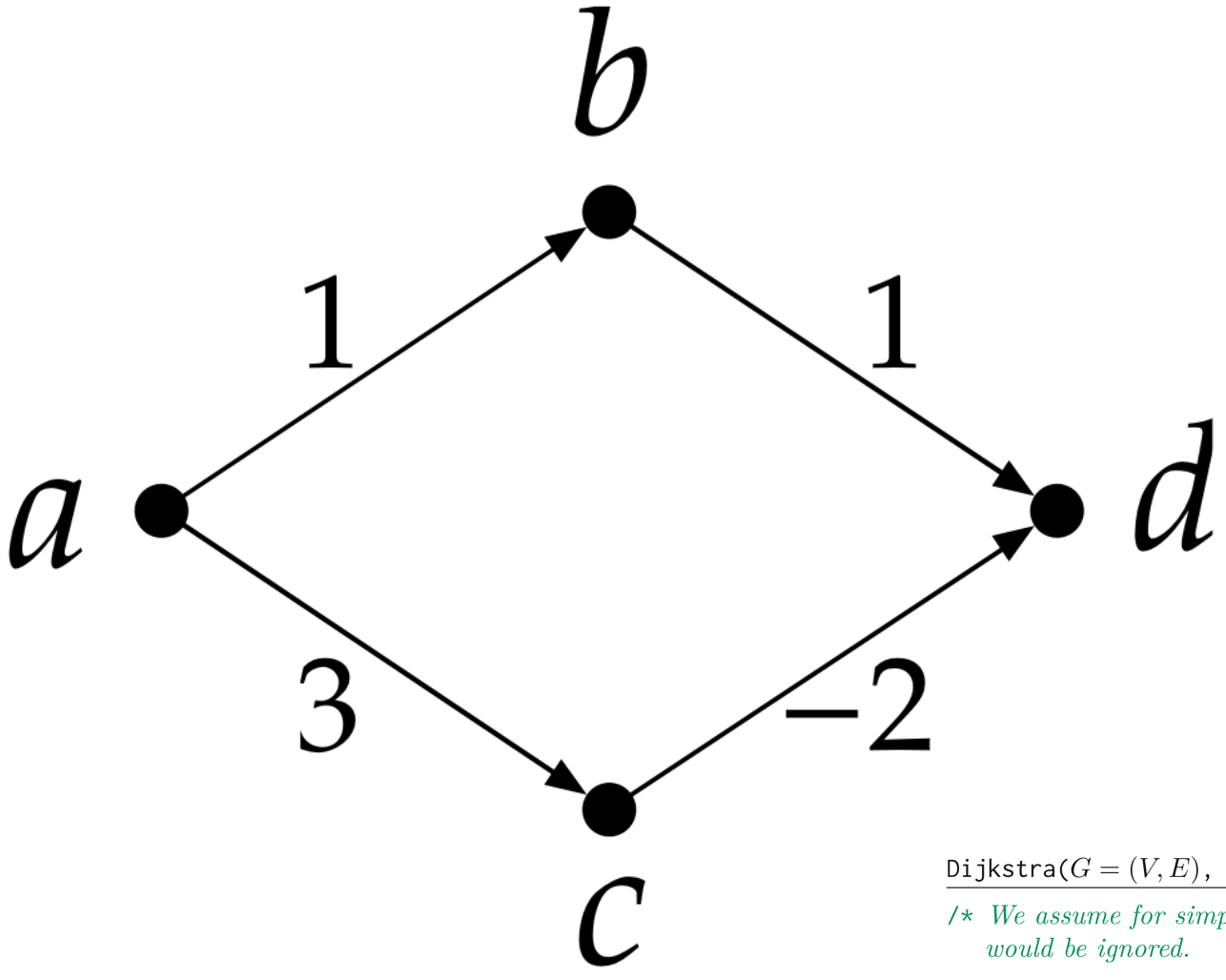
(infimum) (greatest lower bound)

- $d(s,t) = +\infty$ if s cannot reach t
- $d(s,t) = -\infty$ if there are arbitrarily negative length (s,t)-walks



distance from s to t?

Ask Dijkstra: distance from a to d?

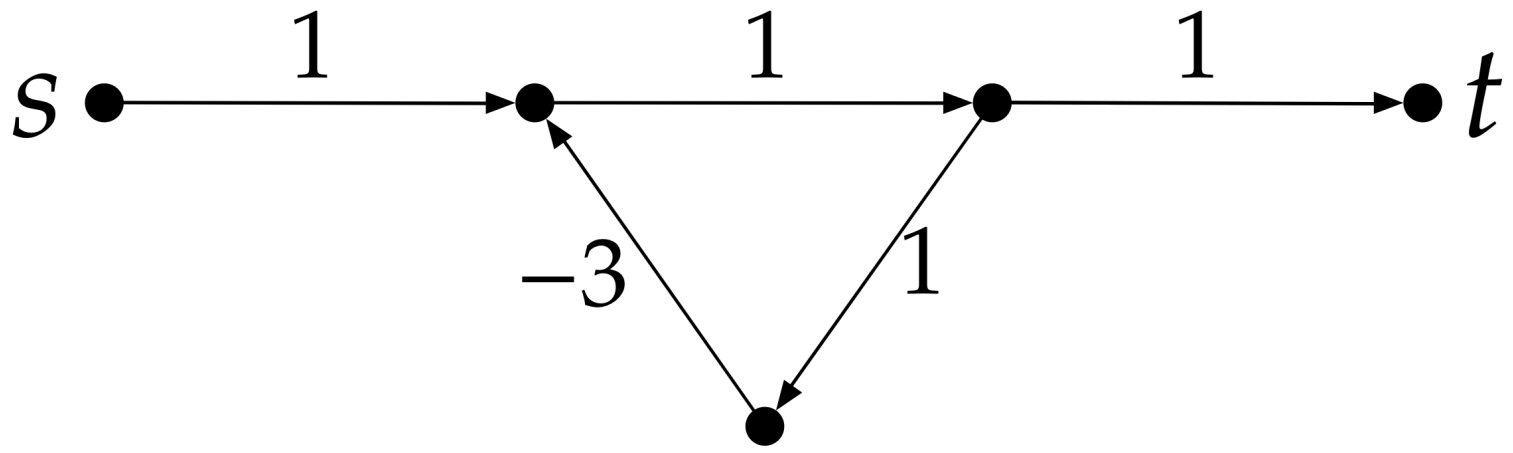


Dijkstra($G = (V, E)$, $\ell : E \to \mathbb{R}_{>0}$, $s \in V$)

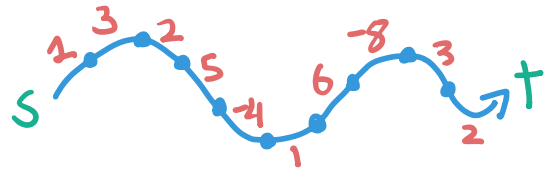
/* *We assume for simplicity that all vertices are reachable from s; other vertices would be ignored.* */

1. Set distance(s) $= 0$.
2. Until all vertices (reachable from s) are assigned a distance:
   A. Let $u, v \in V$ minimize distance(u) $+ \ell(u, v)$ over
      - all vertices u that have been assigned a distance,
      - all edges (u, v) leaving u, and
      - restricting to endpoints v that have not yet been assigned a distance.
   B. Set distance(v) $=$ distance(u) $+ \ell(u, v)$.
3. Return all the distance labels.

Negative cycles



for walk w, $\bar{\ell}(w) = \sum_{e \in w} \ell(e)$

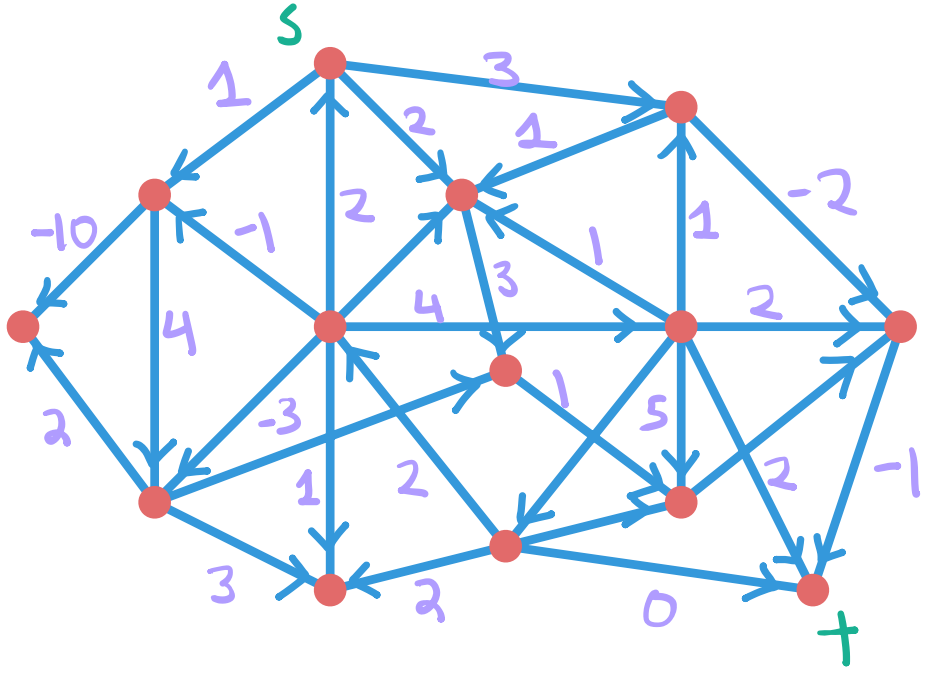


Distance $d(s,t) = \inf \{ \bar{\ell}(w) : w \text{ is an } (s,t)\text{-walk} \}$

(infimum) (greatest lower bound)

- $d(s,t) = +\infty$ if s cannot reach t
- $d(s,t) = -\infty$ if there are arbitrarily negative length (s,t)-walks

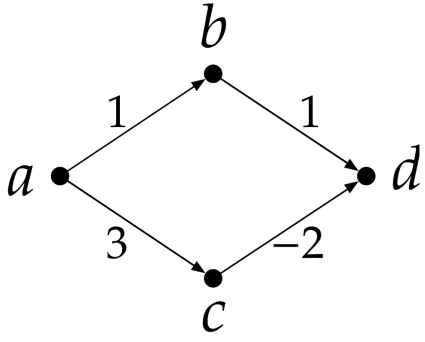
today: Negative Edges

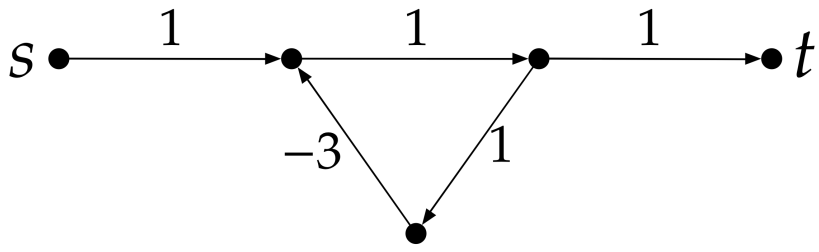


distance from s to t?

weird questions

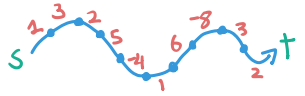
- negative cycles?
- shortest walks vs shortest paths?
- induction hypothesis?





Lemma 9.1. *Let $s, t \in V$ and suppose s can reach t. The distance from s to t is finite (and not $-\infty$) iff the shortest (s,t)-walk is attained by a path.*[1]

for walk w, $\ell(w) = \sum_{e \in w} \ell(e)$



Distance $d(s,t) = \inf \{ \ell(w) : w \text{ is an } (s,t)\text{-walk} \}$
(infimum) (greatest lower bound)

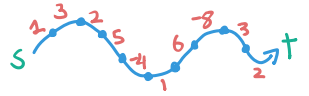
- $d(s,t) = +\infty$ if s cannot reach t
- $d(s,t) = -\infty$ if there are arbitrarily negative length (s,t)-walks

(if) suppose path $p: s \rightsquigarrow t$ w/ $d(s,t) = \ell(w)$

immediate; the path gives the distance

Lemma 9.1. *Let $s, t \in V$ and suppose s can reach t. The distance from s to t is finite (and not $-\infty$) iff the shortest (s,t)-walk is attained by a path.*[1]

for walk w, $\bar{\ell}(w) = \sum_{e \in w} \ell(e)$



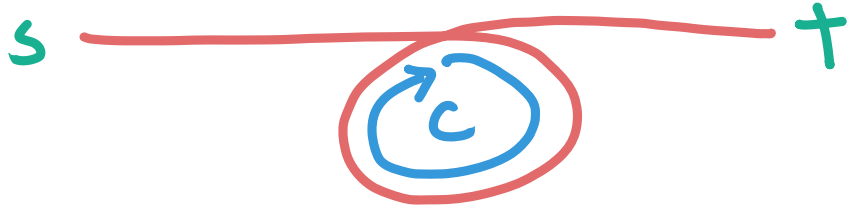
Distance $d(s,t) = \inf \{\bar{\ell}(w) : w \text{ is an } (s,t)\text{-walk}\}$
(infimum) (greatest lower bound)

- $d(s,t) = +\infty$ if s cannot reach t
- $d(s,t) = -\infty$ if there are arbitrarily negative length (s,t)-walks

(only if) suppose $d(s,t) = L$ (finite)

let w = (s,t)-walk w/ length L w/ minimum # of edges.

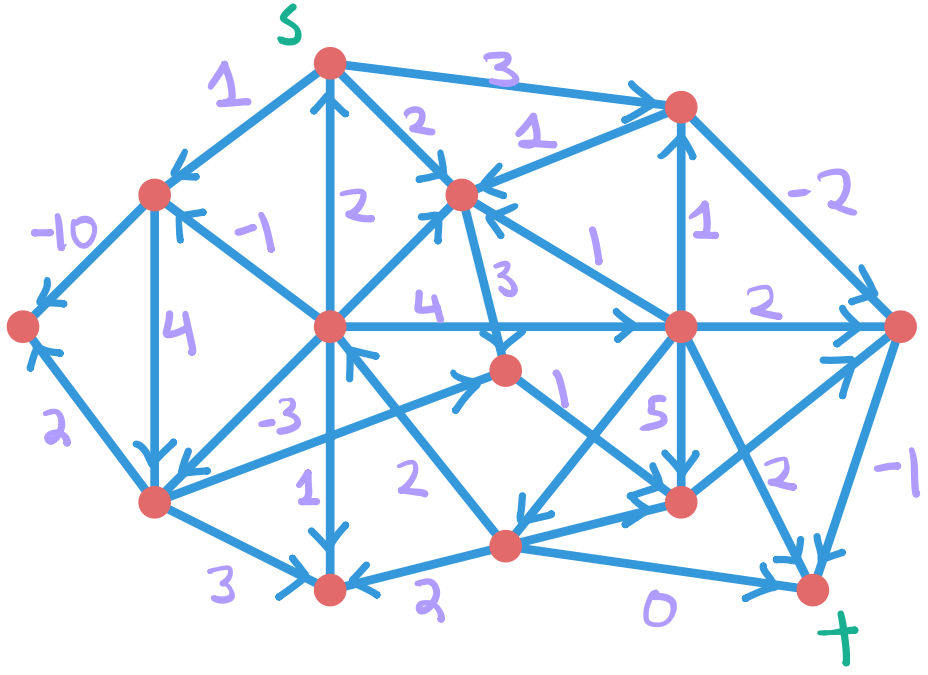
suppose w contains a cycle C



Case $\bar{\ell}(C) \geq 0$: removing cycle does not increase length, decreases # edges, ⇒⇐

Case $\bar{\ell}(C) < 0$: repeating cycle decreases length ⇒⇐

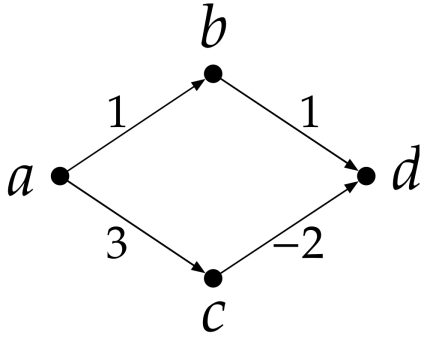
today: Negative Edges

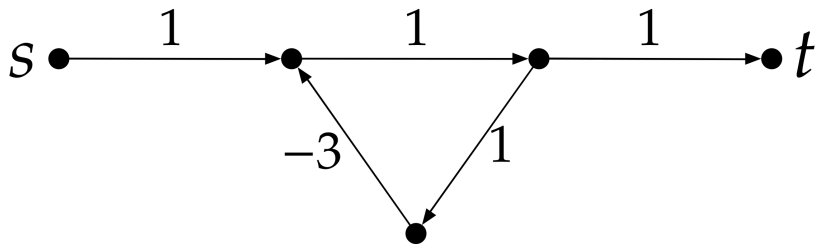


distance from s to t?

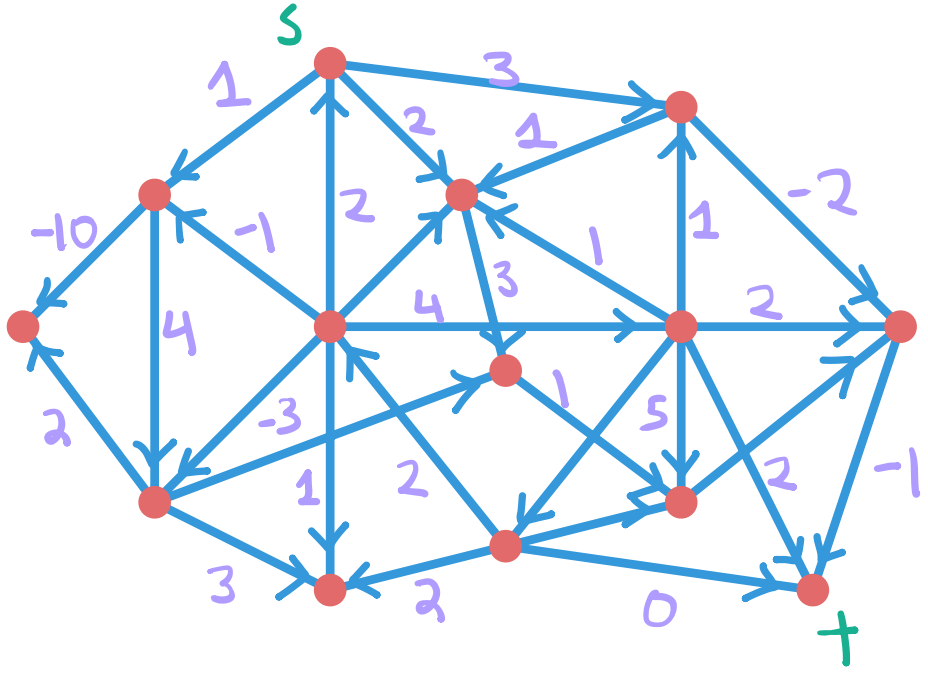
weird questions

- negative cycles?
- ✓ shortest walks vs shortest paths?
- induction hypothesis?





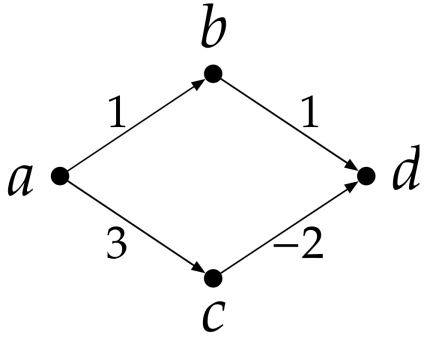
today: Negative Edges

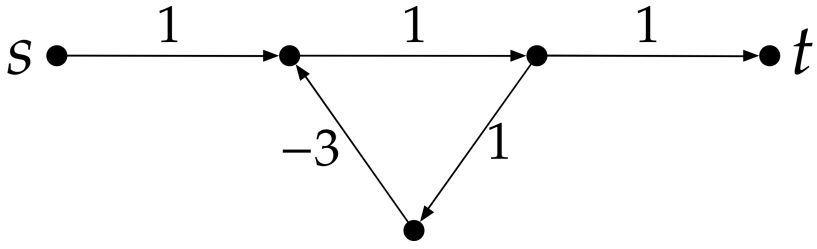


distance from s to t?

weird questions

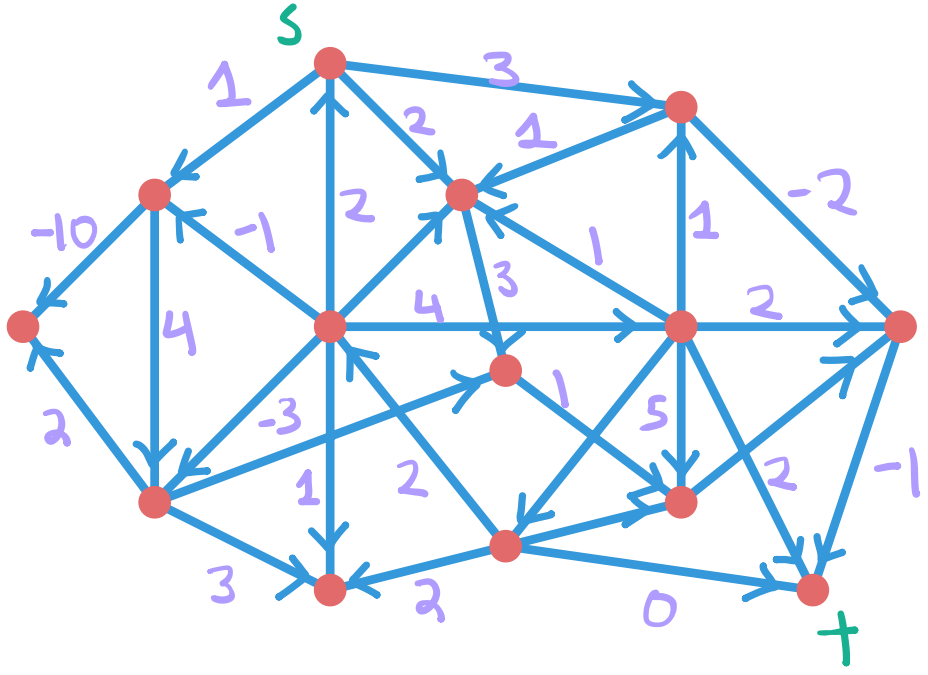
- negative cycles?
- ✓ shortest walks vs shortest paths?
- induction hypothesis?





• induction hypothesis?

today: Negative Edges

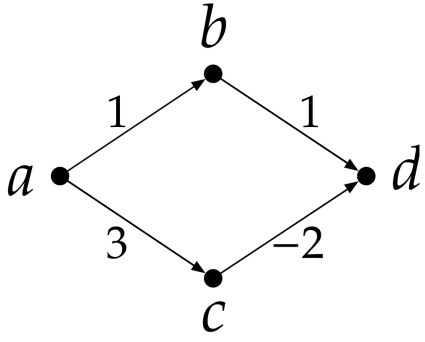


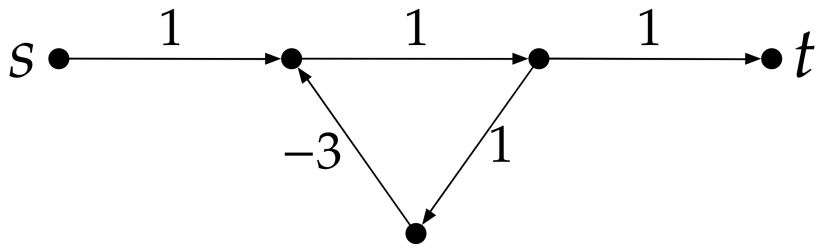
distance from s to t?

BFS, Dijkstra both establish distance to ith closest vertex before (i+1)th closest vertex

this doesn't work w/ negative edges.

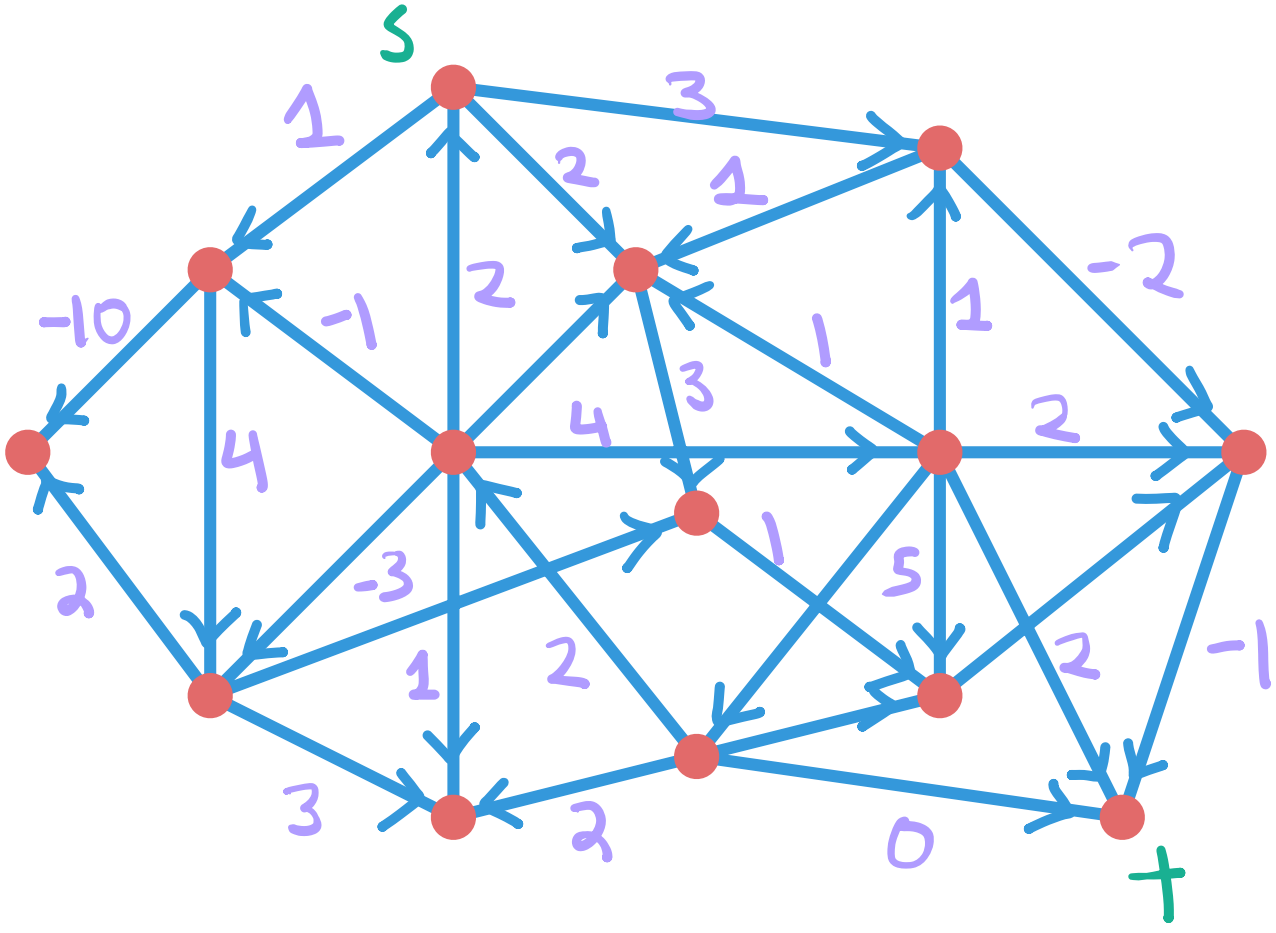
other ideas?





Shortest (s,t)-walk w/ ≤k edges

Goal: go from s to t w/ ≤k edges and min total length



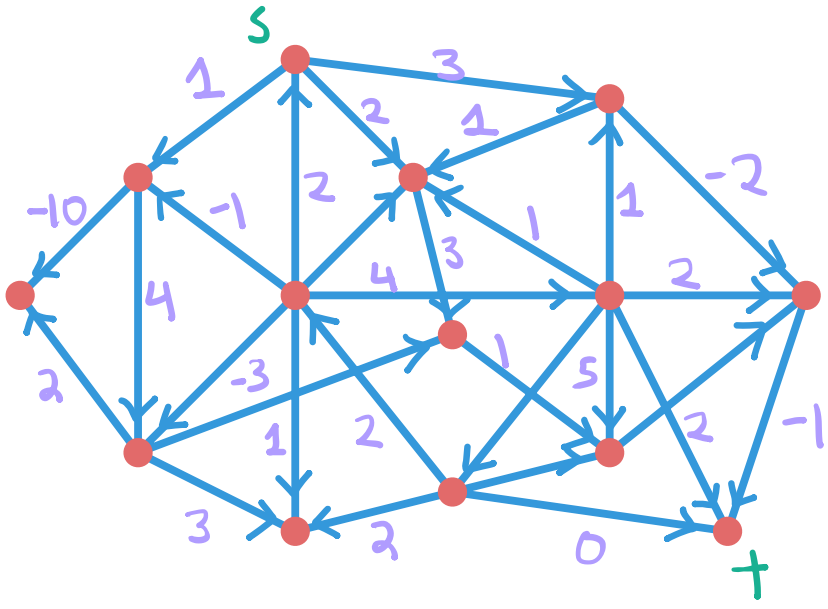
Recursive spec:

SW(v,i) = length of shortest walk from s to v w/ ≤ i edges

Shortest (s,t)-walk w/ ≤ k edges

Recursive spec:

$SW(v,i)$ = length of shortest walk from s to v w/ $\leq i$ edges



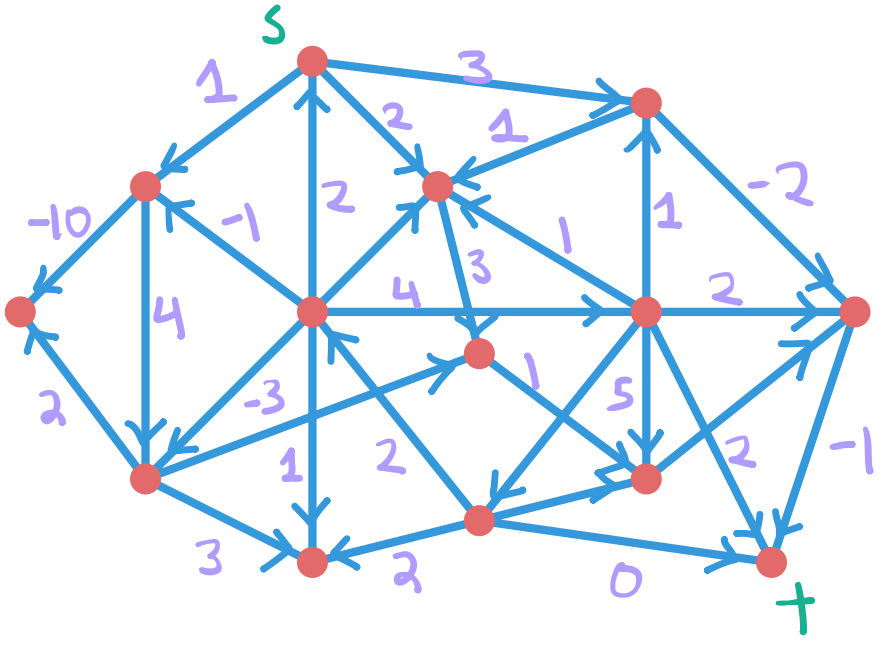
$$SW(v,i) = \begin{cases} 0 & \text{if } v=s \text{ and } i=0 \\ +\infty & \text{if } v \neq s \text{ and } i=0 \\ \min\left\{ SW(v,i-1),\ \min_{(u,v)\in\partial^-(v)} SW(u,i-1) + \ell(u,v) \right\} & \text{otherwise} \end{cases}$$

"$\partial^-(v)$" = edges entering v

Shortest (s,t)-walk w/ ≤ k edges

Recursive spec:

$SW(v,i)$ = length of shortest walk from s to v w/ $\leq i$ edges



$$SW(v,i) = \begin{cases} 0 & \text{if } v=s \text{ and } i=0 \\ +\infty & \text{if } v \neq s \text{ and } i=0 \\ \min\left\{ SW(v,i-1),\ \min_{(u,v)\in\partial^-(v)} SW(u,i-1) + \ell(u,v) \right\} & \text{otherwise} \end{cases}$$

Running time w/ caching

$$\sum_{i=1}^{k} \sum_{v\in V} (\text{time for } SW(v,i)) = k \sum_{v\in V} 1 + \deg^-(v)$$

(out-degree)

$$= O(k(m+n))$$

Lemma 9.1. *Let $s, t \in V$ and suppose s can reach t. The distance from s to t is finite (and not $-\infty$) iff the shortest (s,t)-walk is attained by a path.*[1]

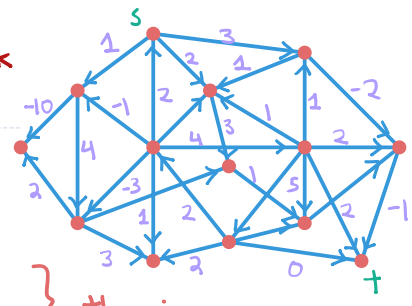
suppose no $-\infty$ distances

(no negative cycles)

Shortest (s,t)-walk w/ ≤k edges

Recursive spec:

SW(v,i) = length of shortest walk from s to v w/ ≤i edges



$$SW(v,i) = \begin{cases} 0 & \text{if } v=s \text{ and } i=0 \\ +\infty & \text{if } v \neq s \text{ and } i=0 \\ \min\left\{ SW(v,i-1),\ \min_{(u,v)\in \overleftarrow{\partial}(v)} SW(u,i-1) + \ell(u,v) \right\} & \text{otherwise} \end{cases}$$

Running time w/ caching

$$\sum_{i=1}^{k} \sum_{v \in V} (\text{time for } SW(v,i)) = k \sum_{v \in V} 1 + \overleftarrow{\deg}(v) = O(k(m+n))$$

(out-degree)

Corollary 9.3. *Let $G = (V, E)$ be a directed graph with m edges, n vertices, real-valued edge weights. Let $s \in V$ be fixed and suppose that all distances from s are finite. In $O(mn)$ time, one can compute the length of the shortest (s,t)-path for all $t \in V$.*

Lemma 9.1. *Let $s, t \in V$ and suppose s can reach t. The distance from s to t is finite (and not $-\infty$) iff the shortest (s,t)-walk is attained by a path.*[1]

let

$$A_k = \{v \in V : SW(v,k) < SW(v,k-1)\}$$

Lemma $\Rightarrow$ all $t \in A_n$ have

$$d(s,t) = -\infty$$

Better still: for all $k \geq n$,

all $t \in A_k$ have $d(s,t) = -\infty$

Q: how to identify $A_n \cup A_{n+1} \cup A_{n+2} \cup \cdots$?

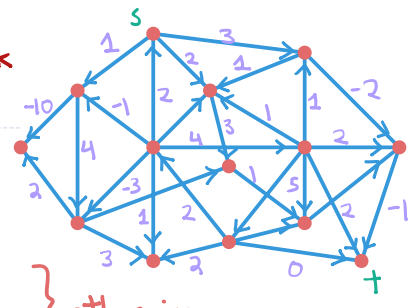
" " " all t w/ $d(s,t) = -\infty$?

Shortest (s,t)-walk w/ $\leq k$ edges

Recursive spec:

$SW(v,i)$ = length of shortest walk from s to v w/ $\leq i$ edges

$$SW(v,i) = \begin{cases} 0 & \text{if } v=s \text{ and } i=0 \\ +\infty & \text{if } v \neq s \text{ and } i=0 \\ \min\left\{ SW(v,i-1),\ \min_{(u,v) \in \vec{\partial}^-(v)} SW(u,i-1) + \ell(u,v) \right\} & \text{otherwise} \end{cases}$$



Running time w/ caching

$$\sum_{i=1}^{k} \sum_{v \in V} (\text{time for } SW(v,i)) = k \sum_{v \in V} 1 + \underbrace{\deg^-(v)}_{\text{(out-degree)}} = O(k(m+n))$$

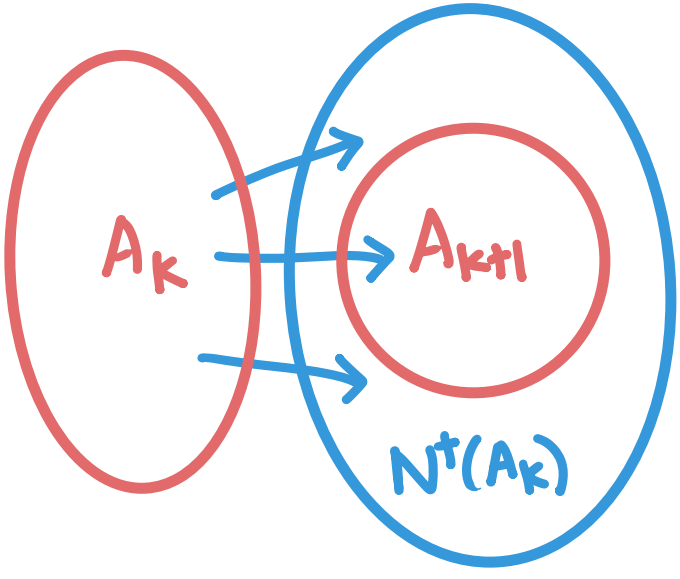
Lemma 9.1. *Let $s, t \in V$ and suppose s can reach t. The distance from s to t is finite (and not $-\infty$) iff the shortest (s,t)-walk is attained by a path.*[1]

let

$$A_k = \{v \in V : SW(v,k) < SW(v,k-1)\}$$

claim: $A_{k+1} \subseteq N^+(A_k)$

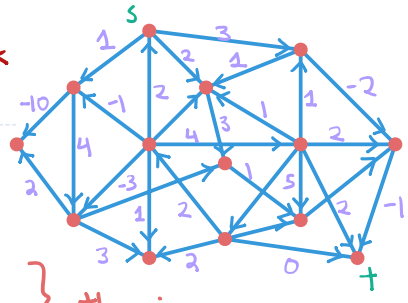
$N^+(X)$ = "out-neighbors of X"
$= \{v \in V : (x,v) \in E \text{ for some } x \in X\}$



why?

Shortest (s,t)-walk w/ ≤k edges

Recursive spec:
$SW(v,i)$ = length of shortest walk from s to v w/ $\le i$ edges



$$SW(v,i) = \begin{cases} 0 & \text{if } v=s \text{ and } i=0 \\ +\infty & \text{if } v \neq s \text{ and } i=0 \\ \min\left\{ SW(v,i-1),\ \min_{(u,v)\in \overleftarrow{E}(v)} SW(u,i-1) + \ell(u,v) \right\} & \text{otherwise} \end{cases}$$

Running time w/ caching

$$\sum_{i=1}^{k} \sum_{v \in V} (\text{time for } SW(v,i)) = k \sum_{v \in V} 1 + \overleftarrow{\deg}(v) = O(k(m+n))$$

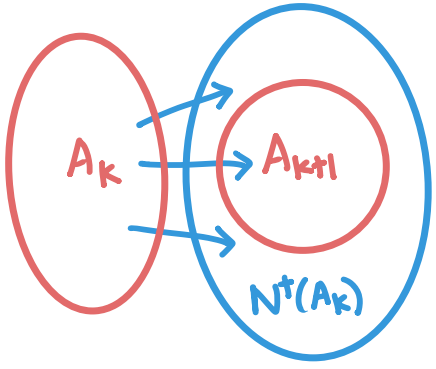
(out-degree)

Lemma 9.1. *Let $s, t \in V$ and suppose s can reach t. The distance from s to t is finite (and not $-\infty$) iff the shortest (s,t)-walk is attained by a path.*[1]

- $A_k = \{v \in V : SW(v,k) < SW(v,k-1)\}$

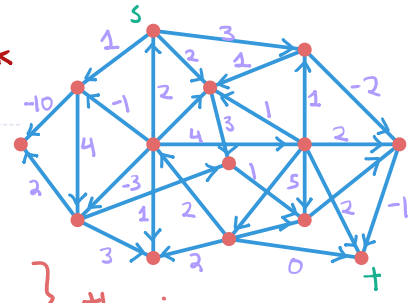
✓ $A_{k+1} \subseteq N^+(A_k)$

$N^+(X)$ = "out-neighbors of X"
$= \{v \in V : (x,v) \in E \text{ for some } x \in X\}$



Shortest (s,t)-walk w/ $\le k$ edges

Recursive spec:
$SW(v,i)$ = length of shortest walk from s to v w/ $\le i$ edges



$SW(v,i) =$
- 0 if $v = s$ and $i = 0$
- $+\infty$ if $v \neq s$ and $i = 0$
- $\min\{SW(v,i-1), \min_{(u,v) \in \vec{\partial}^-(v)} SW(u,i-1) + \ell(u,v)\}$ otherwise

Running time w/ caching

$$\sum_{i=1}^{k} \sum_{v \in V} (\text{time for } SW(v,i)) = k \sum_{v \in V} 1 + \deg^-(v) = O(k(m+n))$$

(out-degree)

$A_{n+1} \subseteq N^+(A_n)$

$A_{n+2} \subseteq N^+(N^+(A_n))$

$A_{n+3} \subseteq N^+(N^+(N^+(A_n)))$

$A_{n+4} \subseteq N^+(N^+(N^+(N^+(A_n))))$

$\vdots$

$\Rightarrow$ if $t \in A_k$ for some $k \ge n$, then t is reachable from A_n

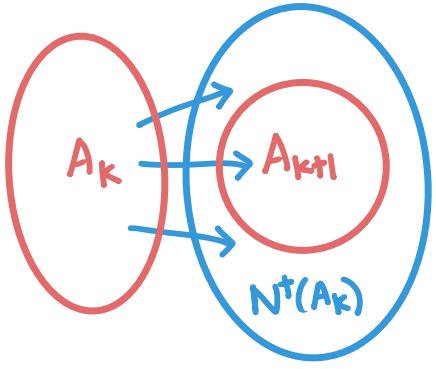
Conversely, if t reachable from A_n, then $d(s,t) = -\infty$

Lemma 9.1. *Let $s, t \in V$ and suppose s can reach t. The distance from s to t is finite (and not $-\infty$) iff the shortest (s,t)-walk is attained by a path.*[1]

- $A_k = \{v \in V : SW(v,k) < SW(v,k-1)\}$

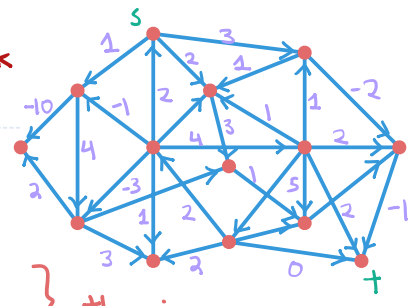
✓ $A_{k+1} \subseteq N^+(A_k)$

$N^+(X)$ = "out-neighbors of X"
$= \{v \in V : (x,v) \in E \text{ for some } x \in X\}$



Shortest (s,t)-walk w/ ≤k edges

Recursive spec:
$SW(v,i)$ = length of shortest walk from s to v w/ $\leq i$ edges



$SW(v,i) =$
- 0 if $v=s$ and $i=0$
- $+\infty$ if $v \neq s$ and $i=0$
- $\min\{SW(v,i-1), \min_{(u,v)\in\vec{\partial}^-(v)} SW(u,i-1) + \ell(u,v)\}$ otherwise

Running time w/ caching

$$\sum_{i=1}^{k} \sum_{v \in V} (\text{time for } SW(v,i)) = k \sum_{v \in V} (\text{out-degree}) + \deg^-(v) = O(k(m+n))$$

All put together:

$d(s,t) = -\infty$

$\Leftrightarrow$

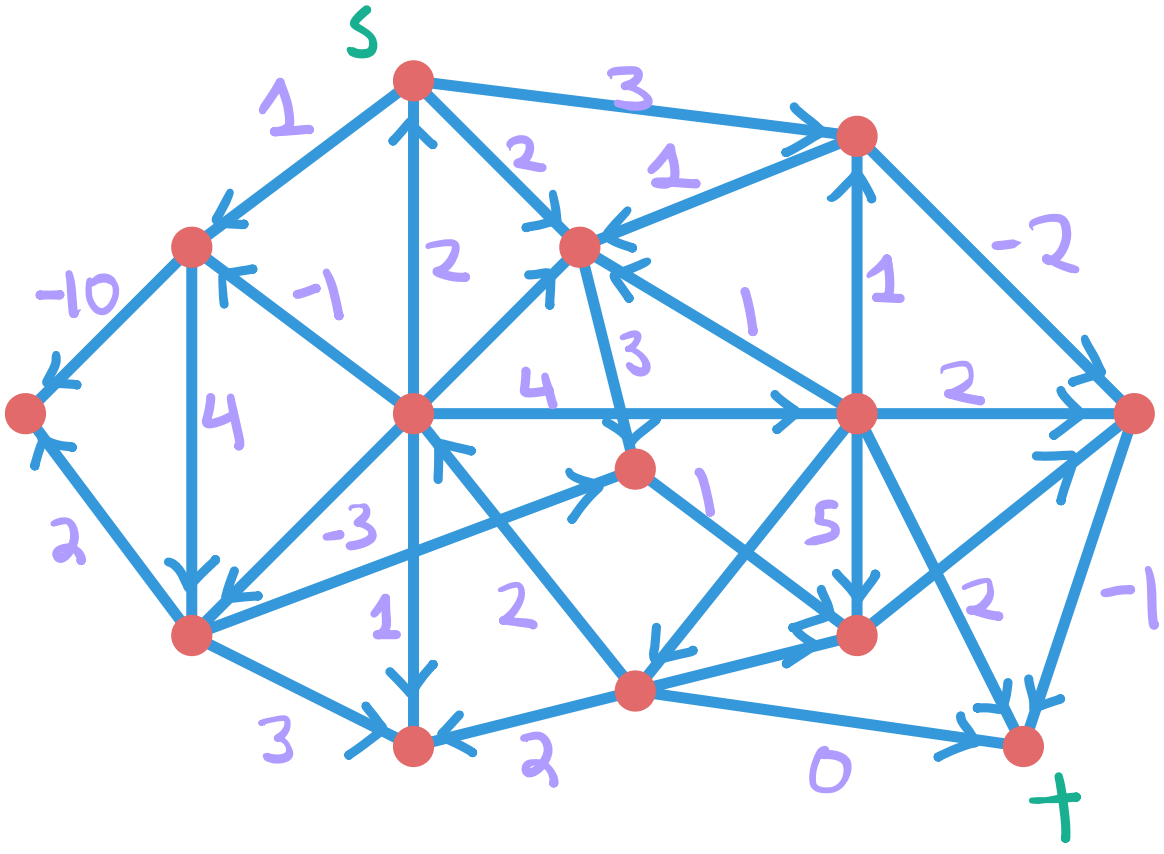
t is reachable from A_n

$A_{n+1} \subseteq N^+(A_n)$
$A_{n+2} \subseteq N^+(N^+(A_n))$
$A_{n+3} \subseteq N^+(N^+(N^+(A_n)))$
$A_{n+4} \subseteq N^+(N^+(N^+(N^+(A_n))))$
$\vdots$

$\Rightarrow$ if $t \in A_k$ for some $k \geq n$, then t is reachable from A_n

Conversely, if t reachable from A_n, then $d(s,t) = -\infty$

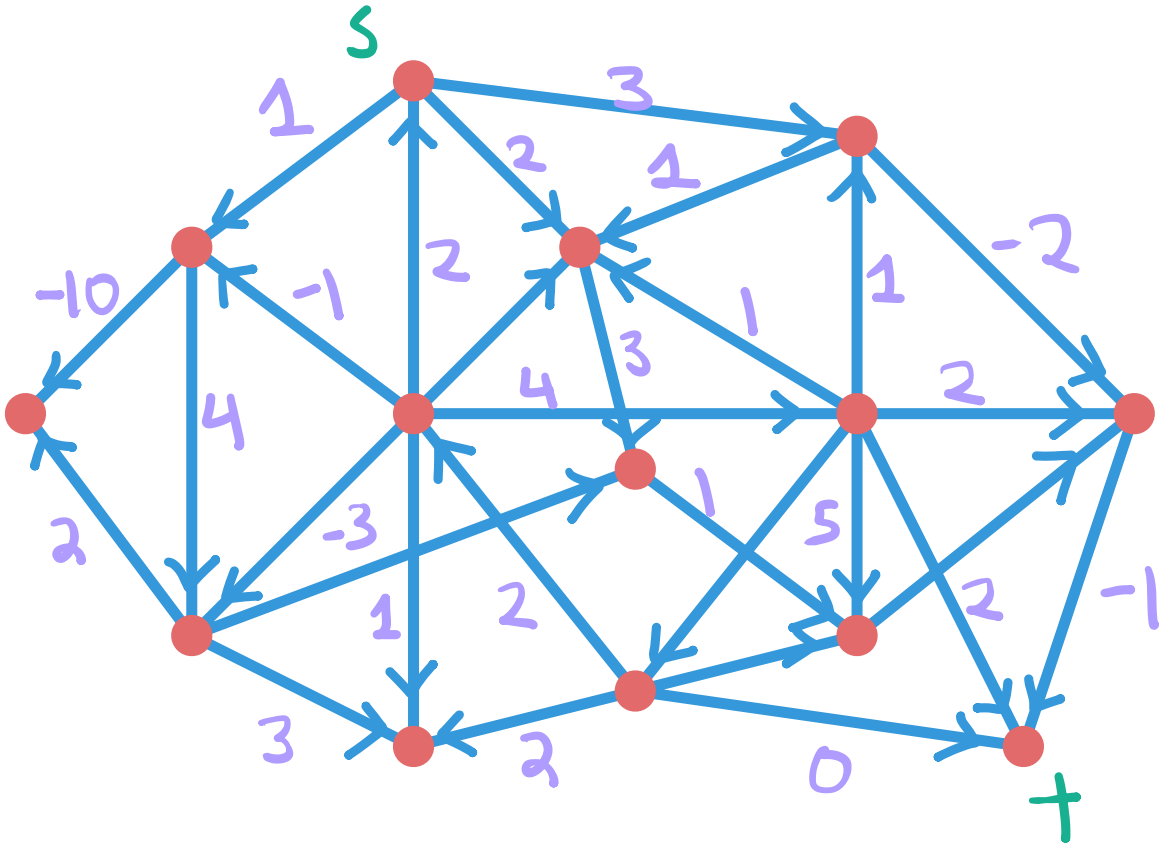
All pairs shortest paths



Goal: compute d(s,t) for <u>all</u> s,t

assume no negative cycles (for simplicity)

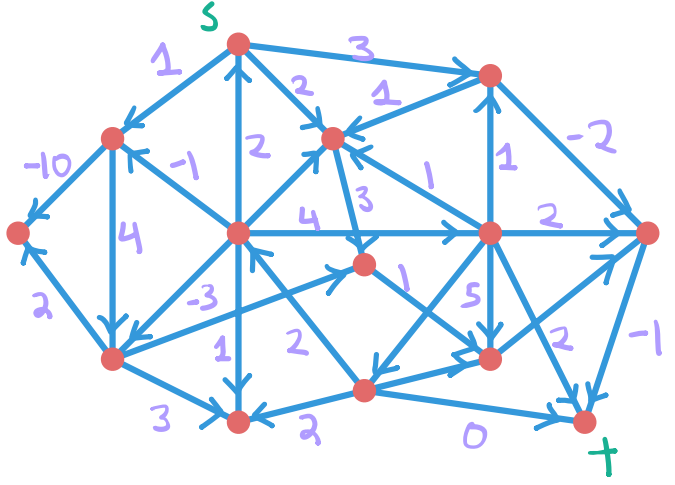
All pairs shortest paths



Goal: compute d(s,t) for <u>all</u> s,t

assume no negative cycles (for simplicity)

All pairs shortest paths



Goal: compute d(s,t) for all s,t
assume no negative cycles (for simplicity)

Recursive spec:

SW(s,t,i) = length of shortest walk from s to t w/ $\leq i$ edges

(essentially same code)

SW(s,t,i) =

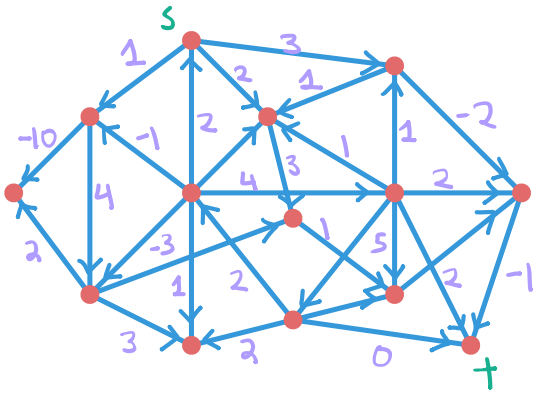
0 if s=t and i=0

$+\infty$ if $s \neq t$ and i=0

$$\min \begin{cases} SW(s,t,i-1) \\ \min_{(u,t)\in\partial^-(t)} SW(s,u,i-1) + \ell(u,t) \end{cases} \text{ otherwise}$$

Running time (w/ caching) $O(mn^2)$ better?

All pairs shortest paths



Goal: compute d(s,t) for all s,t
assume no negative cycles (for simplicity)

Recursive spec:

$SW(s,t,i)$ = length of shortest walk from s to t w/ $\leq i$ edges

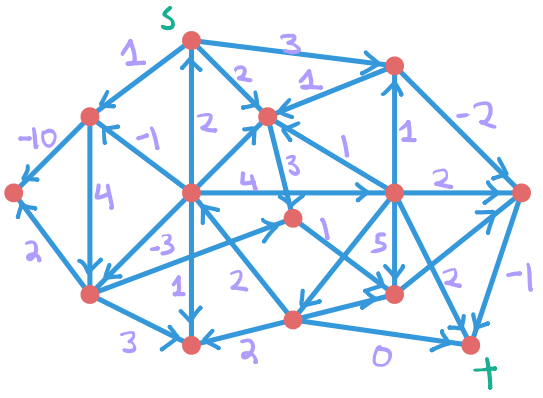
"doubling trick"

$$SW(s,t,2^i) = \begin{cases} \left.\begin{cases} 0 & \text{if } s=t \\ \ell(s,t) & \text{if } (s,t)\in E \\ +\infty & \text{if } (s,t)\notin E \end{cases}\right\} \text{if } i=0 \\ \min\left\{ \begin{array}{l} SW(s,t,2^{i-1}) \\ \min\limits_u \; SW(s,u,2^{i-1}) + SW(u,t,2^{i-1}) \end{array} \right\} \text{otherwise} \end{cases}$$

Running time (w/ caching)

A different recursion for APSP

index $V = \{v_1, v_2, \ldots, v_n\}$

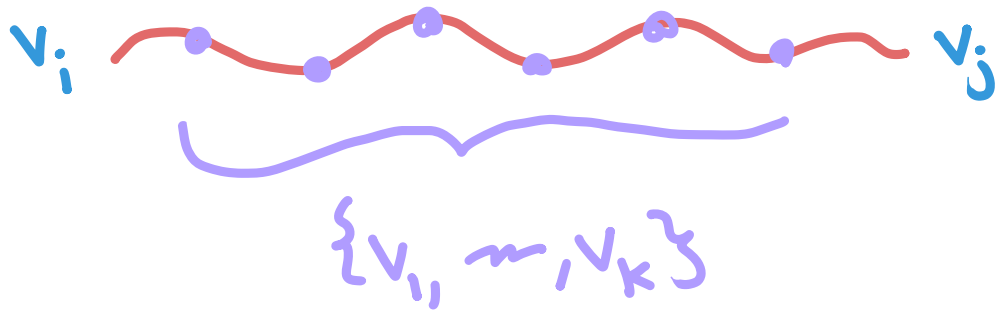


Goal: compute d(s,t) for <u>all</u> s,t

assume no negative cycles (for simplicity)

Recursive spec:

SW(i,j,k) = length of shortest walk from v_i to v_j through $v_1, \ldots, v_k$

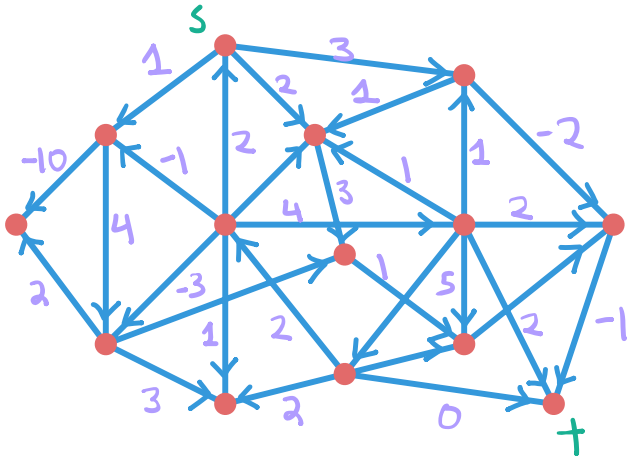


A different recursion for APSP

index $V = \{v_1, v_2, \ldots, v_n\}$

Recursive spec:

$SW(i,j,k)$ = length of shortest walk from v_i to v_j through $v_1, \ldots, v_k$



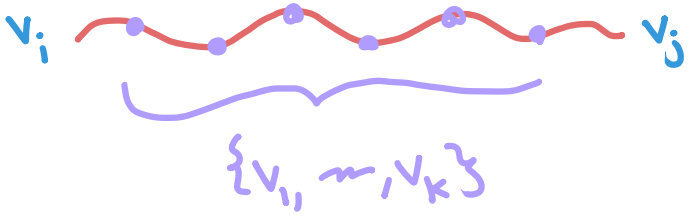
Goal: compute $d(s,t)$ for all s,t

assume no negative cycles (for simplicity)

$SW(i,j,k) =$

$$\begin{cases} 0 & \text{if } i=j \\ \ell(s,t) & \text{if } (v_i,v_j)\in E \\ +\infty & \text{if } (v_i,v_j)\notin E \end{cases} \Big\} \text{ if } k=0$$

$$\min\begin{cases} SW(i,j,k-1) \\ SW(i,k,k-1)+SW(k,j,k-1) \end{cases} \Big\} \text{ otherwise}$$

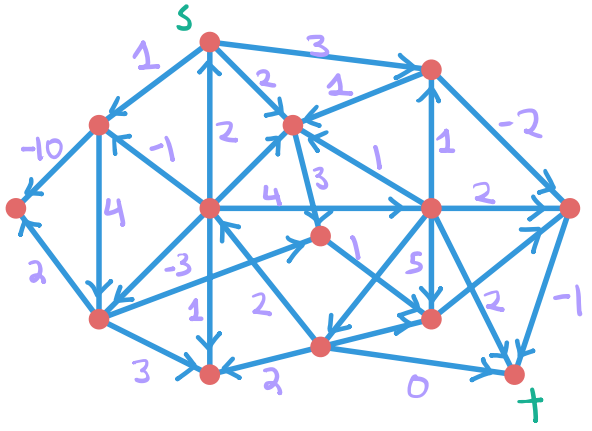


A different recursion for APSP

index $V = \{v_1, v_2, \ldots, v_n\}$

Recursive spec:

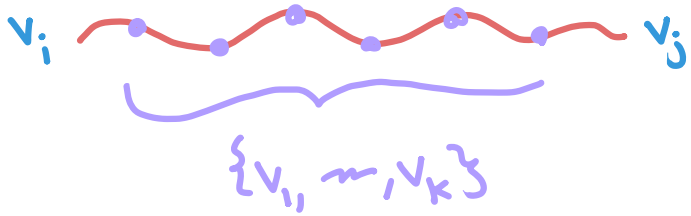
$SW(i,j,k)$ = length of shortest walk from v_i to v_j through $v_1, \ldots, v_k$



Goal: compute $d(s,t)$ for <u>all</u> s,t

assume no negative cycles (for simplicity)

$$SW(i,j,k) = \begin{cases} \begin{cases} 0 & \text{if } i=j \\ \ell(s,t) & \text{if } (v_i,v_j)\in E \\ +\infty & \text{if } (v_i,v_j)\notin E \end{cases} & \text{if } k=0 \\ \min\begin{cases} SW(i,j,k-1) \\ SW(i,k,k-1)+SW(k,j,k-1) \end{cases} & \text{otherwise} \end{cases}$$



Running time (w/ caching)

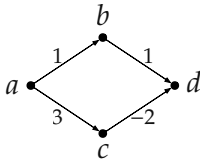
Chapter 8

Negative edge lengths and all-pairs shortest paths

Previously we discussed shortest path problems with positive edge weights, where the goal is to get from point a to point b as fast as possible. Here the points refer to vertices in a directed graph; the routes are the walks through the graph, and "fast" was based on the sum of edge lengths along the walk. Specifically, we obtained the linear time `BFS` algorithm for unweighted graphs and the nearly linear time Dijkstra's algorithm for positively weighted graphs.

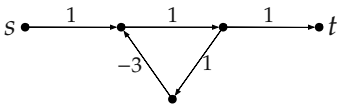
Positive edge lengths are a critical assumption for these algorithms. They permit an induction hypothesis increasing in distance; indeed, both algorithms identified and finalized distance labels in increasing distance from the source. But this argument breaks down for negative edge lengths.

Consider going from point a to point d in the weighted graph on the right. Dijkstra's algorithm would declare that point b was at distance 1, and that point d was at distance 2, before realizing that the path through c has total length 1.



Let us state the problem formally. Let $G = (V, E)$ be a directed graph with real-valued (and possibly negative) edge lengths $\ell : E \to \mathbb{R}$. Fix $s, t \in V$. What is the length of the shortest walk from s to t?

The length can be smaller than any finite number, in which case we say that the distance is $-\infty$. This occurs if and only if we have a negative cycle on the way from s to t (as we will see). On the right, the negative cycle between s and t makes the distance $-\infty$.



8.1 Negative edges, shortest walks, and shortest paths

Recall the definition of distances in a directed graph (page 116 on page 116). The distance from s to t is usually finite and equal to the length of the shortest walk from s to t, but there are two non-finite exceptions. The distance is $+\infty$ if there is no walk,

and the distance is $-\infty$ if for every value L, there is an (s,t)-walk of total length $< L$. That is, there is no lower bound on the lengths of (s,t)-walks. The $+\infty$ case is addressed by reachability. The $-\infty$ case did not arise for positive edge weights, because 0 is always a lower bound. But here, with negative edge weights, $-\infty$ can easily occur; e.g., with a negative cycle.

We address $-\infty$ head on, as well as some of the ambiguity in shortest *walks* vs shortest *paths*, with the following lemma.

Lemma 8.1. *Let $s, t \in V$ and suppose s can reach t. The distance from s to t is finite (and not $-\infty$) iff the shortest (s,t)-walk is attained by a path.*[1]

Proof. If the shortest (s,t)-walk is a path then by definition the (finite) length of this path is the (s,t)-distance.

Conversely, suppose the distance from s to t is a finite value L, which means it is attained by some (s,t)-walk. Among all (s,t)-walks of total length L, let w minimize the total number of edges along that path. We will show that w is a path. Indeed, if w is not a path, then it contains a cycle. The total length of that cycle falls into one of two cases:

1. The total length of the cycle is nonnegative, in which removing the cycle decreases the number of edges in w without increasing the overall length – a contradiction to the choice of w.

2. The total length of the cycle is negative, in which case repeating the cycle decreases the overall length of the walk – a contradiction to the choice of w.

So w is a path. □

8.2 Shortest walks with negative edges

The problem is to find the shortest walk and there is no clear induction hypothesis. As noted above, negative edges means we can't rely on distances to implicitly supply an induction argument, as we did in developing BFS and Dijkstra's shortest path algorithms. Instead we will *lift* the problem, so to speak, by inserting an additional parameter that makes it more conducive to an induction argument. For vertices $s, t \in V$ and an integer $k \in \mathbb{Z}_{\geq 0}$, we declare

> `SW(`s`,`t`,`k`)` = the length of the shortest walk from s to t with at most k edges (defined as $+\infty$ if there is none).[2]

[1] That is, there is an (s,t)-path with total length less than or equal to any other (s,t)-walk.

[2] Currently we are interested in the case where s is fixed, but later it will be convenient to have s as a parameter.

Here there is a clear induction argument on k – the k edge walks build on $k-1$ edge walks by appending an extra edge. In the code below, $\delta^-(t)$ denotes the set of edges (v,t) with endpoint t.[3]

```
SW(s,t,k)
1. If k = 0:
   /* The only zero edge walk is the empty walk.                    */
   A. If s = t then return 0.
   B. Otherwise return +∞.
2. Return the minimum of:
   A. SW(s,t,k − 1).
   B. The minimum of SW(s,v,k − 1) + ℓ(v,t) over all edges (v,t) ∈ δ⁻(t).
```

We cache the solutions to our algorithm above. With dynamic programming, for fixed s and t, and each $k \in \mathbb{Z}_{\geq 0}$, `SW(s,t,k)` takes $O\Big(1+\deg^-(t)\Big)$ time.[4] For fixed s and k, it takes $O(m+n)$ time to compute all of the subproblems for a fixed value of k.

Lemma 8.2. *Let $G=(V,E)$ be a directed graph, with m edges, n vertices, and real-valued edge weights. Let $s \in V$ be fixed. In $O(k(m+n))$ time, one can compute `SW(s,t,i)` for all $t \in V$ and all $i \leq k$.*

Shortest paths when there are no negative cycles. Suppose all the distances from s are finite. Then by lemma 8.1, the (s,t)-distance is attained by a path for every vertex t.

Corollary 8.3. *Let $G=(V,E)$ be a directed graph with m edges, n vertices, real-valued edge weights. Let $s \in V$ be fixed and suppose that all distances from s are finite. In $O(mn)$ time, one can compute the length of the shortest (s,t)-path for all $t \in V$.*

Stationary points and $-\infty$. The remaining issue is $-\infty$. Intuitively, if $d(s,t) = -\infty$, then `SW(s,t,k)` should decrease towards $-\infty$ as $k \to \infty$. That does not mean that `SW(s,t,k)` decreases with *every* k; we may need several edges to go all the way around a negative cycle and decrease `SW(s,t,k)`. It's also not clear how long to keep iterating on `SW(s,t,k)` before we can conclude that there is no lower bound. So we have to analyze `SW(s,t,k)` carefully.

[3] More generally, for $S \subseteq V$, $\delta^-(S) \stackrel{\text{def}}{=} \{((u,v) \in E : u \notin S, v \in S)\}$.

[4] $\deg^-(v)$ denotes the in-degree of v.

For $k \in \mathbb{N}$, let

$$A_k = \{v \in V \text{ where } \texttt{SW}(s,t,k) \neq \texttt{SW}(s,t,k-1)\}$$

be the set of vertices v where $\texttt{SW}(s,t,k)$ decreases compared to the previous iteration for $k-1$. Recalling the definition of $\texttt{SW}(s,v,k)$, these are the vertices where the best k-edge walk from s to v is strictly shorter than the best $(k-1)$-edge walk. Then for each k, we have

$$A_{k+1} \subseteq N^+(A_k),$$

where for $U \subseteq V$, $N^+(U)$ denotes the "out-neighborhood" of U consisting of all vertices that are endpoint to some directed edge starting in U:

$$N^+(U) = \{v \text{ where } (u,v) \in E \text{ for some } u \in U\}.$$

Continuing on, we also have

$$A_{k+2} \subseteq N^+\Big(N^+(A_k)\Big),\ A_{k+3} \subseteq N^+\Big(N^+\Big(N^+(A_k)\Big)\Big),\ A_{k+4} \subseteq N^+\Big(N^+\Big(N^+\Big(N^+(A_k)\Big)\Big)\Big),$$

and so forth. In general, for any $\ell \geq k$, all the vertices in A_ℓ must be reachable from A_k. Conversely, for all vertices t that are *not* reachable from A_k, we know that $\texttt{SW}(s,t,\ell)$ will never change. They are forever *fixed points* in this process. Consequently $d(s,t) = \texttt{SW}(s,t,k-1)$ for any such t.

Now consider in particular A_n. These are the vertices v where the length of the shortest (s,v)-walk with at most n edges is less then the length of the shortest (s,v)-walk with at most $n-1$ edges. In particular, since any path has at most $n-1$ edges, the distance from (s,v) is *not* attained by a path. By lemma 8.1, the distance from s to v must be $-\infty$.

So all vertices $v \in A_n$ have $d(s,v) = -\infty$. Additionally, any vertex v reachable from A_n has distance $-\infty$. (Why?) Meanwhile, if a vertex v is not reachable from A_n, then (since $\texttt{SW}(s,v,\ell)$ would never change for $\ell \geq n$) we have $\texttt{SW}(s,v,n) = d(s,v)$. All together, we now have all the finite distances from s as well as all the vertices with distance $-\infty$. Funny enough, we churn $\texttt{SW}(s,t,n-1)$ for *just one more iteration* to get all the distances, finite or not.

Theorem 8.4. *In $O(mn)$ time, we can compute the (s,t)-distance for all vertices $v \in V$ (with the corresponding shortest path whenever the distance is finite).*

This algorithm is commonly referred to as the *Bellman-Ford algorithm* [Bel58; For56], although it was also described by Shimbel [Shi55]; an alternative, more descriptive name might be "that dynamic programming algorithm based on the number of edges for SSSP with negative edge lengths".

Detecting negative cycles. Suppose we want to decide if there is *any* negative cycle in the graph. The Bellman-Ford algorithm will detect if there is a negative cycle that is reachable from a particular source s. However there may be negative cycles that are not reachable from s. We can try all choices of s but this can be very slow. Can we do better?

We introduce an additional "super-source" vertex $s^\star$ with an edge $(s^\star, v)$ of length 0 for every $v \in V$. In particular $s^\star$ can reach every vertex directly. If we apply theorem 8.4 from $s^\star$ (that is, compute $\texttt{SW}(s^\star, t, n+1)$ for all t), then any vertex on a negative cycle will have distance $-\infty$. In fact any vertex reachable from a negative cycle will also have distance $-\infty$.[5]

Corollary 8.5. *Given a directed graph G with real-valued edge weights, in $O(mn)$ time, one can identify all vertices reachable from a negative cycle.*

8.3 All pairs shortest paths and the doubling trick

We now consider the problem of computing the distances between *all pairs of vertices*, not just from a single source. We will focus on the case where G has no negative cycles for simplicity. Then all finite distances are obtained by shortest paths. Note that we can detect if there is a negative cycle using the Bellman-Ford algorithm (corollary 8.5).

Recall our definition of $\texttt{SW}(s, t, k)$, which we repeat below:

> $\texttt{SW}(s, t, k)$ = the length of the shortest walk from s to t with at most k edges (defined as $+\infty$ if there is none).

The Bellman-Ford algorithm gave one implementation of $\texttt{SW}(s, t, k)$ (on page 131). There, we had s fixed and varied t; now we are interested in all choices of $s, t \in V$. To this end, we could call $\texttt{SW}(s, t, n)$ for all choices $s, t \in V$. With dynamic programming, this would take $O(mn^2)$ time. Can we do better?

Consider the case $k = 4$. When s was fixed we computed $\texttt{SW}(s, t, k)$ based on the values of $\texttt{SW}(s, v, k-1)$ for $v \in V$. Now that we're varying s as well, we could take

[5] Don't change the algorithm, change the graph.

advantage of $\texttt{SW}(u,v,\ell)$ for any $u, v \in V$ and $\ell \leq k$. In particular, any (s,t)-walk of at most 4 edges can be divided into two walks of at most 2 edges, one from s to u and another from u to t for some $u \in V$. Of course we can do this dividing in half for any even k; better yet, we should do this *only for powers of* 2. For k equal to powers of 2, consider the following alternative implementation for SW.

$\texttt{SW}(s,t,2^i)$

1. If $i = 0$, then return the minimum of:
   A. 0 when $i = j$
   B. $\ell(s,t)$ when $(s,t) \in E$.
   C. $+\infty$.
2. Return the minimum of
   A. $\texttt{SW}(s,t,2^{i-1})$
   B. $\texttt{SW}(s,v,2^{i-1}) + \texttt{SW}(v,t,2^{i-1})$ for all $v \in V$. *// $O(n)$ time.*

The key difference between this implementation of $\texttt{SW}(s,t,k)$ and Bellman-Ford is in line (2.B) – we are trying to concatenate two paths with (at most) 2^{i-1} edges to form an edge with at most 2^i edges.

Now, for fixed 2^i, there are $O(n^2)$ subproblems. Each subproblem has a loop of length n. Caching the recursive calls, we have the following.

Lemma 8.6. *Let $G = (V,E)$ be a directed graph with n vertices. In $O(n^3 i)$ time, one can compute $\texttt{SW}(s,t,2^j)$ for all $s,t \in V$ and all $j \leq i$.*

If there is no negative cycles (as we have assumed) then $\texttt{SW}(s,t,n-1)$ (or any value greater than $n-1$) gives the distance from s to t. For $k = \lceil \log n \rceil$, lemma 8.6 gives us the distances between all pairs of vertices.

Corollary 8.7. *Let $G = (V,E)$ be a weighted, directed graph with n vertices and no negative cycles. Then the distances between all pairs of vertices can be computed in $O(n^3 \log n)$ time.*

This running time is better than running Bellman-Ford from every s unless m is extremely close to n.

8.4 A different approach to APSP

We now consider a different approach to the all-pairs shortest-paths problem. This algorithm will only work if there are no negative cycles. In this case all distances are given by shortest paths.

Up to now, all the algorithms in this chapter were based on induction on the number of edges in the walk. Here we present a different approach, based on the subset of vertices used as intermediaries in the shortest path. Let us number the vertices $v_1, \dots, v_n$ arbitrarily. We define:

> $\texttt{SP}(i,j,k)$ = the length of the shortest path from v_i to v_j using only $v_1, \dots, v_k$ as intermediate vertices.

We implement the above as follows.

$\texttt{SP}(i,j,k)$

1. If $k = 0$:
   - A. If $i = j$ return 0.
   - B. If $(v_i, v_j) \in E$ then return $\ell(v_i, v_j)$.
   - C. Otherwise return $+\infty$.
2. Return the minimum of:
   - A. $\texttt{SP}(i,j,k-1)$
   - B. $\texttt{SP}(i,k,k-1) + \texttt{SP}(k,j,k-1)$

There are only $O(n^3)$ subproblems so we can apply dynamic programming and cache all the answers. Each subproblem takes constant time excluding recursive calls. So it takes $O(n^3)$ total time to compute $\texttt{SP}(i,j,k)$ for all i, j, k.

We prove the pseudocode implements the recursive spec by induction on k. The base case is $k = 0$, where there are no intermediate vertices. For fixed v_i and v_j, the only edge available is (v_i, v_j) (if present). Thus the only options for the distance is 0 when $v_i = v_j$, $\ell(v_i, v_j)$ if the edge (v_i, v_j) exists, and by default $+\infty$; the algorithm returns the minimum of these options.

Consider $\texttt{SP}(i,j,k)$ for $k > 0$. The shortest (v_i, v_j)-path through $\{v_1, \dots, v_k\}$ either uses v_k or it doesn't. If it doesn't, then by induction on k, its length is given by $\texttt{SP}(i,j,k-1)$. If it does, then the part from v_1 to v_k is a path through $\{v_1, \dots, v_{k-1}\}$ and the remaining part from v_k to v_j is also path through $\{v_1, \dots, v_{k-1}\}$. By induction on k, $\texttt{SP}(i,k,k-1)$ returns the length of the shortest path of the first type, and $\texttt{SP}(k,j,k-1)$ returns the length of the longest path of the second type. Their sum gives the length of the shortest (v_1, v_j) path through $\{v_1, \dots, v_k\}$. This completes the proof of correctness.

This algorithm is commonly called the *Floyd-Warshall* algorithm although there are several relevant publications from roughly the same time [Flo62; Ing62; Roy59; War62] (see the Wikipedia article).

Theorem 8.8. *Let $G = (V, E)$ be a directed graph with real-valued edge weights and no negative cycles. Then one can compute all pairs shortest paths in $O(n^3)$ time.*

This is faster by a logarithmic factor than the running time obtained by the doubling trick in the previous section.

We should point out that when the edge weights are positive, you are better off running Dijkstra's algorithm from all $s \in V$. We point out that there is an extension of Dijkstra's algorithm to handle negative edges, called Johnson's algorithm [Joh77], that we do not discuss.

8.5 Additional notes and references

The content in this chapter overlaps with [Eri19, chapter 9], [KT06, sections 6.8–6.9], [DPV08, chapter 4], [CLRS09, chapter 24], and [DD11, (video) lecture 19].

Spring 2022 lecture materials. Click on the links below for the following files:

- Handwritten notes prepared before the lecture.
- Handwritten notes annotated during the presentation.
- Handwritten notes annotated during the re-recording.
- Re-recorded video lecture.

8.6 Exercises

Exercise 8.1. The following problem (and more elaborate extensions) appear in reinforcement learning. Let $G = (V, E)$ be a directed graph and let the edges be annotated by positive edge weights $r : E \to \mathbb{R}_{>0}$. We think of the vertices as states of some device under our control, and the edge weights $r(e)$ as rewards obtained by traversing the edge e as follows. Let $s \in V$ be a fixed starting vertex / state. You may assume for simplicity that G has no sinks.

1. Given an integer $k \leq n$, the goal is to compute a walk of length k maximizing the sum of rewards along that walk.[6] (If you repeat an edge, you get the same reward each time you repeat the edge.) Design and analyze an algorithm for this problem.

2. Here we also incorporate a *discount rate*. Let $\alpha \in (0, 1)$ be given. Given a walk with edges $e_1, \ldots, e_k$, the *discounted total reward* of the walk is given by

$$r(e_1) + \alpha r(e_2) + \cdots + \alpha^{k-1} r(e_k).$$

[6] There is always a k-edge walk under the assumption that G has no sinks.

The idea is that if we think of each edge traversal as also taking a unit of time, then the rewards attained far off in the future are perceived to be worth less than the rewards attained now and in the short term.

Given an integer $k \leq n$, the goal is to compute a walk of length k maximizing the discounted total reward of the walk. Design and analyze an algorithm for this problem.

Exercise 8.2. Recall that in foreign exchange, different currencies can be exchanged at various exchange rates. For example, at the time of writing, one can exchange 1 US dollar for about 0.89 Euro's. *Currency arbitrage* occurs when you start with some quantity of one currency, and make a sequence of exchanges through different currencies and end up with even more of the original currency than you started with.

Suppose we have n countries and for every (ordered) pair of countries X and Y we have an exchange rate $r_{X,Y}$; meaning, one unit of currency X can be exchanged for $r_{X,Y}$ units of currency Y. Consider the problem of identifying currency arbitrary starting from a fixed currency X, or deciding that no arbitrage is possible. Design and analyze an algorithm for this problem.