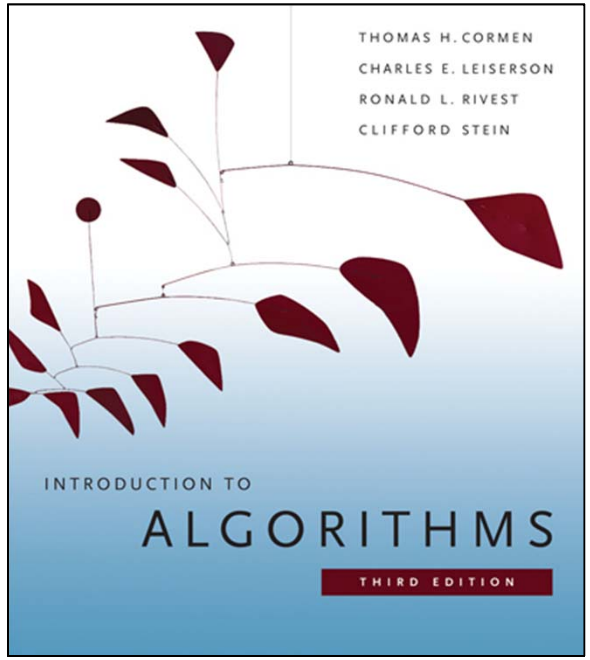
6.006

Introduction to Algorithms



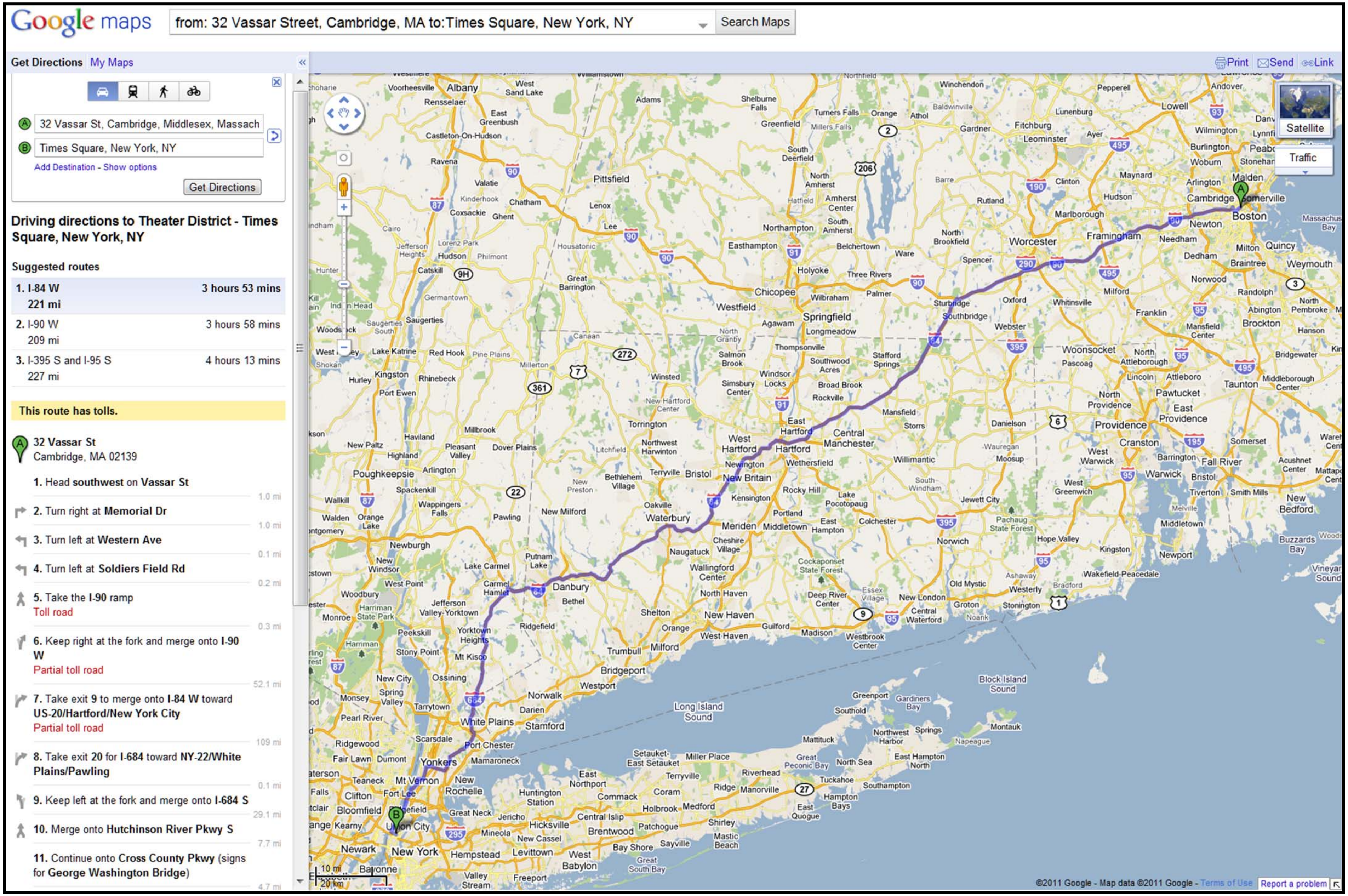
Lecture 14: Shortest Paths I

Prof. Erik Demaine

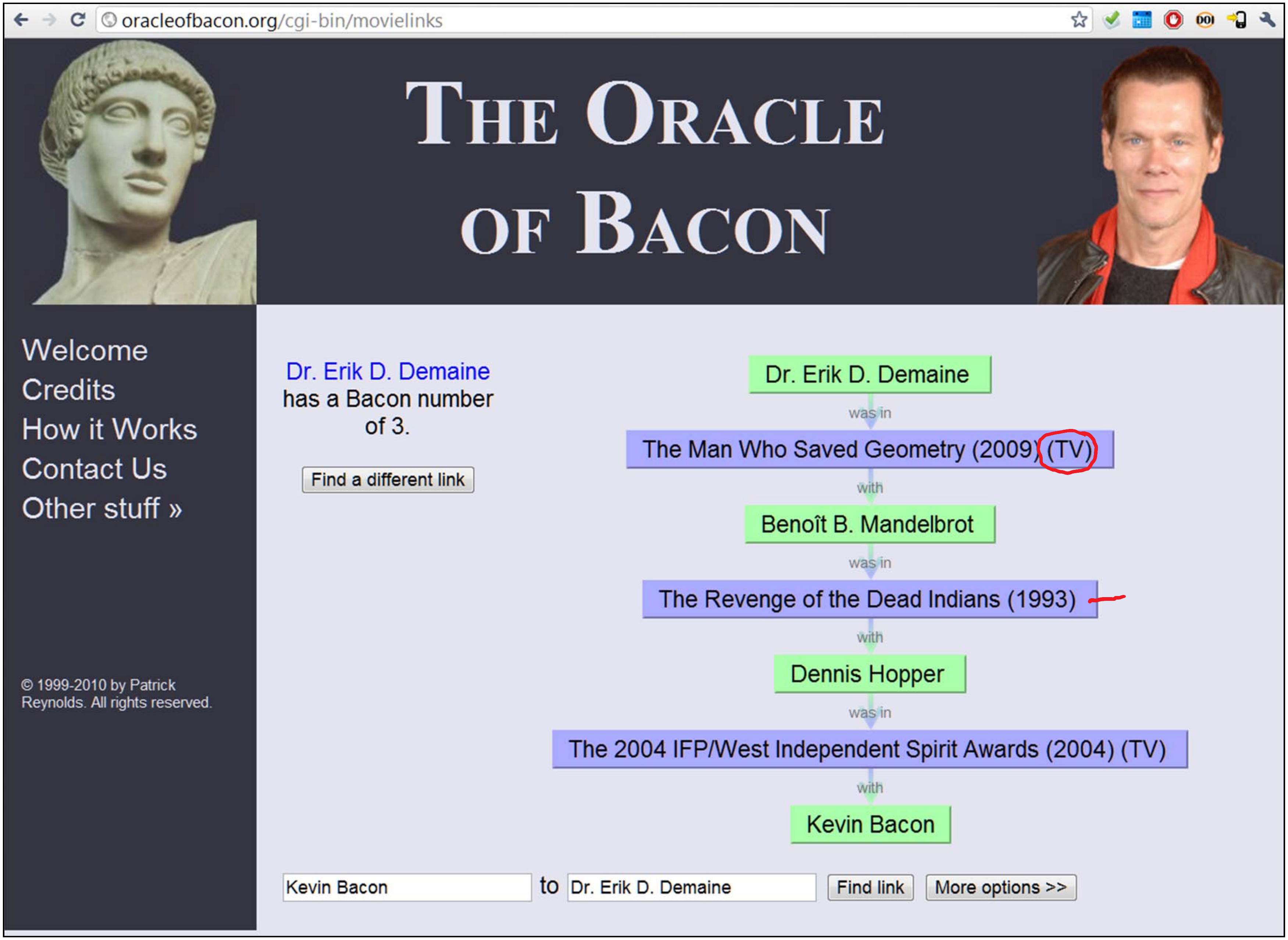
Today

- Shortest paths
- Negative-weight cycles
- Triangle inequality
- Relaxation algorithm
- Optimal substructure

Shortest Paths



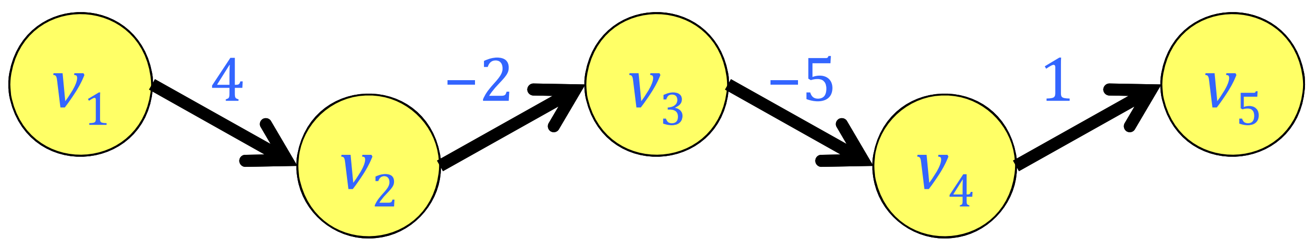
Shortest Paths



How Long Is Your Path?

- Directed graph $G = (V, E)$
- Edge-weight function $w : E \to \mathbb{R}$
- Path $p = v_1 \to v_2 \to \cdots \to v_k$
- ***Weight*** of p, denoted $w(p)$, is
  $w(v_1, v_2) + w(v_2, v_3) + \cdots + w(v_{k-1}, v_k)$

<u>Example:</u>

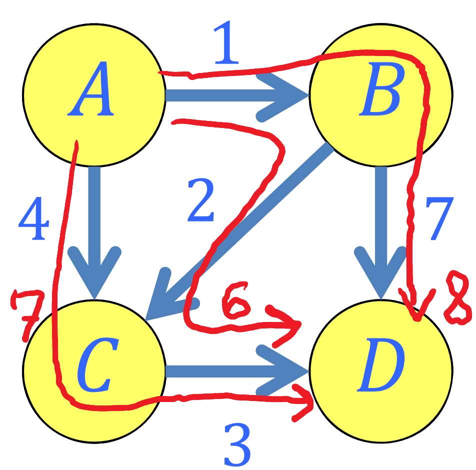


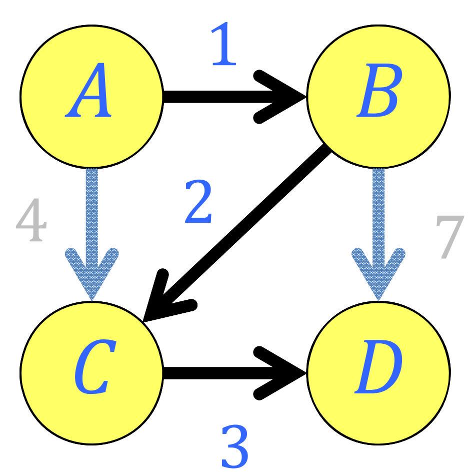
$$w(p) = 4 - 2 - 5 + 1 = -2$$

My Path Is Shorter Than Yours

- A ***shortest path*** from u to v is a path p of minimum possible weight $w(p)$ from u to v
- The shortest-path weight $\delta(u,v)$ from u to v is the weight of any such shortest path:
  $\delta(u,v) = \min\{w(p) : p \text{ is a path from } u \text{ to } v\}$

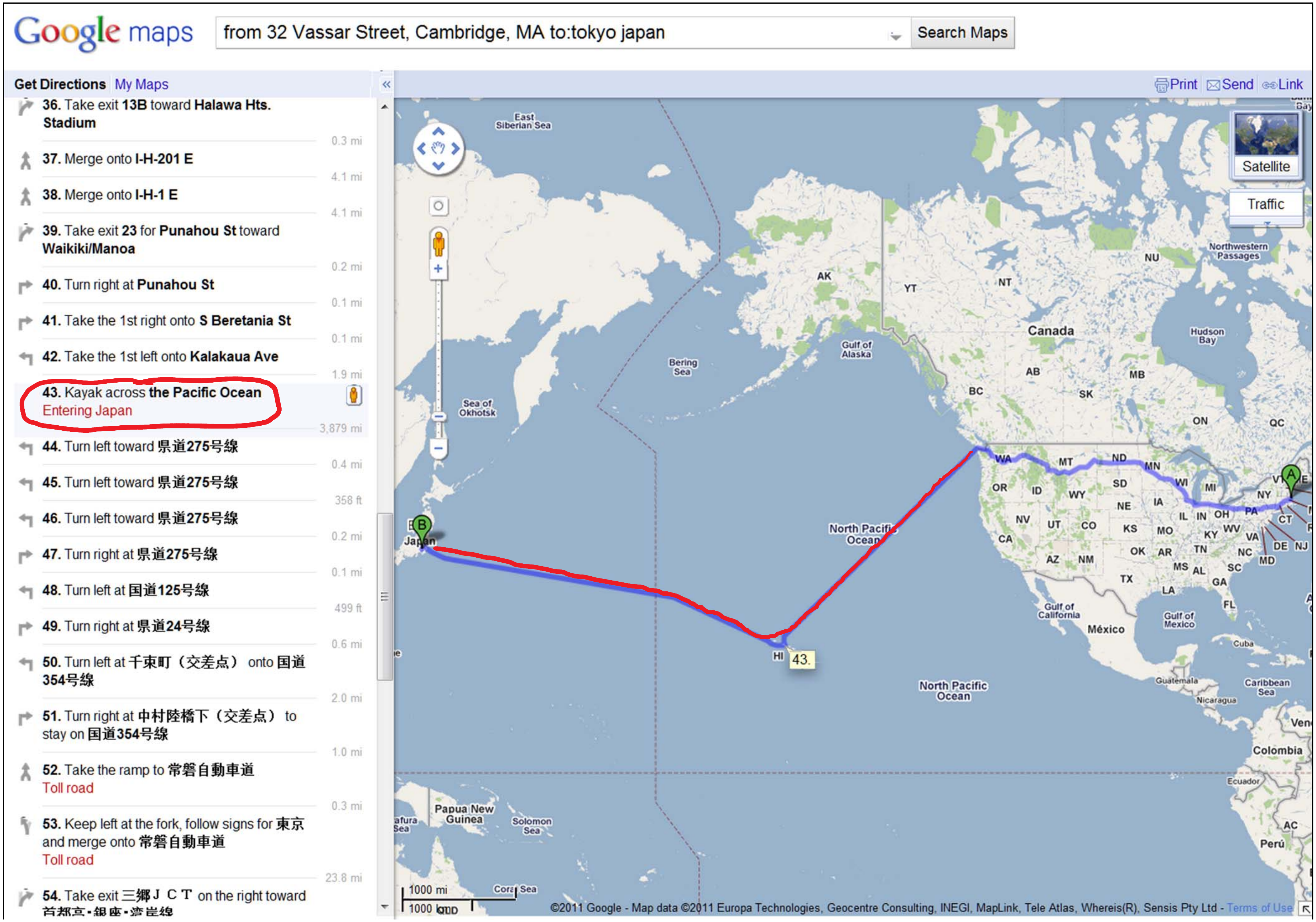
<u>Example:</u>





$\delta(A,D)$
$= 6$

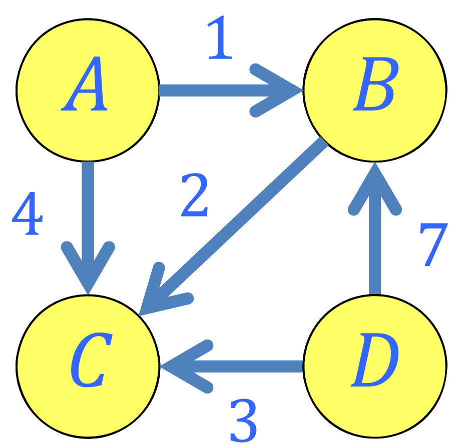
You Can't Get There From Here



You Can't Get There From Here

- If there is no path from u to v, then neither is there a *shortest* path from u to v
- Define $\delta(u, v) = \infty$ in this case

Example:



$\delta(A, D) = \infty$

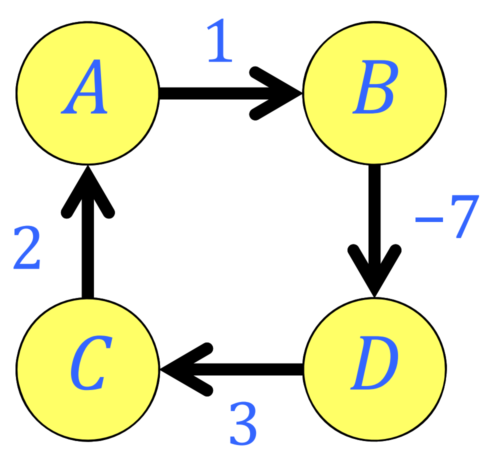
The More I Walk, The Less It Takes

- A shortest path from u to v might not exist, even though there is a path from u to v
- ***Negative-weight cycle***

$$c = v_1 \to v_2 \to \cdots \to v_k \to v_1$$

has $\quad w(c) < 0$

Example:

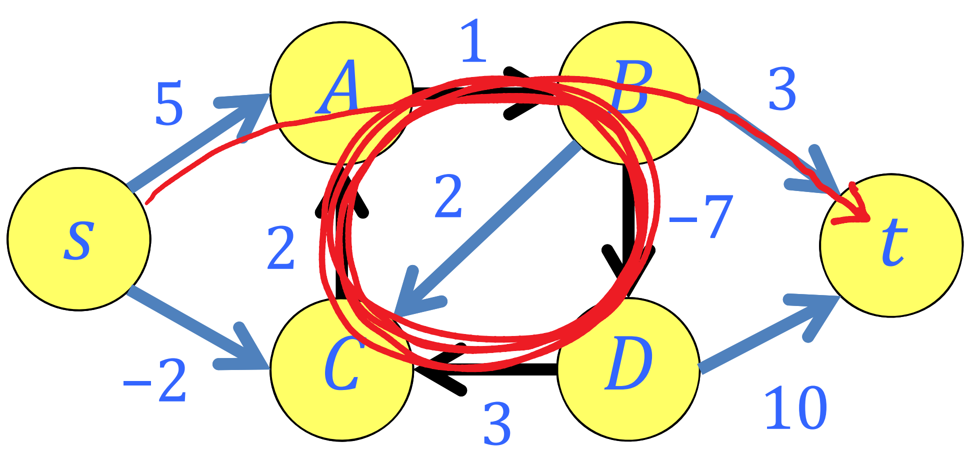


$$w(A \to B \to C \to D \to A) = -1$$

The More I Walk, The Less It Takes

- Define $\delta(u, v) = -\infty$ if there's a path from u to v that visits a negative-weight cycle
- $\delta(u, v) = \inf\{w(p) : p \text{ is a path from } u \text{ to } v\}$

<u>Example:</u>

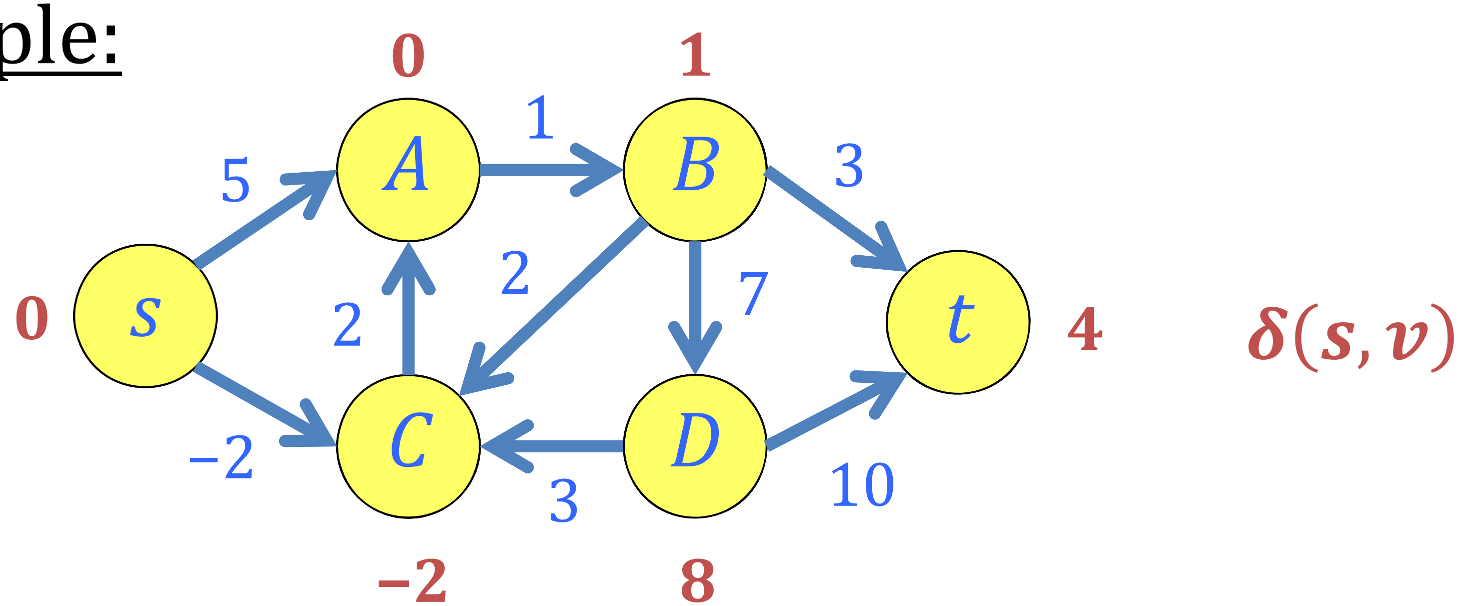


$\delta(s, t) = -\infty$

Single-Source Shortest Paths

- <u>Problem:</u> Given a directed graph $G = (V, E)$ with edge-weight function $w : E \to \mathbb{R}$, and a ***source*** vertex s, compute $\delta(s, v)$ for all $v \in V$

<u>Example:</u>

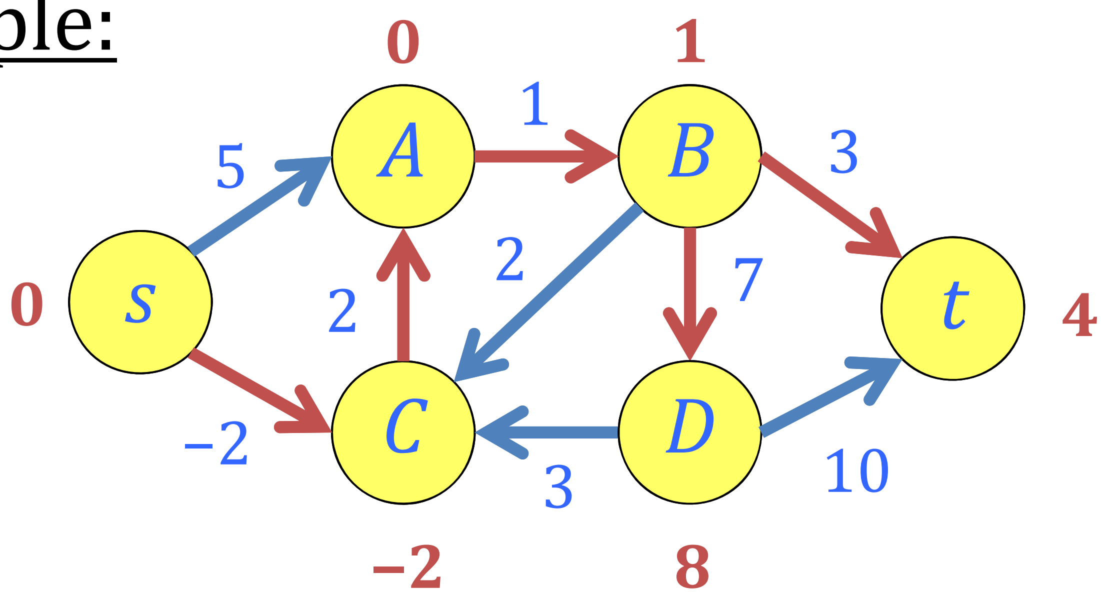


Shortest-Path Tree

- Ideally also compute a ***shortest-path tree*** containing a shortest path from source s to every $v \in V$ (assuming shortest paths exist)
  - Represent by storing ***parent*** $v.\pi$ for each $v \in V$

$= \pi[v]$ in earlier CLR(S)

<u>Example:</u>

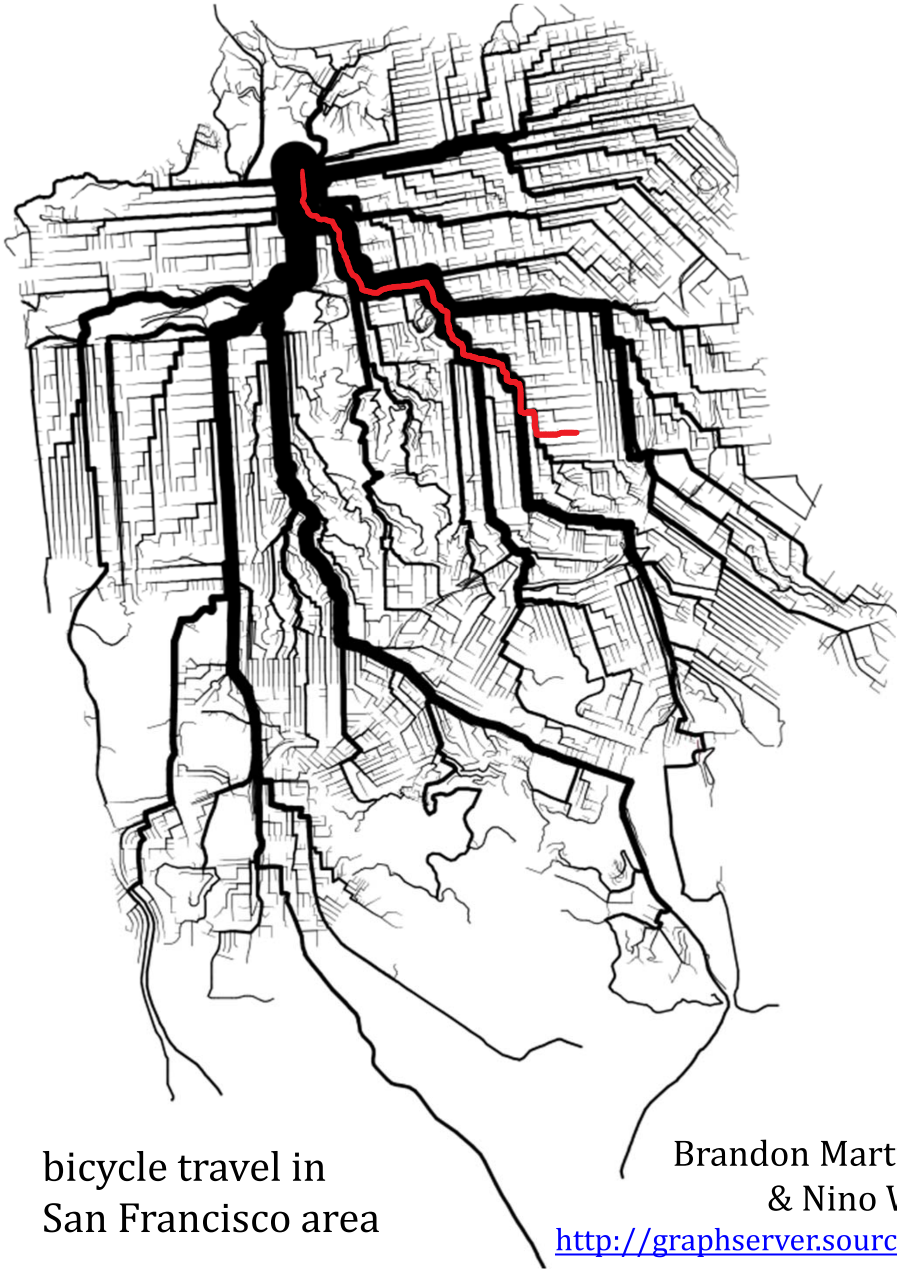


$t.\pi = B$
$B.\pi = A$
$D.\pi = B$

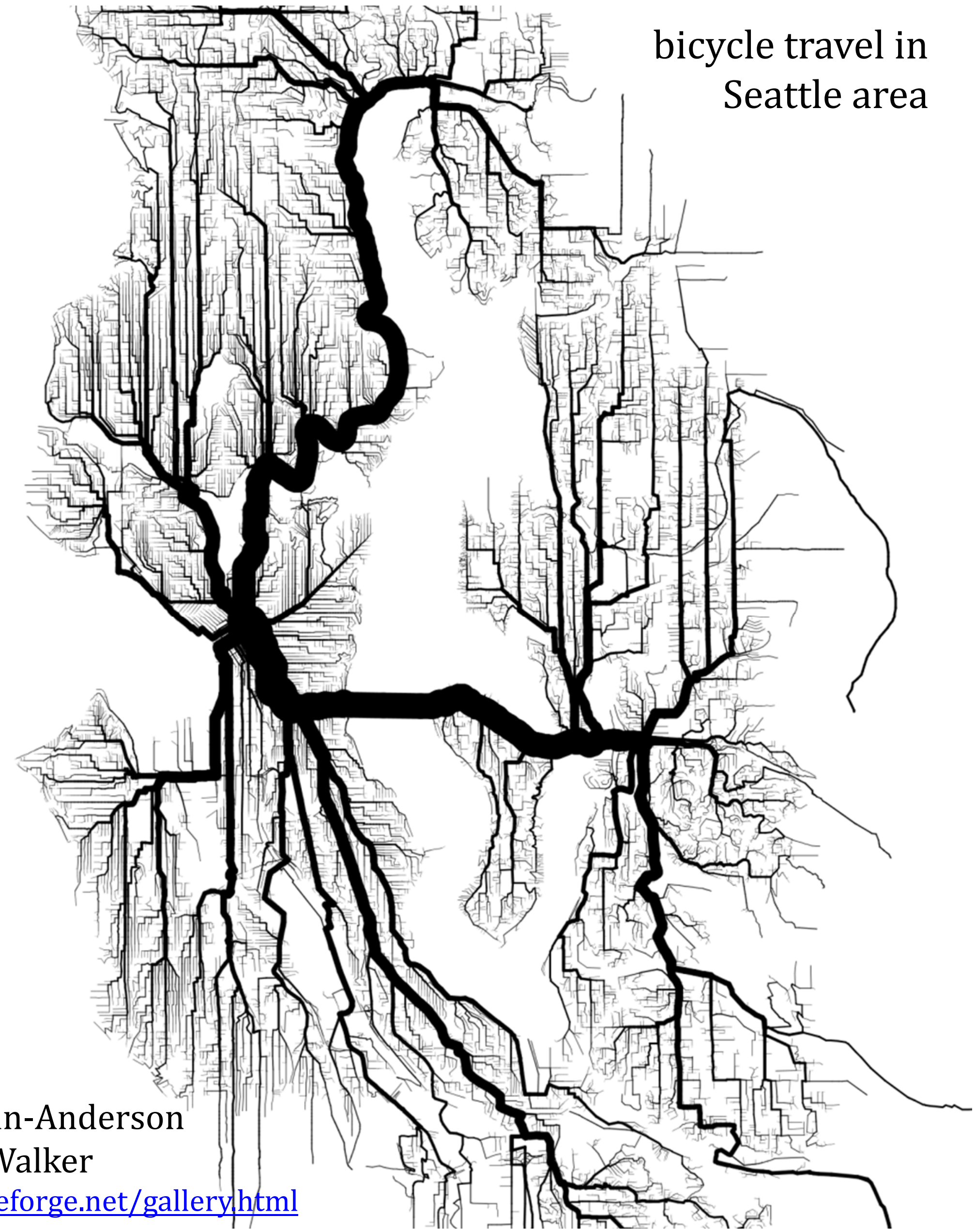
$\delta(s, v)$

$A.\pi = C$
$C.\pi = s$
$s.\pi = \text{None}$

Shortest-Path Trees



bicycle travel in
San Francisco area



bicycle travel in
Seattle area

Brandon Martin-Anderson
& Nino Walker
http://graphserver.sourceforge.net/gallery.html

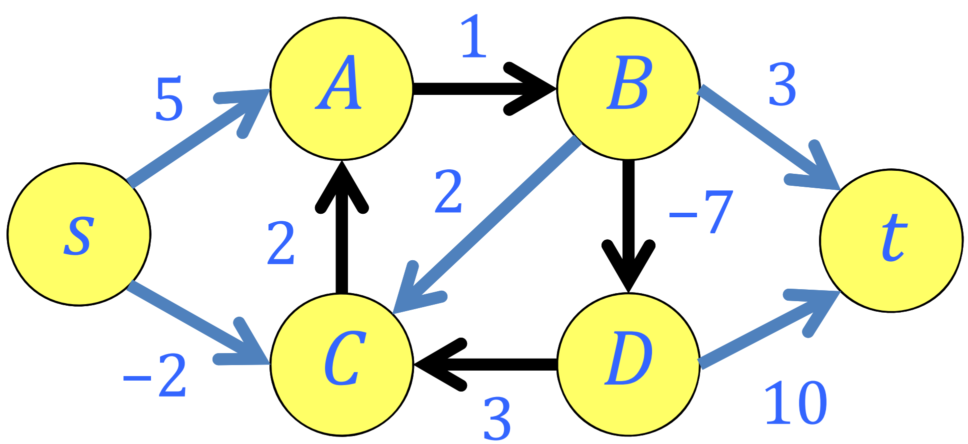
Single-Source Shortest-Path Algorithms

- **Relaxation algorithm** *(TODAY)*
  - Framework for most shortest-path algorithms
  - Not necessarily efficient
- **Bellman-Ford algorithm** *(LECTURE 15)*
  - Deals with negative weights
  - Slow but polynomial
- **Dijkstra's algorithm** *(LECTURE 16)*
  - Fast (nearly linear time)
  - Requires nonnegative weights

Brute-Force Algorithm

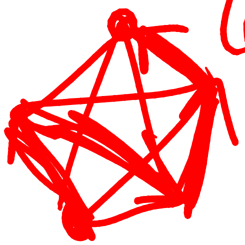
distance(s, t):
  for each path p from s to t:
    compute $w(p)$
  return p encountered with smallest $w(p)$

- Number of paths can be *infinite*:



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

$(n-2)! \sim n^n \approx 2^{n \lg n}$



Brute-Force Algorithm

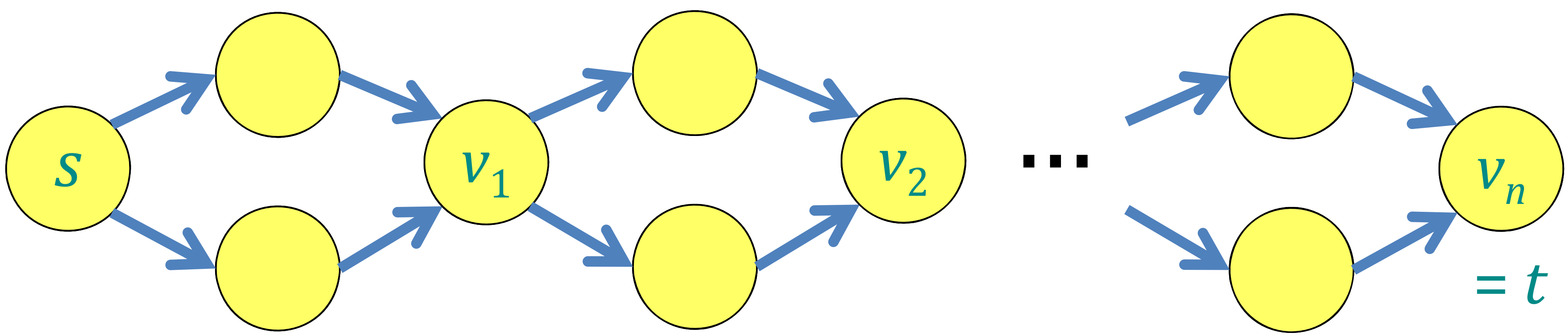
distance(s, t): *## assume no negative-weight cycles*

  for each **simple** path p from s to t: ← # paths

    compute $w(p)$ ← $O(V)$

  return p encountered with smallest $w(p)$

- Number of paths can be *exponential*:



2^n paths from s to v_n; $O(n)$ vertices and edges

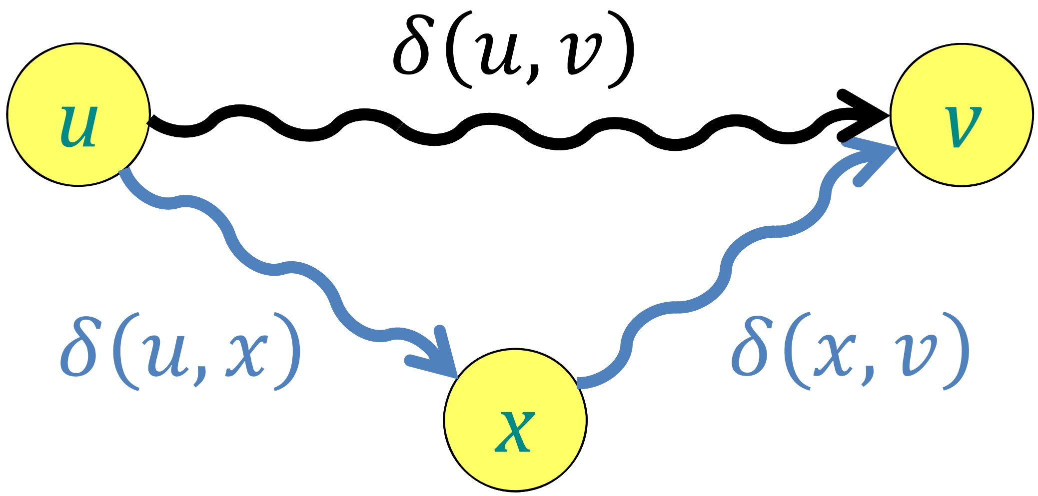
Relaxation

- In general, refers to letting a solution (temporarily) violate a constraint, and trying to fix these violations

Magic Geek
http://www.youtube.com/watch?v=Y12daEZTUYo

Triangle Inequality

- Theorem: For all $u, v, x \in V$, we have
$$\delta(u, v) \leq \delta(u, x) + \delta(x, v).$$



- Proof: Shortest path from u to v is at most any particular path, e.g., the blue chain. ■

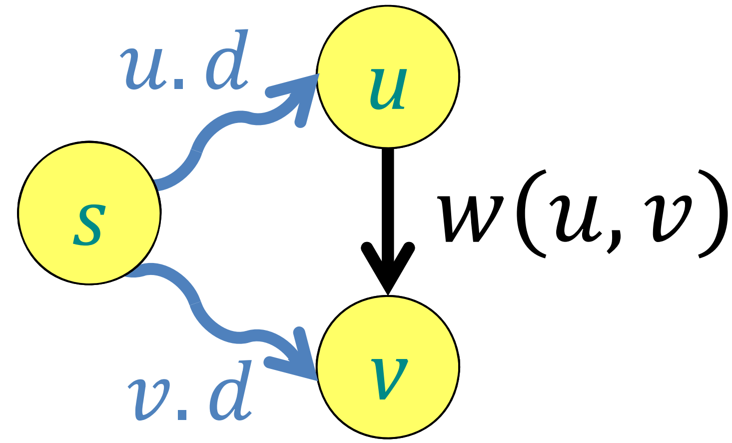
Relaxation Approach

- Maintain ***distance estimate*** $v.d$ = d[v] in older CLR(S) for each $v \in$ V
- Goal: $v.d = \delta(s, v)$ for all $v \in V$
- Invariant: $v.d \geq \delta(s, v)$
- Initialization:

$$\boxed{\begin{array}{l} \text{for } v \text{ in } V: \\ \qquad v.d = \infty \\ s.d = 0 \end{array}}$$

- Repeatedly improve estimates toward goal, by aiming to achieve triangle inequality

Edge Relaxation



- Consider an edge (u, v)

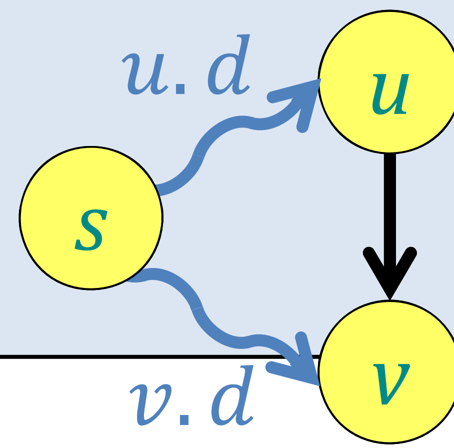
- $\delta(s, v) \leq \delta(s, u) + \delta(u, v)$ [triangle ineq.]
  $\leq \delta(s, u) + w(u, v)$ [candidate path]

$\Rightarrow$ want $v.d \leq u.d + w(u, v)$

relax(u, v):
  if $v.d > u.d + w(u, v)$:
    $v.d = u.d + w(u, v)$

Relaxation Algorithm

for v in V:
        $v.d = \infty$
$s.d = 0$
while some edge (u, v) has $v.d > u.d + w(u, v)$:
        pick such an edge (u, v)
        relax(u, v):
                if $v.d > u.d + w(u, v)$:
                        $v.d = u.d + w(u, v)$



Relaxation Algorithm with Shortest-Path Tree

for v in V:

    $v.d = \infty$

    $v.\pi = \text{None}$

$s.d = 0$

while some edge (u, v) has $v.d > u.d + w(u, v)$:

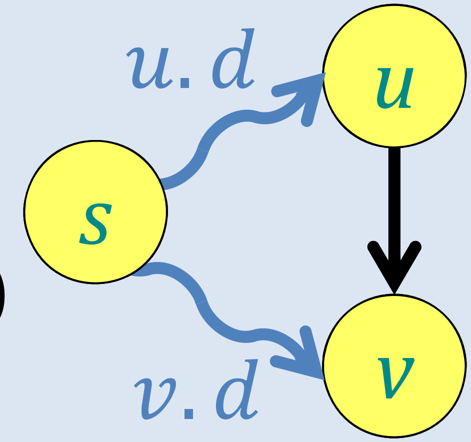
    pick such an edge (u, v)

    relax(u, v):

        if $v.d > u.d + w(u, v)$:

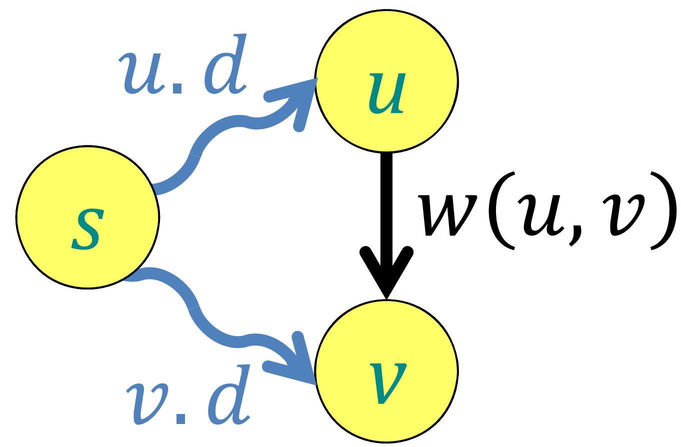
            $v.d = u.d + w(u, v)$

            $v.\pi = u$



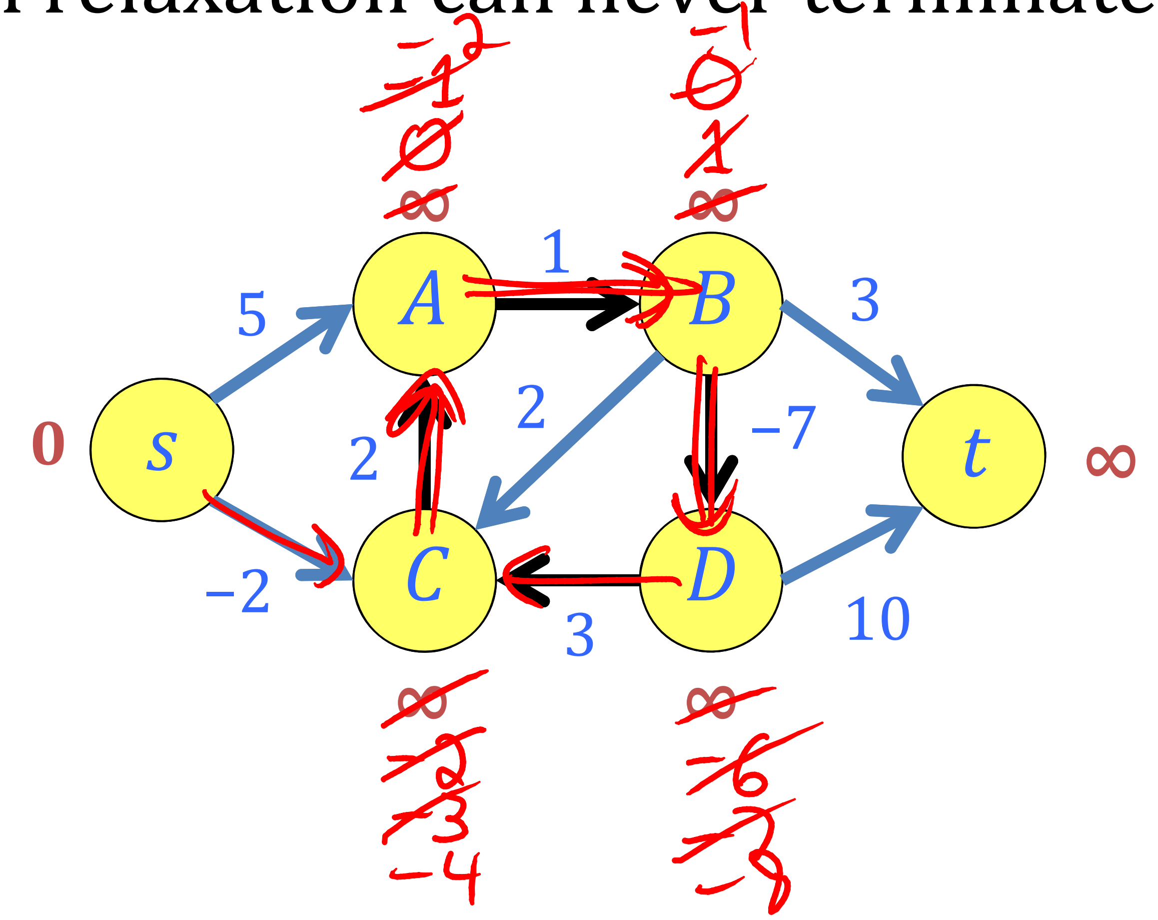
Relaxing Is Safe

- Lemma: The relaxation algorithm maintains the invariant that $v.d \geq \delta(s,v)$ for all $v \in V$.
- Proof: By induction on the number of steps.
  - Consider relax(u,v)
  - By induction, $u.d \geq \delta(s,u)$
  - By triangle inequality,
$$\begin{aligned}\delta(s,v) &\leq \delta(s,u) + \delta(u,v)\\ &\leq u.d + w(u,v)\end{aligned}$$
  - So setting $v.d = u.d + w(u,v)$ is "safe" ∎

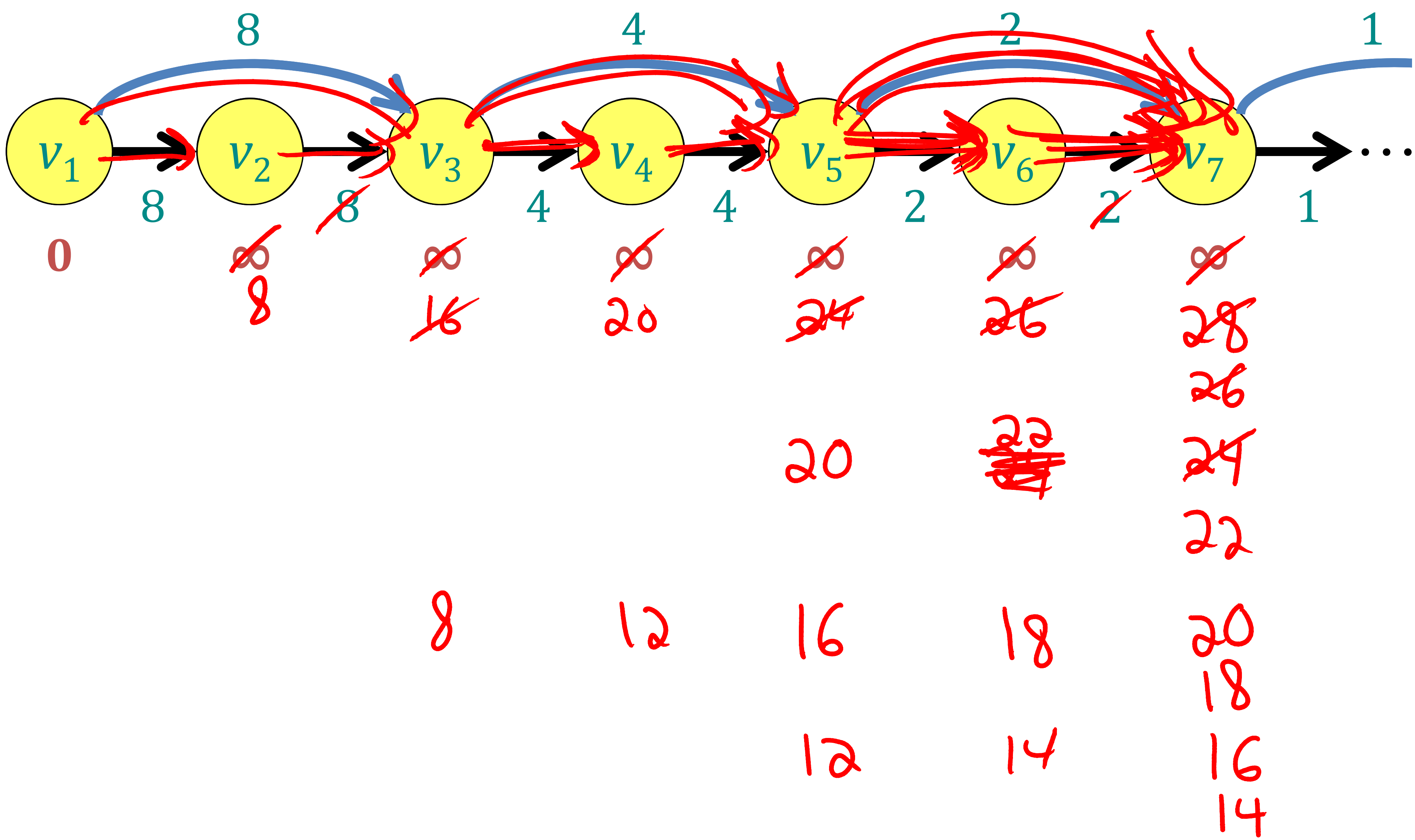


Infinite Relaxation

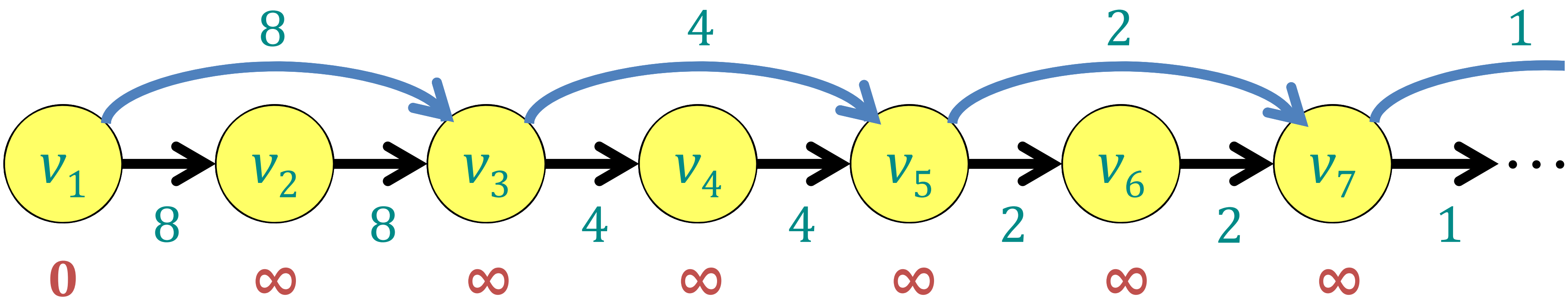
- If a negative-weight cycle is reachable from s, then relaxation can never terminate



Long Relaxation



Long Relaxation



- Analysis:
  - relax(v_1, v_2) – 1
  - relax(v_2, v_3) – 2
  - recurse on $v_3, v_4, \dots, v_n$ } $T(n-2)$
  - relax(v_1, v_3) – 3
  - recurse on $v_3, v_4, \dots, v_n$ } $T(n-2)$

$$T(n) = 2\,T(n-2) + 3$$

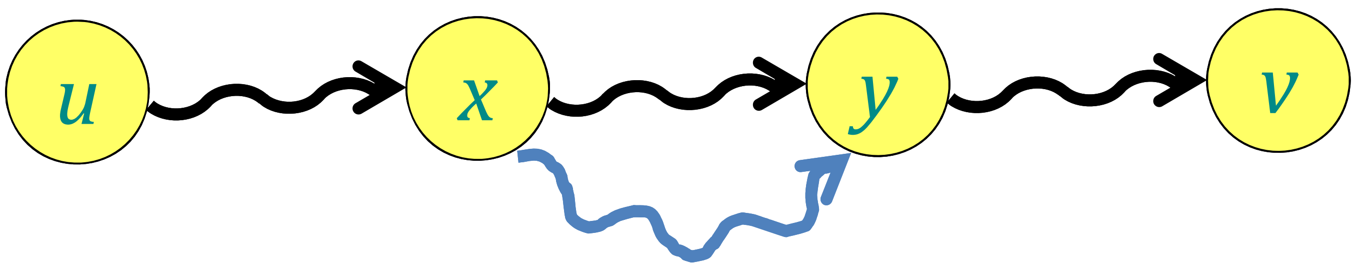
$$T(n) = \Theta(2^{n/2})$$

Are You Sure This Is a Good Idea?

- **Bellman-Ford algorithm:** *(LECTURE 15)*
  - Relax all of the edges
  - Repeat $\sim|V|$ times
  - Polynomial time!
- **Dijkstra's algorithm:** *(LECTURE 16)*
  - Relax edges in a growing ball around s
  - Nearly linear time!
  - (but doesn't work with negative edge weights)

Optimal Substructure

- Lemma: A subpath of a shortest path is a shortest path (between its endpoints).



- Proof: By contradiction.
  - If there were a shorter path from x to y, then we could **shortcut** the path from u to v, contradicting that we had a shortest path. ■