

Negative Edge Lengths
and All Pairs Shortest Paths

(high-level comment)

"only two algorithms"*

① identify subproblems

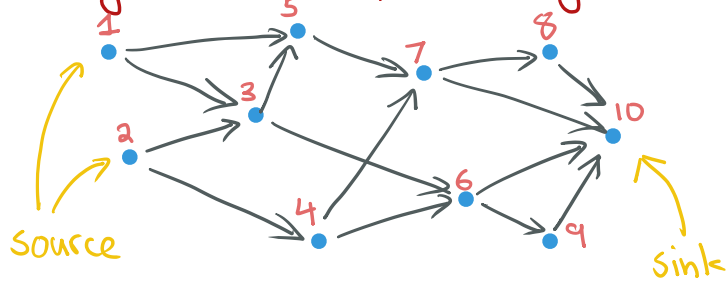
② change the input

*(mostly; we also had a little
divide + conquer extend. merge-sort)

1.5 weeks ago

Connectivity in Directed Graphs

Topological sort: keep removing sources



"depth-first search"

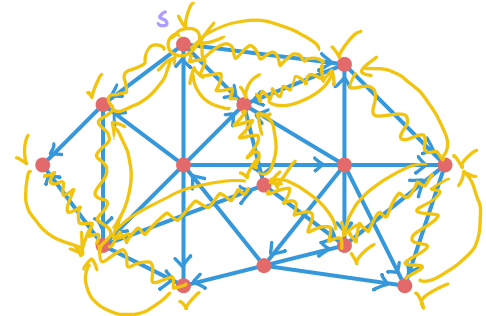
DFS(v):

① if v is unmarked

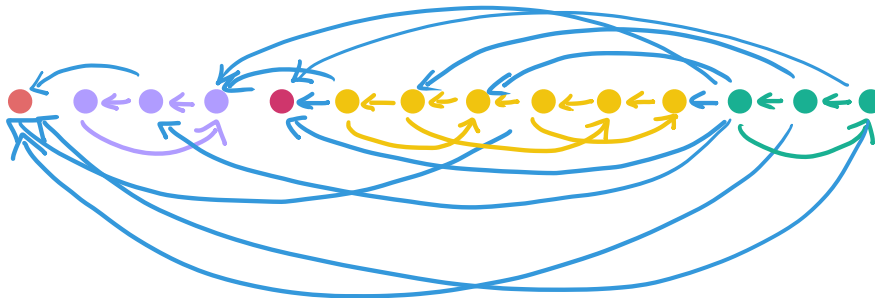
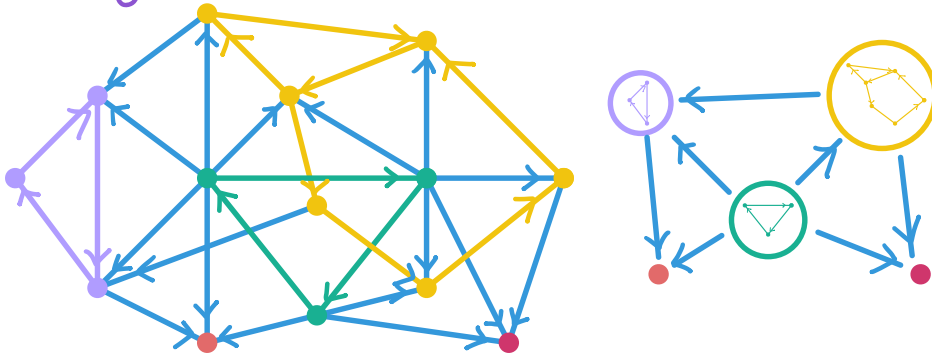
 Ⓐ mark v

 Ⓑ for each edge $v \rightarrow w$

 ① DFS(w)



strongly connected component (SCC)



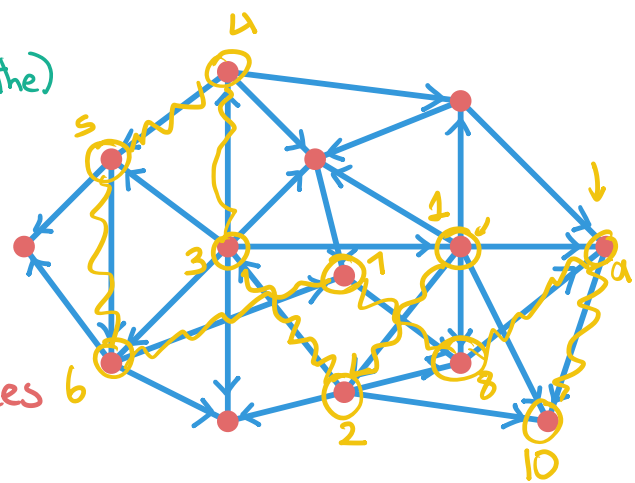
sink-first-SCC($G = (V, E)$)

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 $O(m+n)$ time

1 week ago: longest path

Find the (length of the) longest path in G (eg. # edges)



Hamiltonian path:

path visiting all vertices

does G have a Hamiltonian path?

Longest ~~path~~ walk in a DAG

no cycles
 $\Rightarrow$ walk = path

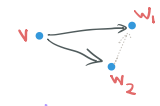
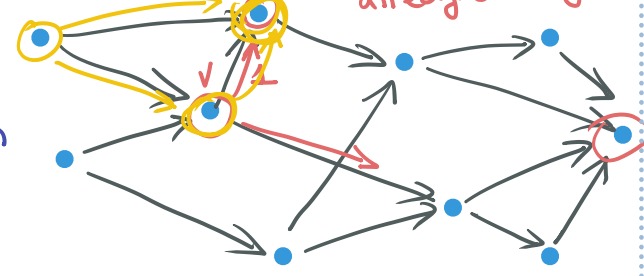
Recursion

$LW(v)$ = length of Longest Walk from v

$LW(v) = \begin{cases} 0 & \text{if } v \text{ is a sink} \end{cases}$

+ caching $\left\{ 1 + \max_{(v,w) \in E(v)} LW(w) \right.$

$J^+(v)$ = "out-cut" δv
= all edges leaving v

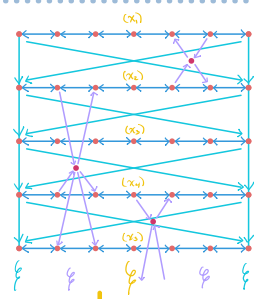


otherwise

Theorem 7.1. A polynomial time algorithm for the Hamiltonian path problem implies a polynomial time algorithm for SAT.

$\varphi(x_1, \dots, x_n) = (x_1 \vee \bar{x}_2) \wedge (x_2 \vee x_3 \vee \bar{x}_4) \wedge \dots$

yes, no



Shortest (s,t) -walk w/ $\leq k$ edges (next time: remove this constraint)

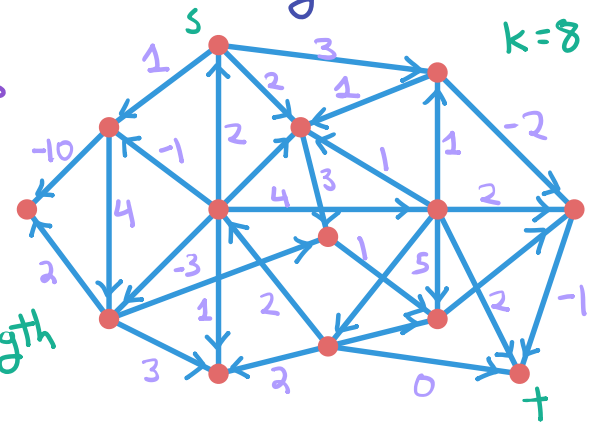
(time permitting)

edges have lengths

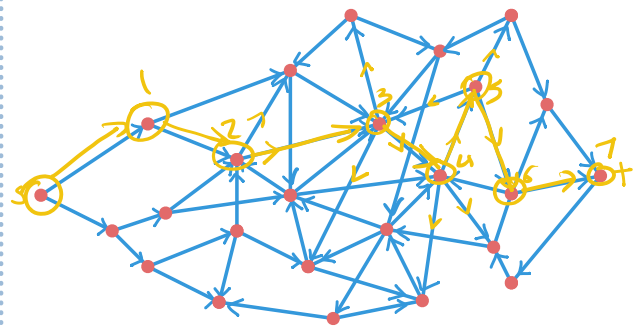
Goal: go from s to t w/ $\leq k$ edges and min total length

Recursion:

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



last time: distances w/ positive weights



shortest way from s to t ?

Directed $G=(V,E)$, edge lengths $\ell: E \rightarrow \mathbb{R}$



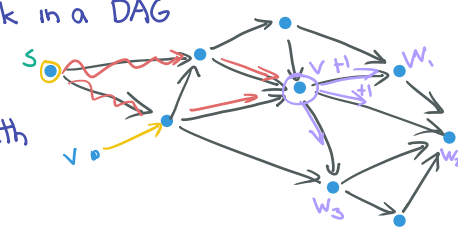
Distance $d(s,t)$

$d(s,t) = \min$ length of any (s,t) walk

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

Shortest walk in a DAG

no cycles
 $\Rightarrow$ walk = path



Recursion

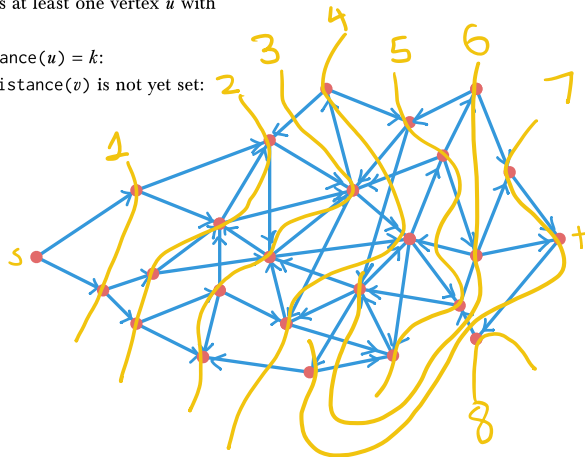
$SW(v) =$ length of Shortest Walk from s to v

$$SW(v) = \begin{cases} 0 & \text{if } v=s \\ +\infty & \text{if } v \text{ is a source} \\ \min_{(u,v) \in E} \ell(u,v) + SW(u) & \text{otherwise} \end{cases}$$

before v ?

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

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



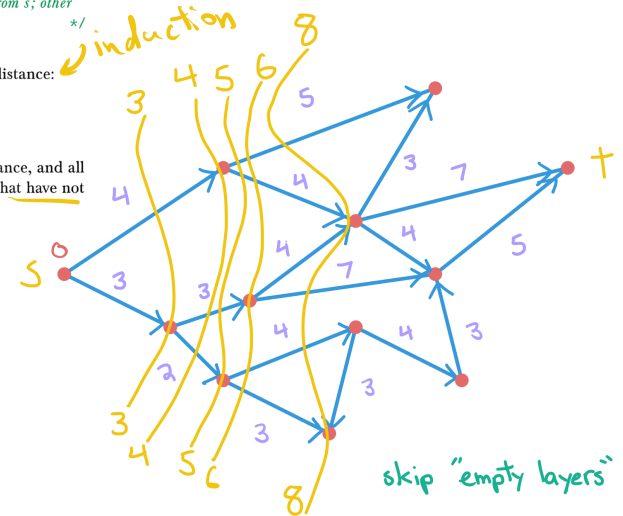
$O(mn)$ time

(w/ simple bookkeeping)

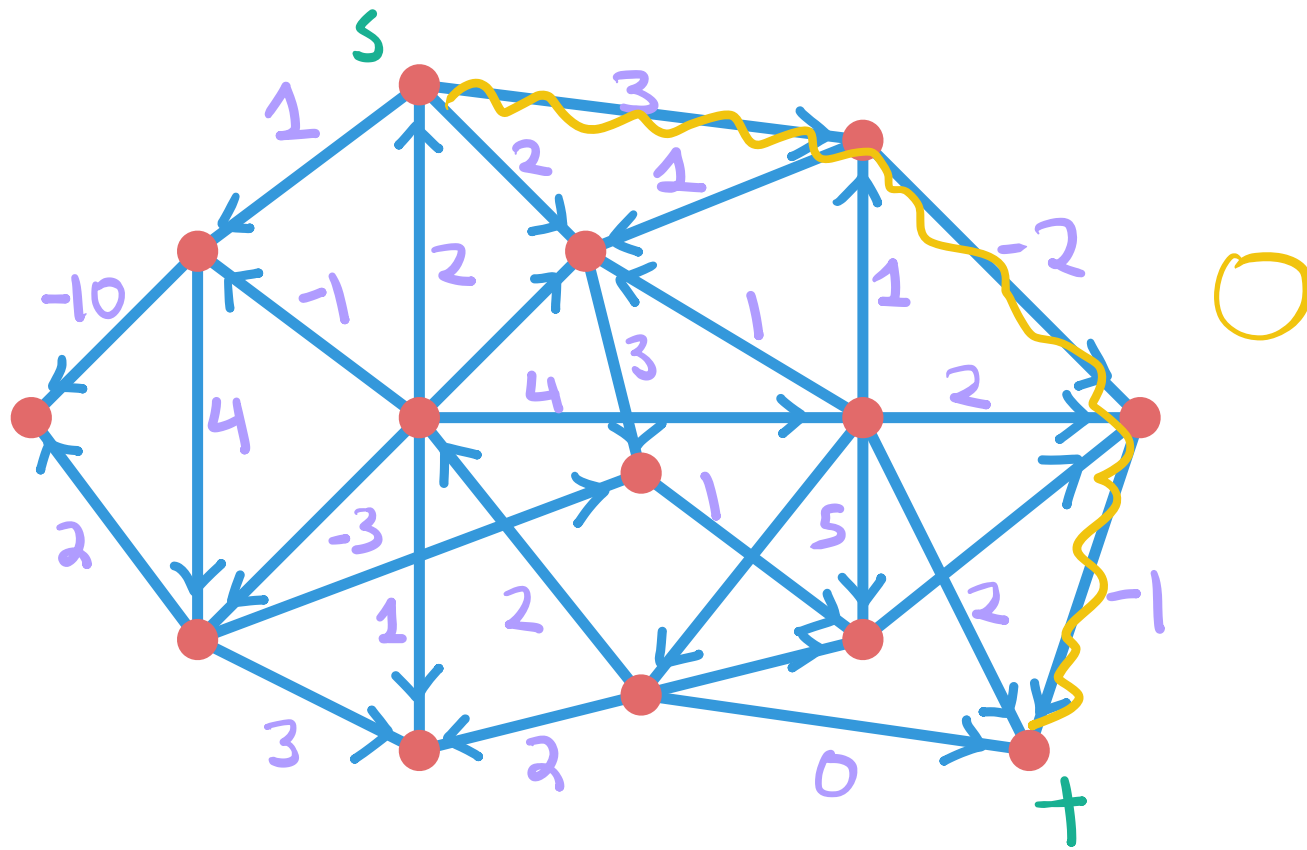
Dijkstra($G=(V,E)$, $\ell: E \rightarrow \mathbb{R}_{>0}$, $s \in V$)

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

1. Set $\text{distance}(s) = 0$.
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 $\text{distance}(v) = \text{distance}(u) + \ell(u,v)$.
3. Return all the distance labels.

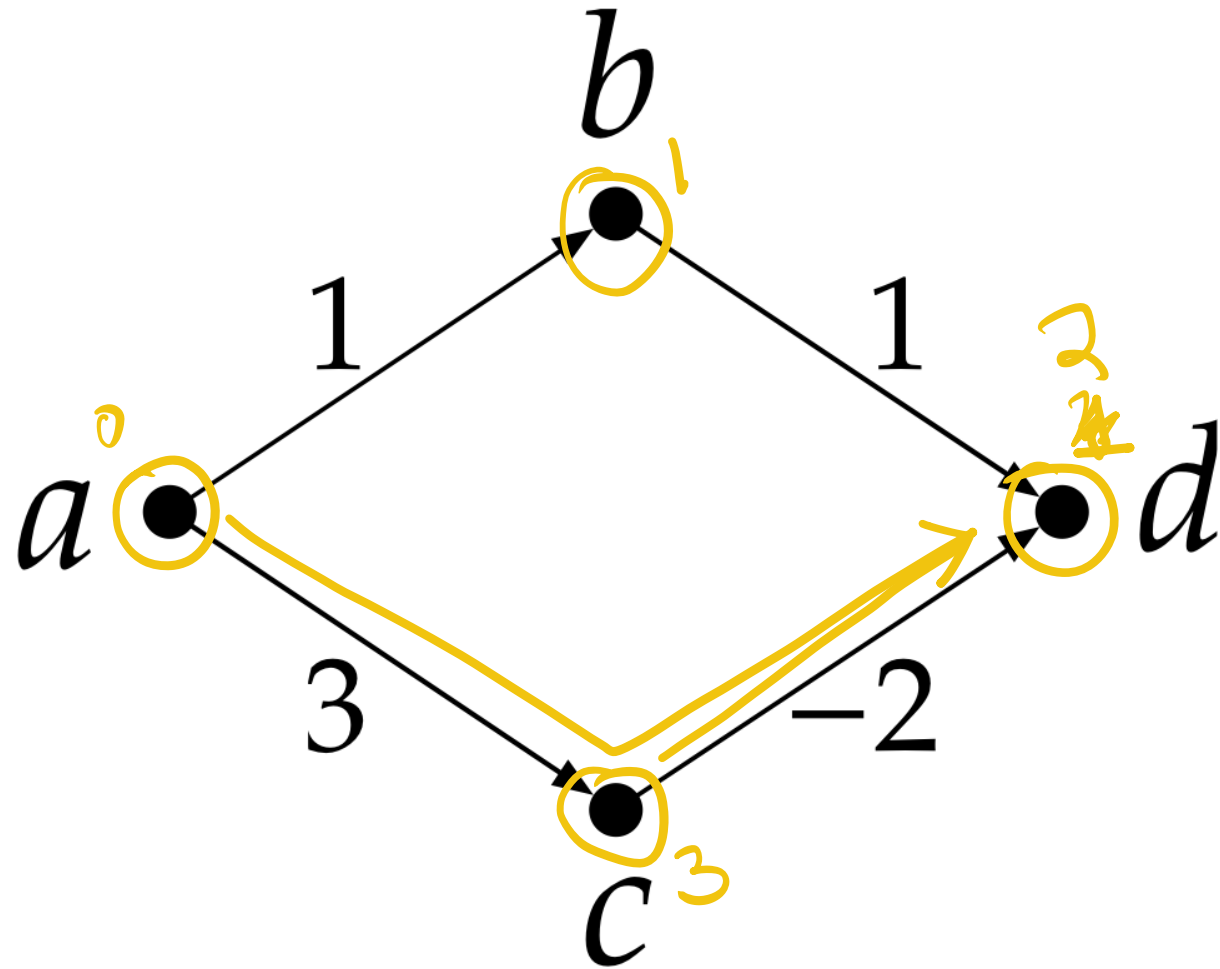


today: Negative Edges

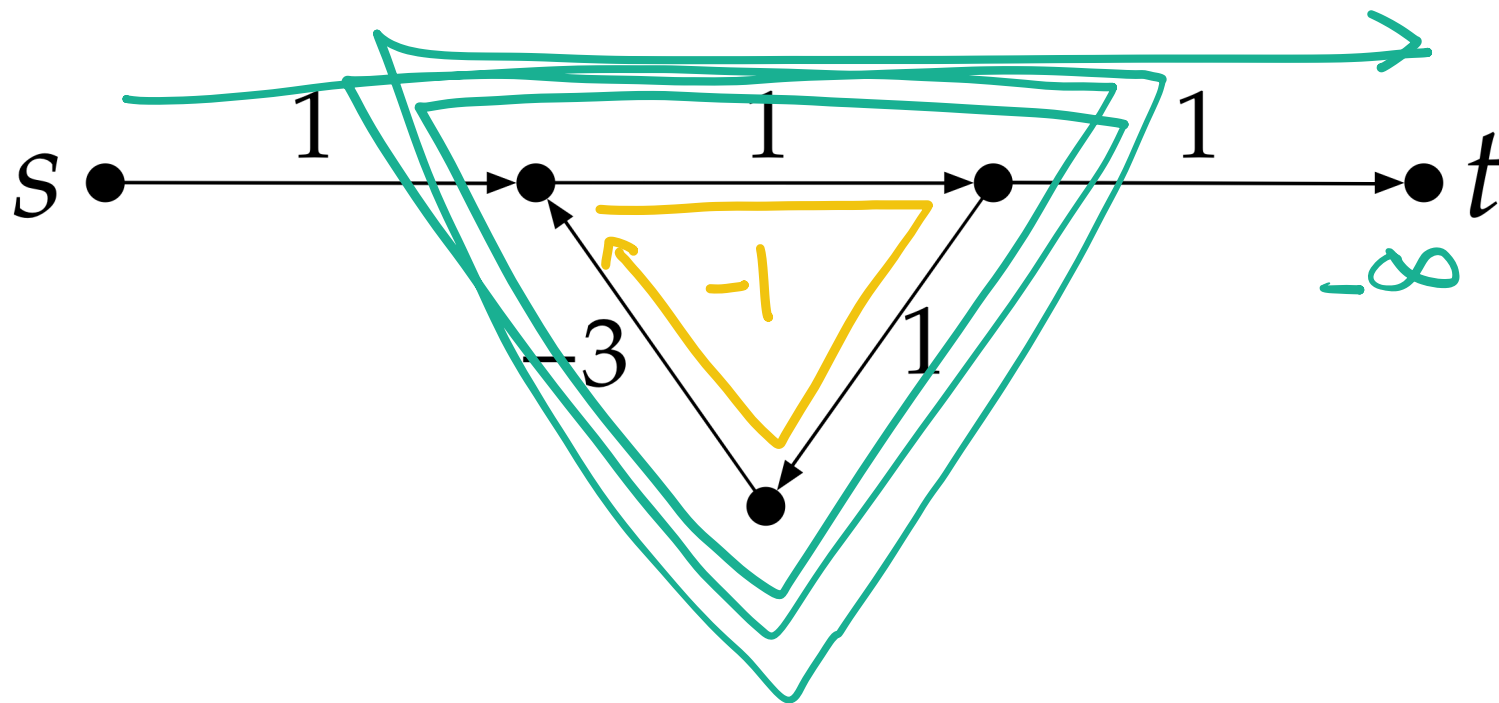


distance from s to t?

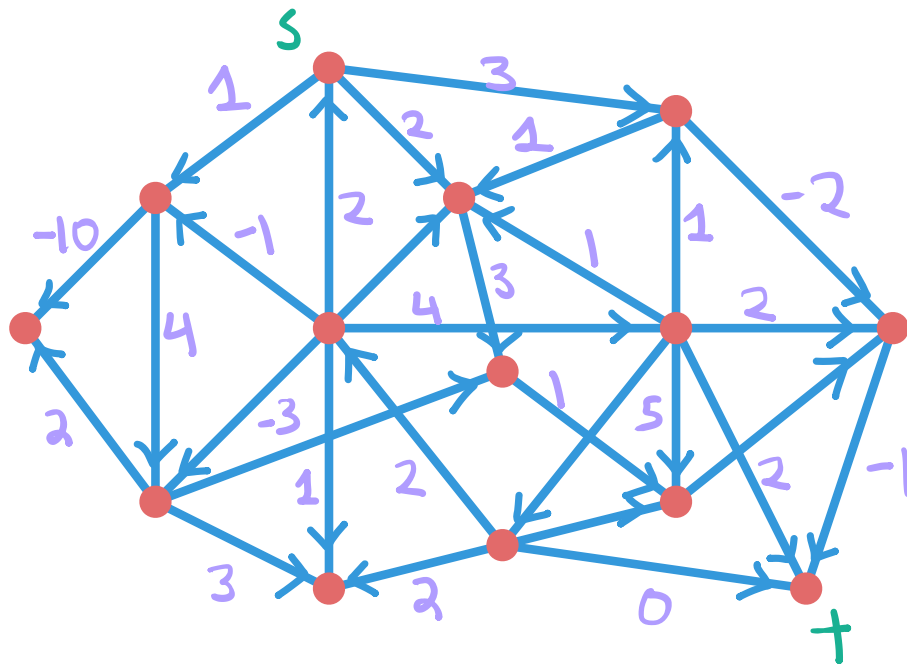
Ask Dijkstra: distance from a to d?



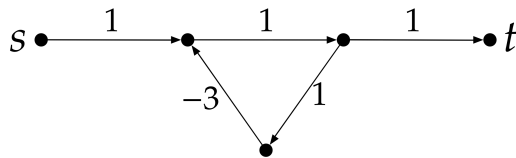
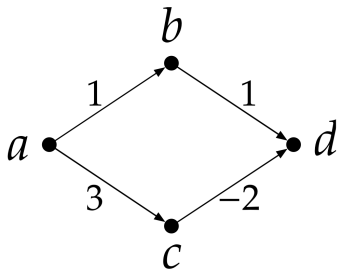
Negative cycles



today: Negative Edges



distance from s to t ?



weird questions

negative cycles?

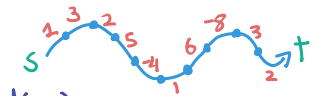
shortest walks vs

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

Directed $G=(V,E)$, edge lengths $\ell: E \rightarrow \mathbb{R}$



Distance $d(s, t)$

$d(s, t) = \min$ length of any (s, t) walk

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

(i) $d(s, t) = L$



let w be an (s, t) -walk of length L
minimizing the # edges in the walk

claim: w is a path.

suppose not. w has a cycle. either

① length of cycle is ≥ 0

(w-cycle) has length $\leq L$, but fewer edges

② length of cycle is < 0

(w+cycle) has length $< L$

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

(only if)



Directed $G=(V,E)$, edge lengths $\ell: E \rightarrow \mathbb{R}$



Distance $d(s, t)$

$d(s, t) = \min$ length of any (s, t) walk

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

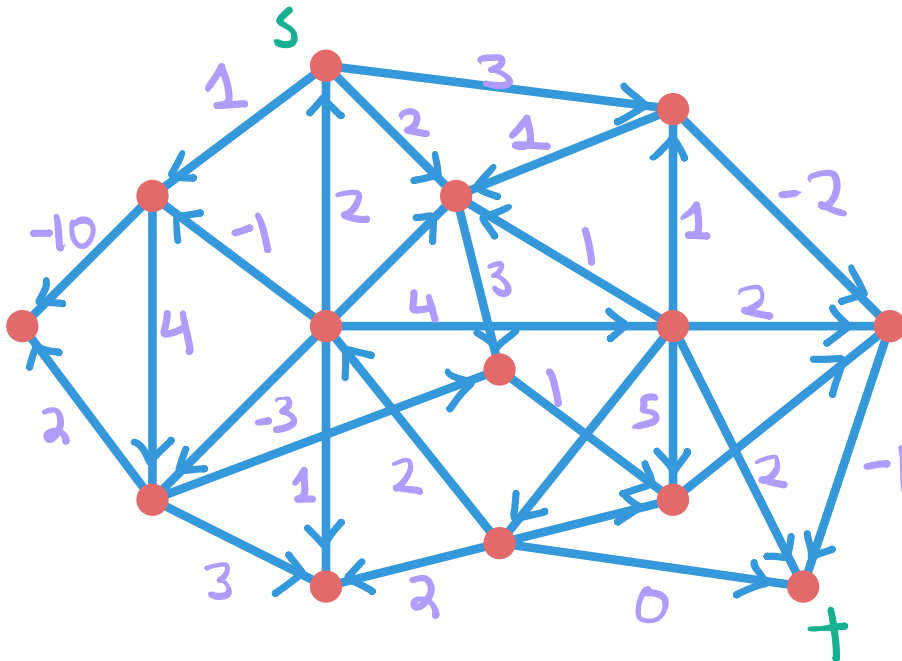
weird questions

✓ negative cycles?

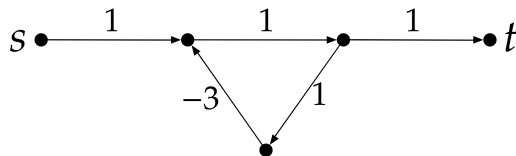
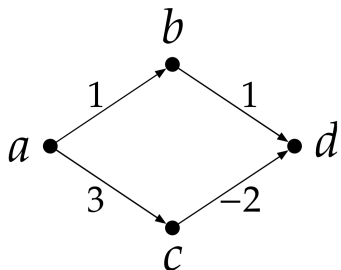
✓ shortest walks vs shortest paths?

induction hypothesis?

today: Negative Edges



distance from s to t?



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

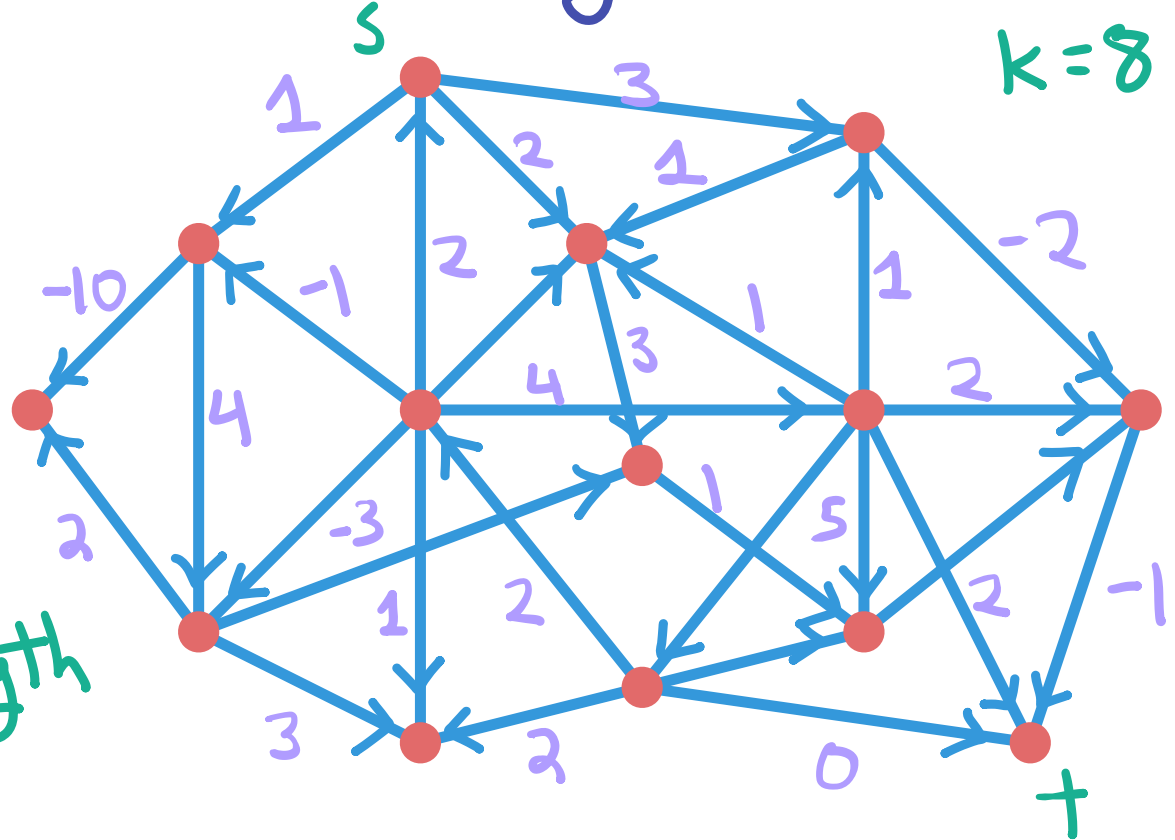
(next time: remove this constraint)

(time permitting)

edges have lengths

$k=8$

Goal:
go from s to t
w/ $\leq k$ edges
and min total length



Recursion:

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

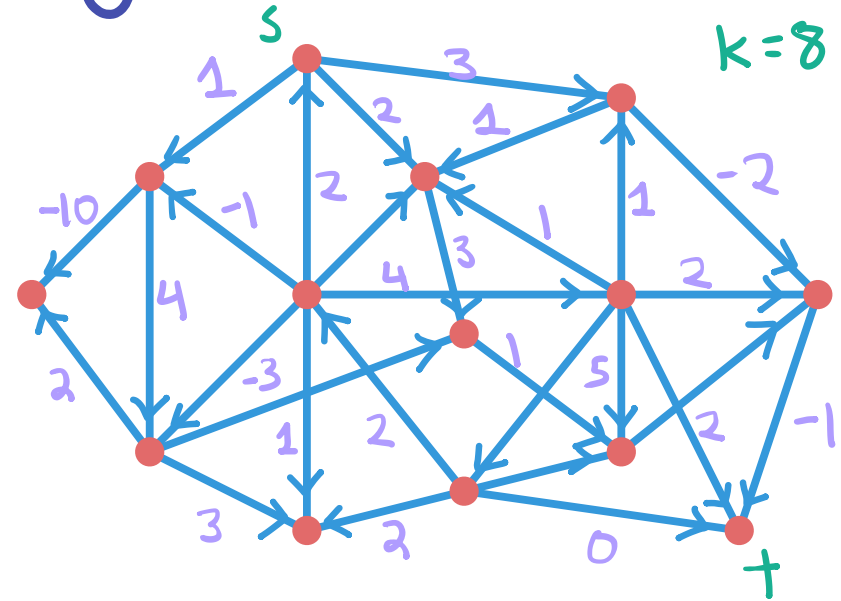
Shortest (s,t)-walk w/ $\leq k$ edges

(next time: remove this constraint)

(time permitting)

Recursion:

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



$k=8$

$SW(v, i) =$

$+\infty$ if $i < 0$

$+\infty$ if $v \neq t$, v sink

$\min_{(w,v) \in \vec{E}(v)} SW(w, i-1)$

$\min \{ 0, \min_{(w,v) \in \vec{E}(v)} SW(w, i-1) \}$

if $v \neq t$

if $v = t$

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

Recursion:

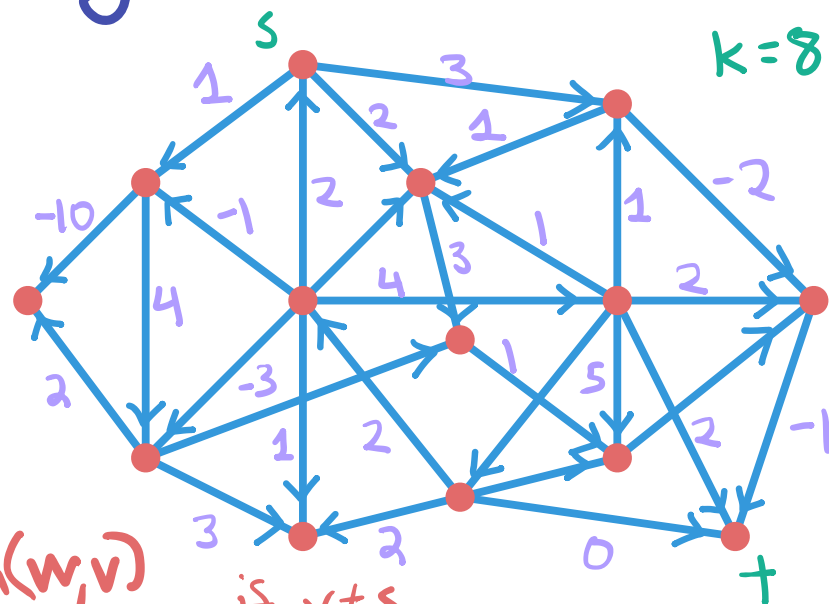
$SW(v, i)$ = length of shortest walk
from s to v w/ $\leq i$ edges

$SW(v, i) =$

$+\infty$ if $i < 0$

$+\infty$ if $v \neq t$, v sink

$\min \{ SW(v, i-1), \min_{(w,v) \in \vec{E}(v)} SW(w, i-1) + \text{len}(w,v) \}$ if $v \neq s$
 $\min \{ 0, \min_{(w,v) \in \vec{E}(v)} SW(w, i-1) + \text{len}(w,v), SW(v, i-1) \}$ if $v = s$



Running time w/ caching

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

(in-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

Suppose no $-\infty$ distances
(no negative cycles)

$k = n-1$ gives
all distance

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

Recursion:

$SW(v, i)$ = length of shortest walk

from s to v w/ $\leq i$ edges

$SW(v, i) =$

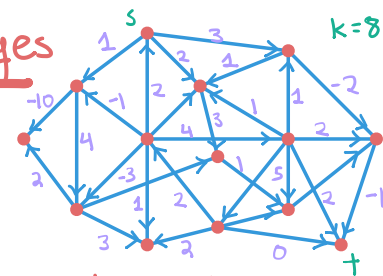
$+\infty$ if $i < 0$

$+\infty$ if $v \neq t$, v sink

$\min_{(v, w) \in \mathcal{E}^+(v)} SW(w, i-1)$

$\min \{ 0, \min_{(v, w) \in \mathcal{E}^+(v)} SW(w, i-1) \}$

if $v \neq t$
if $v = t$ } types



Running time w/ caching $O(k(m+n))$

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

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

$$B_k = V \setminus A_k$$

$$SW(?, i) \rightarrow SW(?, i+1)$$

$$\begin{array}{ccc} \underline{SW(w_1, i)} & \rightarrow & \underline{SW(v, i+1)} \\ \underline{SW(w_2, i)} & \rightarrow & \end{array}$$



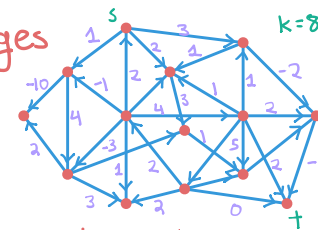
$$A_{k+2} \subseteq N^+(A_{k+1}) \subseteq N^+(N^+(A_k))$$

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

Recursion:

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

$$SW(v, i) = \begin{cases} +\infty & \text{if } i < 0 \\ +\infty & \text{if } v \neq t, v \text{ sink} \\ \min_{(u,v) \in E} SW(u, i-1) & \text{if } v \neq t \\ \min\{0, \min_{(u,v) \in E} SW(u, i-1)\} & \text{if } v = t \end{cases}$$

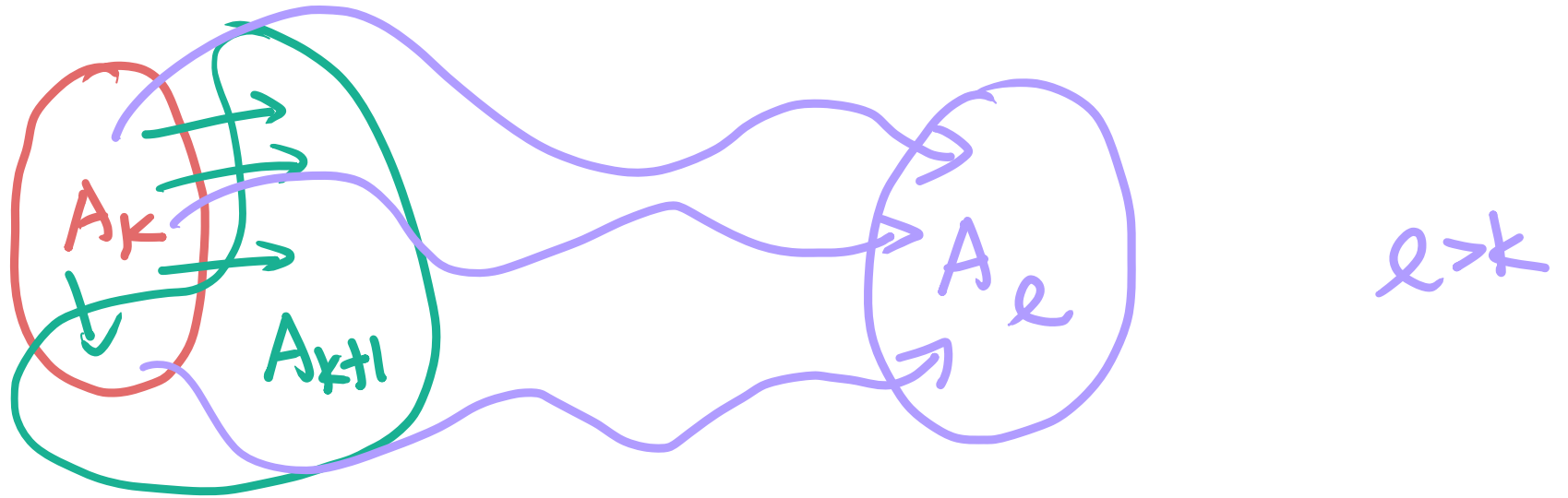


Running time w/ caching $O(k(m+n))$

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

$$N^+(U) = \{v : (u, v) \in E, u \in U\}$$

$$\forall l \geq k \quad A_l \subseteq \{v \in V: v \text{ reachable } A_k\}$$



$$\underline{A_n} = \{v \in V: SW(v, n) < SW(v, n-1)\}$$

$$d(s, v) = -\infty \quad \forall v \in A_n$$





$O(mn)$ time algo neg. edge lengths
Bellman-Ford

Q: Let G be directed, real-valued edge weights.
 $s, t \in V$

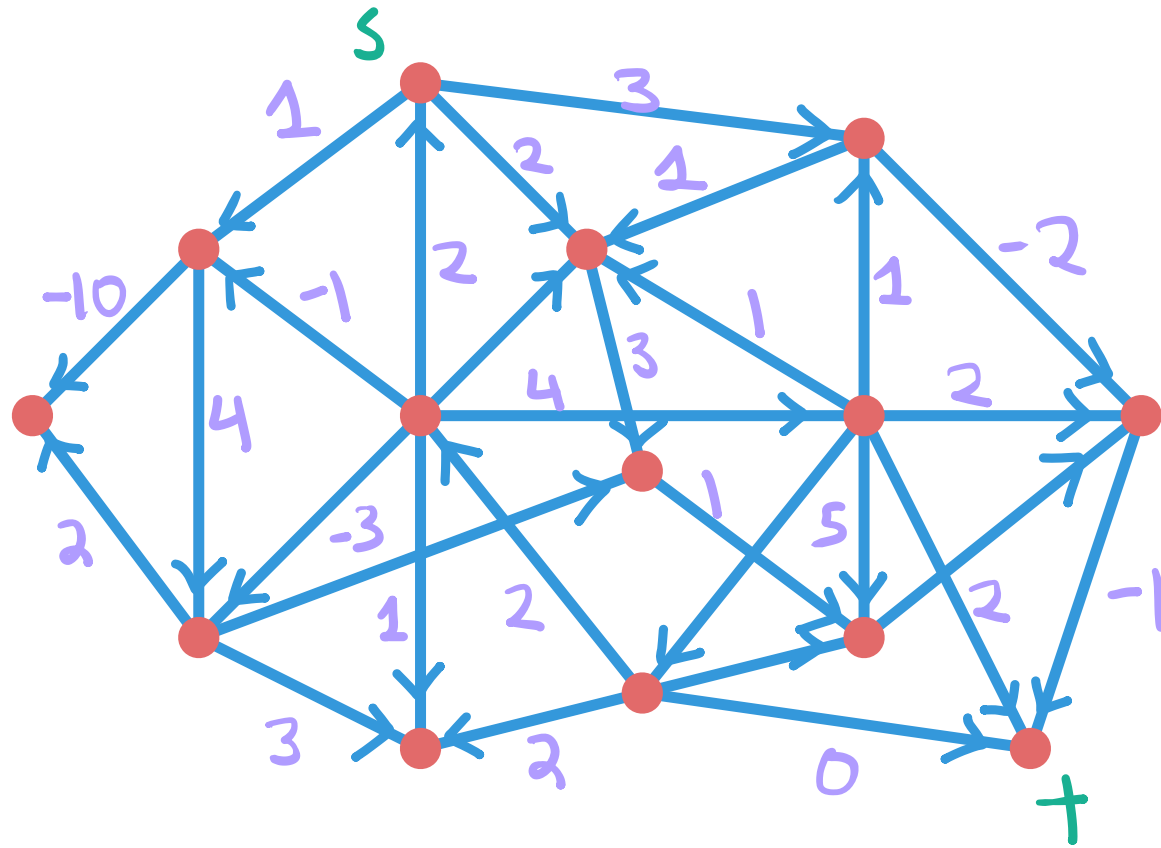
goal: length of the shortest path from
 s to t .

hard. (s, t) -hamiltonian path



$-(n-1) \Rightarrow$ hamiltonian path.

All pairs shortest paths

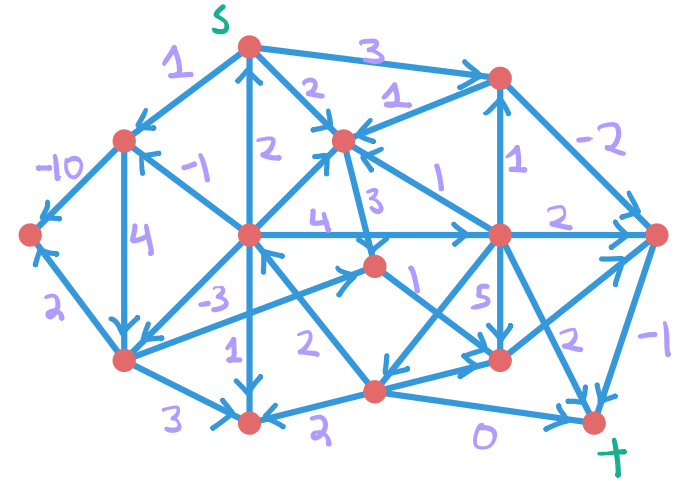


compute $d(s,t)$ for all s,t

assume no negative cycles (for simplicity)

All pairs shortest paths

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



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

All pairs shortest paths

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

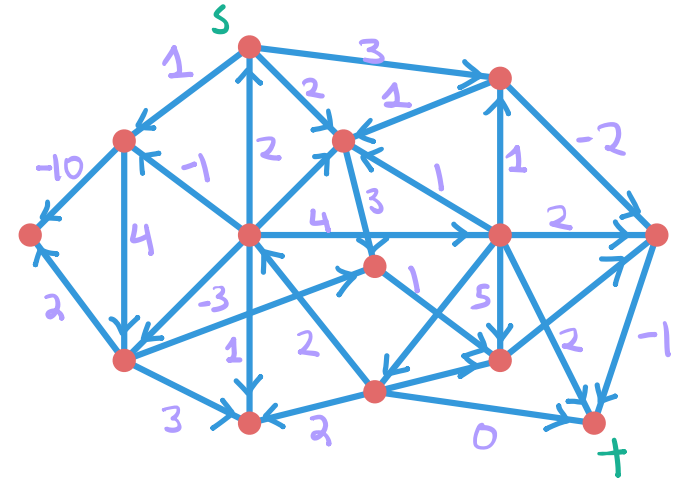
$\underbrace{SW(s, t, 2^i)}_{\text{doubling trick}}$

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. $SW(s, t, 2^{i-1})$
- B. $SW(s, v, 2^{i-1}) + SW(v, t, 2^{i-1})$ for all $v \in V$.



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

$O(n^3)$ per 2^i .
 $O(\log n)$ choices of i .

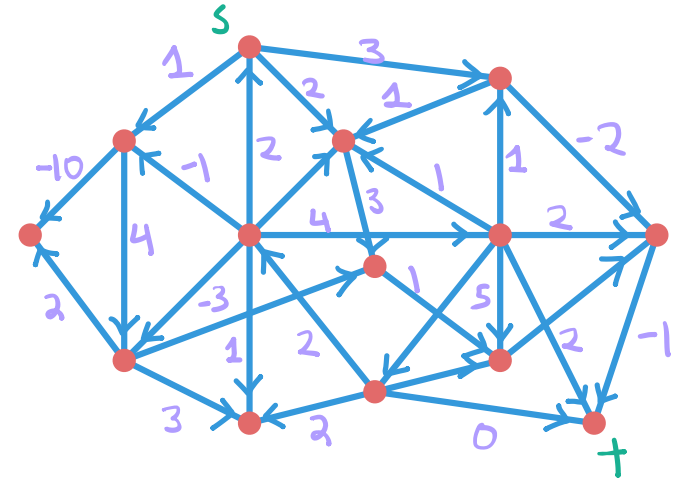
$O(n)$
 $O(n^3 \log n)$

All pairs shortest paths

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

$$SW(s, t, 2^i)$$

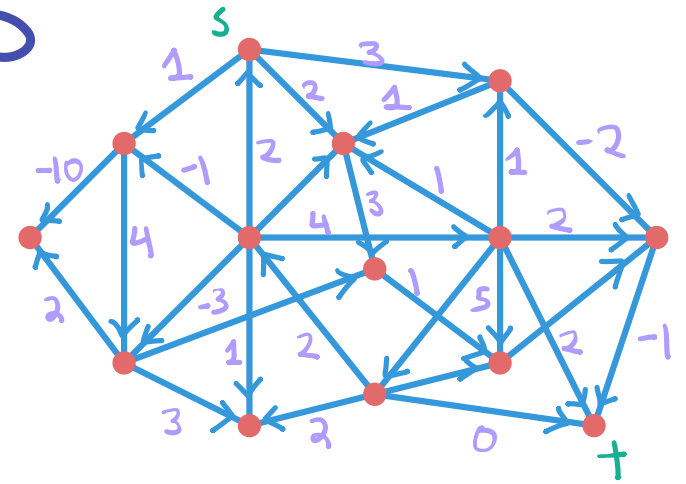
1. If $i = 0$, then return the minimum of:
 - A. 0 when $i = j$
 - B. $\ell(s, t)$ when $(s, t) \in E$.
 - C. $+\infty$.
2. Return the minimum of
 - A. $\text{SW}(s, t, 2^{i-1})$
 - B. $\text{SW}(s, v, 2^{i-1}) + \text{SW}(v, t, 2^{i-1})$ for all $v \in V$.



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

A different recursion for APSP

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



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

A different recursion for APSP

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

$$V = \{v_1, v_2, \dots, v_n\}$$

$$SP(i, j, k) \leftarrow O(n^3)$$

1. If $k = 0$:

A. If $i = j$ return 0.

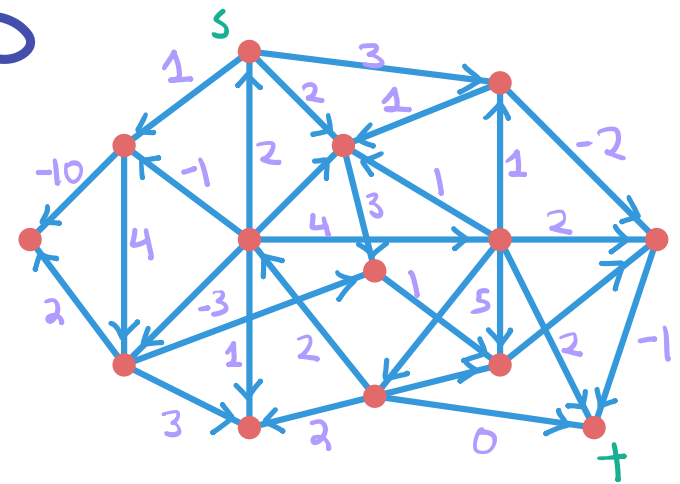
B. If $(v_i, v_j) \in E$ then return $\ell(v_i, v_j)$.

C. Otherwise return $+\infty$.

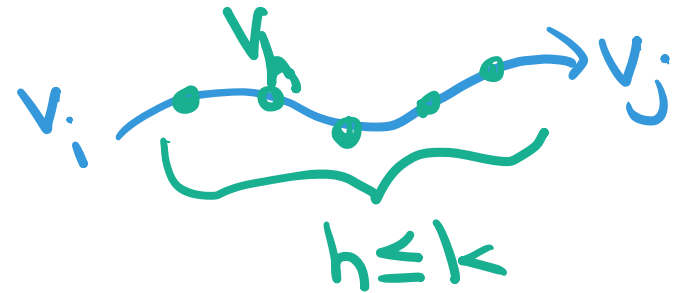
2. Return the minimum of:

A. $SP(i, j, k-1)$

B. $SP(i, k, k-1) + SP(k, j, k-1)$



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



$O(1)$

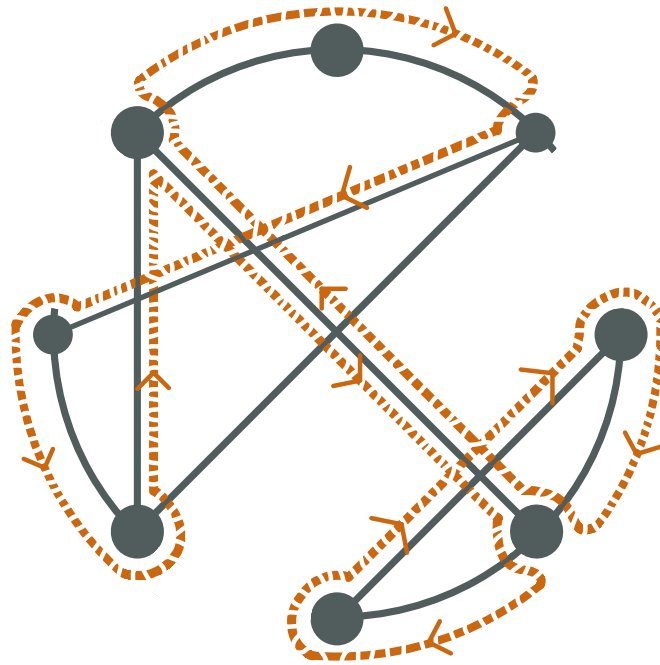
$\Rightarrow O(n^3)$

Floyd-Warshall

Traveling Salesman Problem

Input: weighted graph $G=(V,E)$

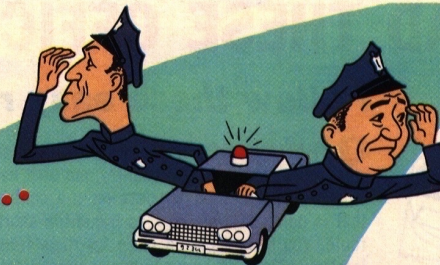
Goal: tour all the vertices w/ minimum total cost



Directed graph: "Asymmetric TSP"

Undirected graph "Metric TSP"

HELP! WE'RE LOST!



HELP "CAR 54"...AND WIN CASH

**54...\$1,000 PRIZES
ONE...\$10,000 GRAND PRIZE**



Help Toody and Muldoon find the shortest round trip route to visit all 33 locations shown on the map.

All you do is draw connecting straight lines from location to location to show the shortest round trip route.

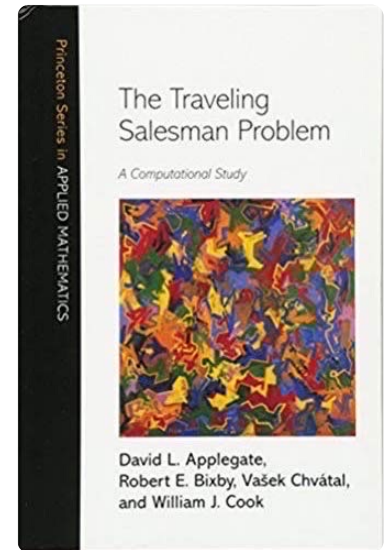
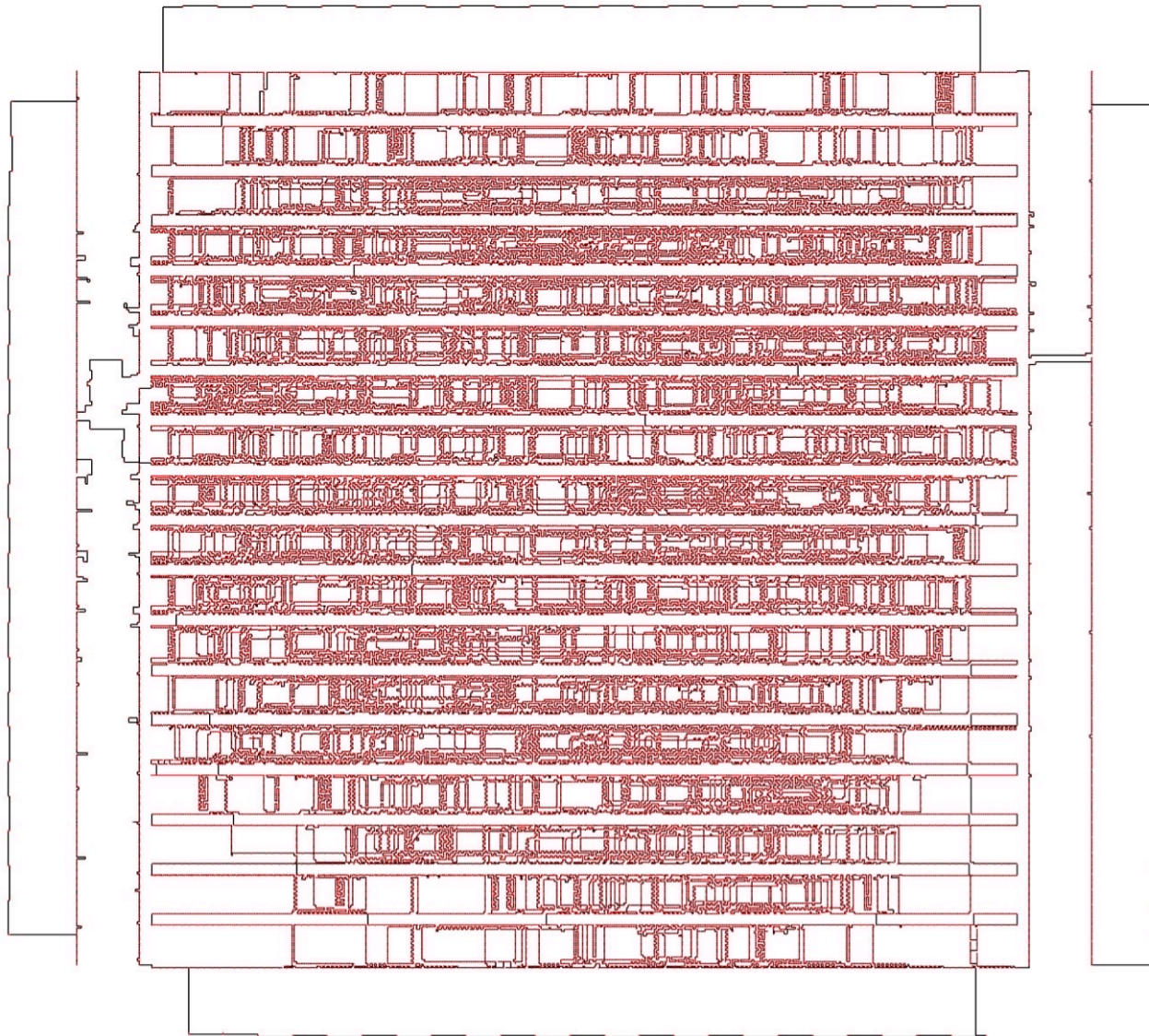
HERE'S THE CORRECT START...

Begin at Chicago, Illinois. From there, lines show correct route as far as Erie, Pennsylvania. Next, do you go to Carlisle, Pennsylvania or Wana, West Virginia? Check the easy instructions on back of this entry blank for details.



VLSI

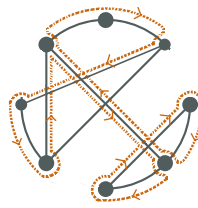
$n = 85,900$



Traveling Salesman Problem

Input: weighted graph $G=(V,E)$

Goal: tour all the vertices w/ minimum total cost



Theorem 9.9. *A polynomial time algorithm for the (directed) traveling salesman problem implies a polynomial time algorithm for SAT.*

Hamiltonian path
Hamiltonian cycle
↓
TSP

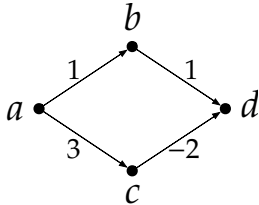


Chapter 9

Negative edge lengths and all-pairs shortest paths

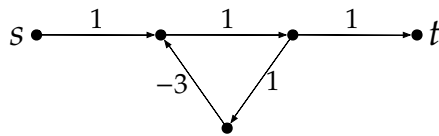
Previously we discussed shortest path problems, where the goal is to get from point a to point b as fast as possible. Here the points refer to vertices in a directed graph; the routes are the walks through the graph, and “fast” was based on the sum of edge lengths along the walk. Specifically, we obtained 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 below.



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 $\ell : E \rightarrow \mathbb{R}$. Fix $s, t \in V$. What is the length of the shortest walk from s to t ? We note that the length can possibly 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 .



In passing, let us test the reader with another closely question. Assuming the same input as before - a directed graph with real-valued edge weights and $s, t \in V$. The goal is to compute the length of the shortest *path* from s to t . Is there an algorithm for this problem? How or why?

9.1 Negative edges, shortest walks, and shortest paths

Recall the definition of distances in a directed graph (page 119). 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 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*

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. 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 non negative, 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. ■

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

9.2 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 Here there is a clear induction argument on k – the k edge walks build on $k - 1$ edge walks with an extra edge.

$SW(s, t, k)$

1. If $k = 0$:

/ The only zero edge walk is the empty walk. */*

A. If $s = t$ then return 0.

B. Otherwise return $+\infty$.

2. Return the minimum of:

A. $SW(s, t, k - 1)$.

B. The minimum of $SW(s, v, k - 1) + \ell(v, t)$ over all edges $(v, t) \in \partial^-(t)$.

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 $O(1 + \deg^-(t))$ time. For fixed s and k , it takes $O(m + n)$ time to compute all of the subproblems for a fixed value of k .

Lemma 9.2. *Let $G = (V, E)$ be a directed graph, 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 9.1, the (s, t) -distance is attained by a path for every vertex t .

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

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 $t \in V$.*

Stationary points and $-\infty$. For $k \in \mathbb{N}$, let

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

be the set of vertices v where $\text{SW}(s, t, k)$ decreases compared to the previous iteration for $k-1$. 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 :

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

In particular, 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 $\text{SW}(s, t, \ell)$ never changes. They are *fixed point* in this process. Consequently $d(s, t) = \text{SW}(s, t, k-1)$ 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 than 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 9.1, 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 $-\infty$. Meanwhile, if a vertex v is not reachable from A_n , then (since $\text{SW}(s, v, \ell)$ would never change for $\ell \geq n$) we have $\text{SW}(s, v, n) = d(s, v)$. All together, we now have all the finite distances from s as well as all the vertices with distance $-\infty$. Funny enough, we only had to churn $\text{SW}(s, t, n-1)$ for *just one more iteration* to get all the distances, finite or not.

Theorem 9.4. *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 [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 vertex s^* with an edge (s^*, v) of length 0 for every $v \in V$. In particular s^* can reach every vertex directly. If we apply theorem 9.4 from s^* (that is, compute $\text{SW}(s^*, t, n+1)$ 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 $-\infty$.³

Corollary 9.5. *Given a directed graph G with real-valued edge weights, in $O(mn)$ time, one can identify all vertices reachable from a negative cycle.*

9.3 All pairs shortest paths and the doubling trick

We now move on a related problem of computing the distances between *all pairs of vertices*, not just 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 ??).

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

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

Before we had s fixed and varied t ; now we are interested in all choices of $s, t \in V$.

Consider the case $k = 4$. When s was fixed we computed $\text{SW}(s, t, k)$ based on the values of $\text{SW}(s, v, k-1)$ for $v \in V$. Now that we're varying s as well, we could take advantage of $\text{SW}(u, v, \ell)$ for any $u, v \in V$ and $\ell \leq k$. In particular, any (s, t) -path of at most 4 edges can be divided into two paths 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 powers of 2, consider the following alternative implementation for SW.

³Don't change the algorithm, change the graph.

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

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. $\text{SW}(s, t, 2^{i-1})$
 - B. $\text{SW}(s, v, 2^{i-1}) + \text{SW}(v, t, 2^{i-1})$ for all $v \in V$. // $O(n)$ time.

The key difference between this implementation of $\text{SW}(a, n, d)$ Bellman-Ford is in (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 9.6. *Let $G = (V, E)$ be a directed graph with n vertices. In $O(n^3 i)$ time, one can compute $\text{SW}(s, t, 2^j)$ for all $s, t \in V$ and all $j \leq i$.*

If there is no negative cycles (as we have assumed) then $\text{SW}(s, t, n - 1)$ (or any value greater than $n - 1$) gives the distance from s to t . For $k = \lceil \log n \rceil$, lemma 9.6 gives us the distances between all pairs of vertices.

Corollary 9.7. *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 .

9.4 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:

$SP(i, j, k)$ = 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.

$SP(i, j, k)$

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. $SP(i, j, k - 1)$
 - B. $SP(i, k, k - 1) + SP(k, j, k - 1)$

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 $SP(i, j, k)$ for all i, j, 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 $SP(i, j, k)$ 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 $SP(i, j, k - 1)$. If it does, then the part from v_i 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 , $SP(i, k, k - 1)$ returns the length of the shortest path of the first type, and $SP(k, j, k - 1)$ returns the length of the longest path of the second type. Their sum gives the length of the shortest (v_i, v_j) path through $\{v_1, \dots, v_k\}$.

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

Theorem 9.8. *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 this approach to handle negative edges, called Johnson's algorithm [Joh77], that we do not discuss.

9.5 The traveling salesman problem

Let $G = (V, E)$ be a directed graph with positive edge weights⁴. A **tour** of G is a walk, starting and ending at the same vertex⁵, that visits every vertex at least once. The **traveling salesman problem** is to compute the minimum length tour.

The traveling salesman problem is extremely useful as a prototypical planning and routing problem (broadly construed). It has inspired a number of techniques both in theory and in practice, and also captured the popular imagination. See <https://www.math.uwaterloo.ca/tsp/history/index.html> for more on the history of this problem.

The problem turns out to be difficult, as it is a thinly disguised Hamiltonian path problem. Indeed, let $G = (V, E)$ be a directed graph in which we want to find a Hamiltonian path. Let us add a new vertex s^* with edges (s^*, v) and (v, s^*) to and from s^* , for every $v \in V$. Call this new graph G^* .

Each Hamiltonian path maps to a Hamiltonian cycle in G^* by connecting the beginning and end through s^* . The inverse mapping takes a Hamiltonian cycle in G^* and produces a Hamiltonian path by splitting the cycle at s^* .

Finally, we point out that a Hamiltonian cycle is a tour of G^* with $n^* = n + 1$ edges. So a TSP solver can find a Hamiltonian cycle in G^* , hence a Hamiltonian path in G .

Theorem 9.9. *A polynomial time algorithm for the (directed) traveling salesman problem implies a polynomial time algorithm for SAT.*

9.6 Exercises

Exercise 9.1. Recall that in foreign exchange, different currencies can be exchanged at 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

⁴One could also consider negative edge weights; we restrict to positive edge weights for simplicity.

⁵Such a walk is sometimes called a *circuit*, but should not be confused with logical circuits that have been discussed elsewhere.

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. For this problem, either (a) design and analyze a polynomial time algorithm, or (b) prove that a polynomial time algorithm would imply a polynomial time algorithm for SAT.

(One might also consider the problem where one can start at any currency X .)