6.006	Lecture 12	Oct. 23, 2018

TODAY: Weighted Shortest Paths (intro.)
- negative-weight cycles
- relaxation meta-algorithm
- subpath property
- triangle inequality
- safety lemma
- termination lemma
- SSSP in DAGs (directed acyclic graphs)

Recall: [L10 on BFS]

(Unweighted) shortest path from $u \in V$ to $v \in V$
= a path of minimum length from u to v
(among all paths from u to v)
- length of such a path is:
- shortest-path length $\delta(u,v)$

$$\begin{cases} \min\{\ell(p) \mid \text{path } p \text{ from } u \text{ to } v\} \\ \infty \quad \text{if no path from } u \text{ to } v \end{cases}$$

Shortest-paths problems:
- single-pair: $\delta(s,t)$, and such a path
- single-source: $\delta(s,v)$ for all $v \in V$, and shortest-path tree containing a shortest path from s to every $v \in V$
- all-pairs: $\delta(u,v)$ for all $u,v \in V$ [L15]

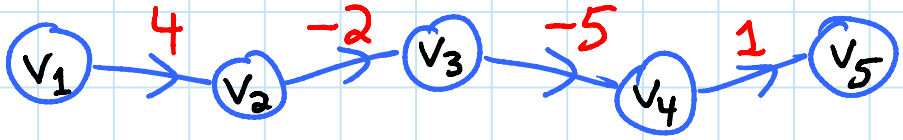
Motivation:

- Google Maps (fastest/shortest way A → B)
- network routing (fastest way to route packet)
- not all edges equal cost/weight to traverse

Edge-weighted graph = directed graph $G=(V,E)$
+ **edge-weight function** $w: E \to \mathbb{R}$
  → $w(e)$ is weight of edge e
- representations: - $W[u][v]$ via dict/array of dicts
  - store weight within Adj

Weight of a path $p = v_1 \to v_2 \to \cdots \to v_k$ is
$$w(p) = w(v_1, v_2) + w(v_2, v_3) + \cdots + w(v_{k-1}, v_k)$$

Example:



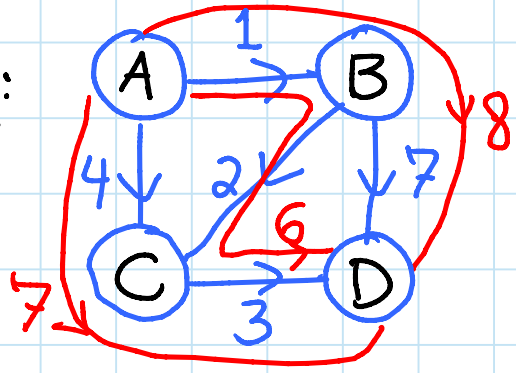
$$w(p) = 4 - 2 - 5 + 1 = -2$$

(Weighted) shortest path from $u \in V$ to $v \in V$
= a path of minimum weight from u to v
(among all paths from u to v)
- weight of such a path is:
- **shortest-path weight** $\delta(u,v)$

$$\begin{cases} \inf\{w(p) \mid \text{path } p \text{ from } u \text{ to } v\} \\ \infty \quad \text{if no path from } u \text{ to } v \end{cases}$$

see next page

Example:



$A \to B \to C \to D$

$\delta(A, D) = 6$

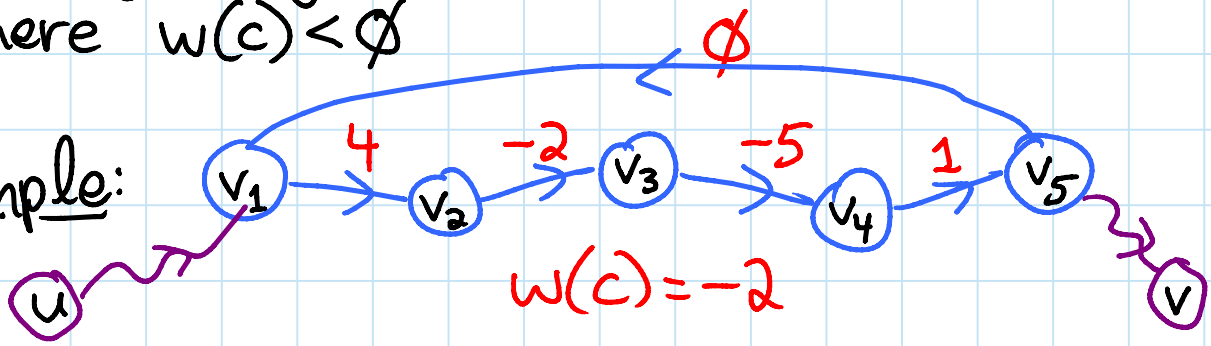
$\delta(B, A) = \infty$

- even if there's a path from u to v, there may not be a shortest path:

Negative-weight cycle where $w(c) < \emptyset$ (↷ path starting & ending @ same vertex)

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

Example:

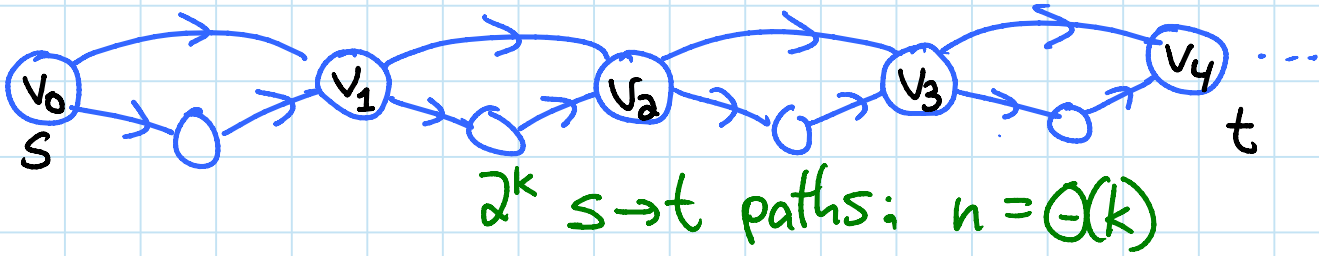


- $\delta(u, v) = -\infty$ if there's a negative-weight cycle along a path from u to v

Brute-force algorithm:

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

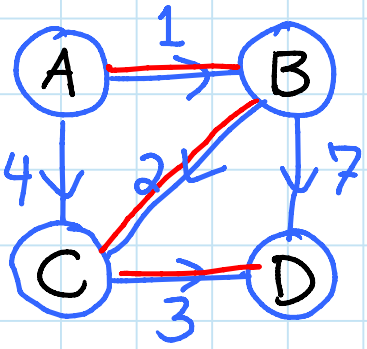
- infinitely many paths ($\Rightarrow$ infinite time)
  if $\delta(s,t) = -\infty$ (i.e. neg.-weight cycle on the way)
- even when finite, often exponential # paths:
  - $(n-2)!$ in complete graph
  - $2^{\Theta(n)}$ even in directed acyclic graph (DAG):



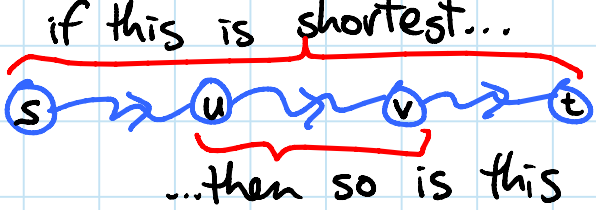
SSSP algorithm		Setting	Time
BFS [L10]		$w(e) = 1\ \forall e$	$O(V+E)$
DAG shortest paths	TODAY	G acyclic	$O(V+E)$
Dijkstra [L14]		$w(e) \geq 0\ \forall e$	$O(V \lg V + E)$
Bellman-Ford [L13]		general	$O(VE)$

Why can shortest paths from s always form a tree?

Example:



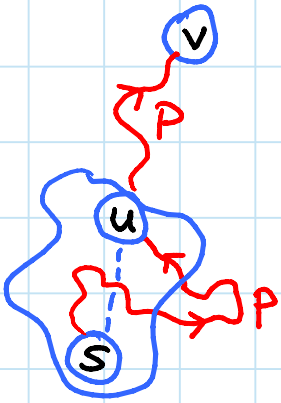
Subpath property: subpaths of shortest paths are shortest paths



Proof: cut & paste
- shorter $u \to v$ path would make a shorter $s \to t$ path, contradiction. □

Tree property: there's a tree T rooted at s containing a shortest $s \to v$ path for all $v \in V$ that have such a path

Proof: start with just s in tree



- for $v \in V$ with finite $\delta(s,v)$:
  - find a shortest path p from s to v
  - find last vertex $u \in T$ visited by p (e.g. $s \in T$)
  - add suffix of p to T
  - $s \overset{T}{\rightsquigarrow} u \overset{p}{\rightsquigarrow} v$ still shortest path □

shortest by induction
⇒ same weight as $s \overset{p}{\rightsquigarrow} u$ by Subpath Prop.

Relaxation meta-algorithm for SSSP:

framework for most shortest-path algorithms

- maintain distance estimate $d[v]$ for each $v \in V$
- goal: $d[v] = \delta(s,v)$ for all $v \in V$
- invariant: $d[v] \geq \delta(s,v)$ for all $v \in V$
- repeatedly improve estimates until reach the goal

① initialize: $d = \{s: 0, \text{rest}: \infty\}$ ← satisfies invariant
(as in BFS) $\text{parent} = \{s: s\}$

② pick an edge $(u,v) \in E$

③ relax(u,v):
if $d[v] > d[u] + w(u,v)$:
$d[v] = d[u] + w(u,v)$
$\text{parent}[v] = u$

④ repeat until done
↳ no relaxations will change d
(see Termination Lemma below)

Safety Lemma: relax(u,v) maintains invariant $d[v] \geq \delta(s,v)$
⇒ remains true throughout proof

Stronger invariant: for all $v \in V$, $d[v] =$ weight of some $s \to v$ path
$\Rightarrow \geq \delta(s,v)$ (inf of all such paths)

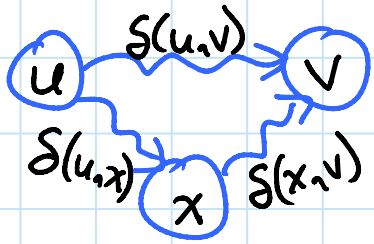
Proof: whenever we change $d[v]$, we have a path: $s \overset{d[u]}{\rightsquigarrow} u \overset{w(u,v)}{\to} v$ □

Terminology:

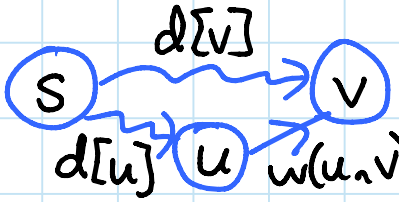
- "relaxation" refers to letting a solution (temporarily) violate a constraint, and trying to fix these violations
- for shortest paths, the constraint is:

Triangle inequality: $\delta(u,v) \le \delta(u,x) + \delta(x,v)$ for all $u,v,x \in V$

Proof: shortest $u \to v$ path $\le$ any particular path e.g. $u \rightsquigarrow x \rightsquigarrow v$ □



Edge relaxation:



- consider edge (u,v)
- $\delta(s,v) \le \delta(s,u) + \delta(u,v)$ — Δ inequality
  $\le \delta(s,u) + w(u,v)$ — any path ≥ shortest
- $\Rightarrow$ want $d[v] \le d[u] + w(u,v)$
- relax(u,v) checks for violation & fixes it

(and Termination Lemma $\Rightarrow$ enough checks for all of Δ ineq.)

Alternate proof of Safety Lemma:

when we change $d[v]$ to $d[u] + w(u,v)$

$\ge \delta(s,u) + w(u,v)$ — invariant before

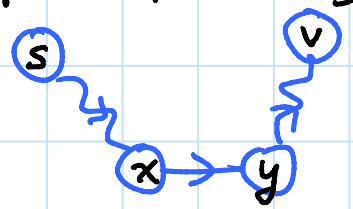
$\ge \delta(s,u) + \delta(u,v)$ — any path ≥ shortest

$\ge \delta(s,v)$ — Δ inequality □

Termination Lemma: if no edge can be relaxed, then $d[v] = \delta(s,v)$ for all $v \in V$

Proof: suppose a shorter $s \to v$ path p: $w(p) < d[v]$

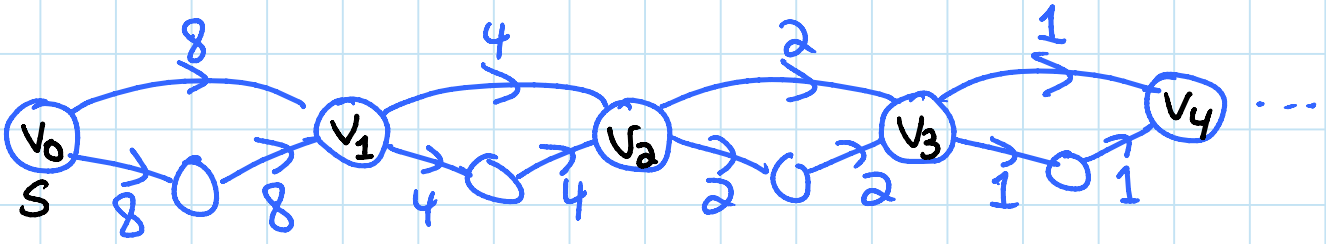
- find first edge (x,y) of p with $d[y] > \delta(s,y)$
- (x,y) can be relaxed □



Infinite relaxation: whenever negative-weight cycle reachable from source s

- relaxation continues around the cycle forever
- proof: always finite $d[v]$'s, but need $-\infty$ so can't terminate by Termination Lemma

Exponential relaxation: even in a DAG

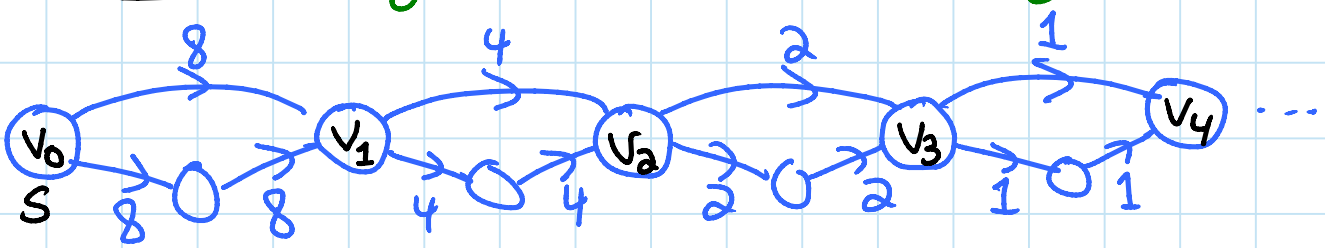


d:	∅	∞	∞	∞	∞	∞	∞	∞	∞
		8	16	20	24	26	28	29	30
									29
							26	27	28
									27
					20	22	24	25	26
									25
							22	23	24
									23
					etc.				

→ relax lower edges left → right
relax rightmost upper edge
repeat

L13: Bellman-Ford algorithm finds a polynomial-length relaxation order!

Good order: edges ordered left to right



d:	∅	∞	∞	∞	∞	∞	∞	∞	∞
	∅	8	8	12	12	14	14	15	15

Why so fast? DAG + topological sort order

Shortest paths in a DAG:

```
initialize
topologically sort G          - O(V+E)
for u in V:                   (in topo. sort order)
    for v in Adj[u]:          } deg(u)   } O(V+E)
        relax(u,v)
```

- no (negative-weight) cycles to deal with

Correctness: $d[v] = \delta(s,v)$ (when v visited by the outer loop)

Proof: consider shortest $s \to v$ path

$p = v_1 \to v_2 \to \cdots \to v_k$ $(v_1 = s,\ v_k = v)$

- algorithm relaxes edges of p in order (in addition to other edges mixed in)

$\Rightarrow d[v_i] \le d[v_{i-1}] + w(v_{i-1}, v_i)$, $d[s] = \emptyset$

$\Rightarrow d[v] \le \sum_{i=2}^{k} w(v_{i-1}, v_i) = w(p)$

- equal by Safety Lemma $\square$

Google maps

from: 32 Vassar Street, Cambridge, MA to:Times Square, New York, NY

Search Maps

Get Directions | My Maps

Print | Send | Link

32 Vassar St, Cambridge, Middlesex, Massach

Times Square, New York, NY

Add Destination - Show options

Get Directions

Driving directions to Theater District - Times Square, New York, NY

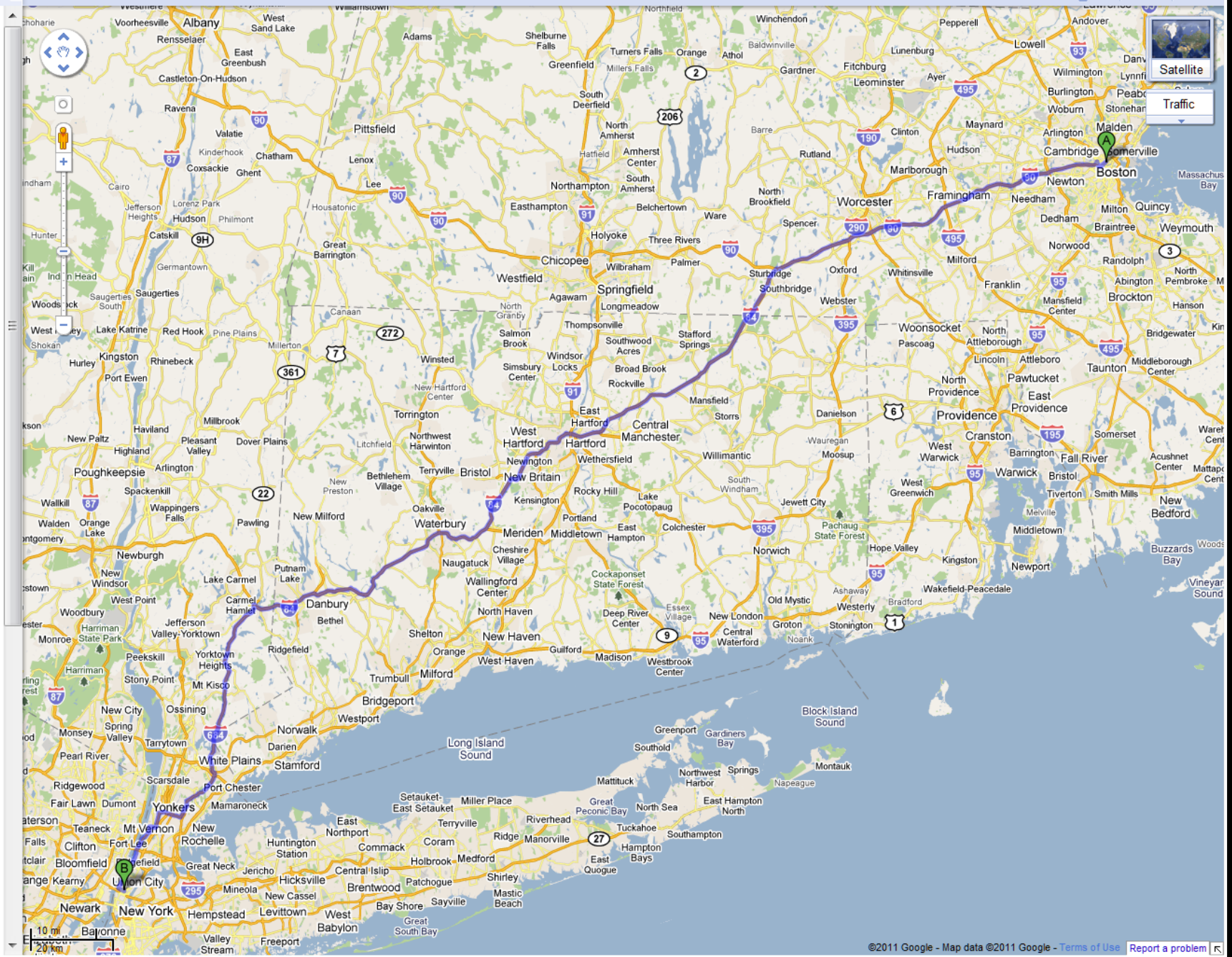
Suggested routes

1.	**I-84 W** **221 mi**	**3 hours 53 mins**
2.	I-90 W 209 mi	3 hours 58 mins
3.	I-395 S and I-95 S 227 mi	4 hours 13 mins

This route has tolls.

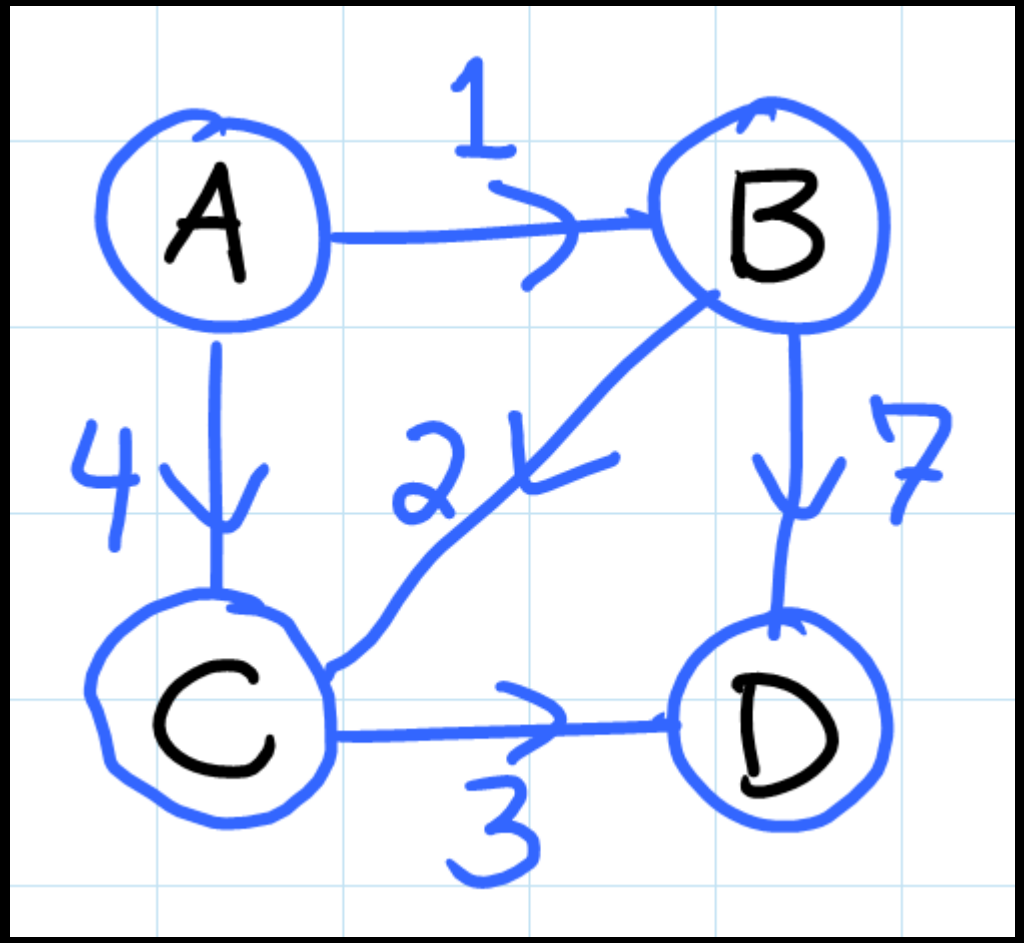
32 Vassar St
Cambridge, MA 02139

	1. Head **southwest** on **Vassar St**	1.0 mi
	2. Turn right at **Memorial Dr**	1.0 mi
	3. Turn left at **Western Ave**	0.1 mi
	4. Turn left at **Soldiers Field Rd**	0.2 mi
	5. Take the **I-90** ramp Toll road	0.3 mi
	6. Keep right at the fork and merge onto **I-90 W** Partial toll road	52.1 mi
	7. Take exit **9** to merge onto **I-84 W** toward **US-20/Hartford/New York City** Partial toll road	109 mi
	8. Take exit **20** for **I-684** toward **NY-22/White Plains/Pawling**	0.1 mi
	9. Keep left at the fork and merge onto **I-684 S**	29.1 mi
	10. Merge onto **Hutchinson River Pkwy S**	7.7 mi
	11. Continue onto **Cross County Pkwy** (signs for **George Washington Bridge**)	4.7 mi

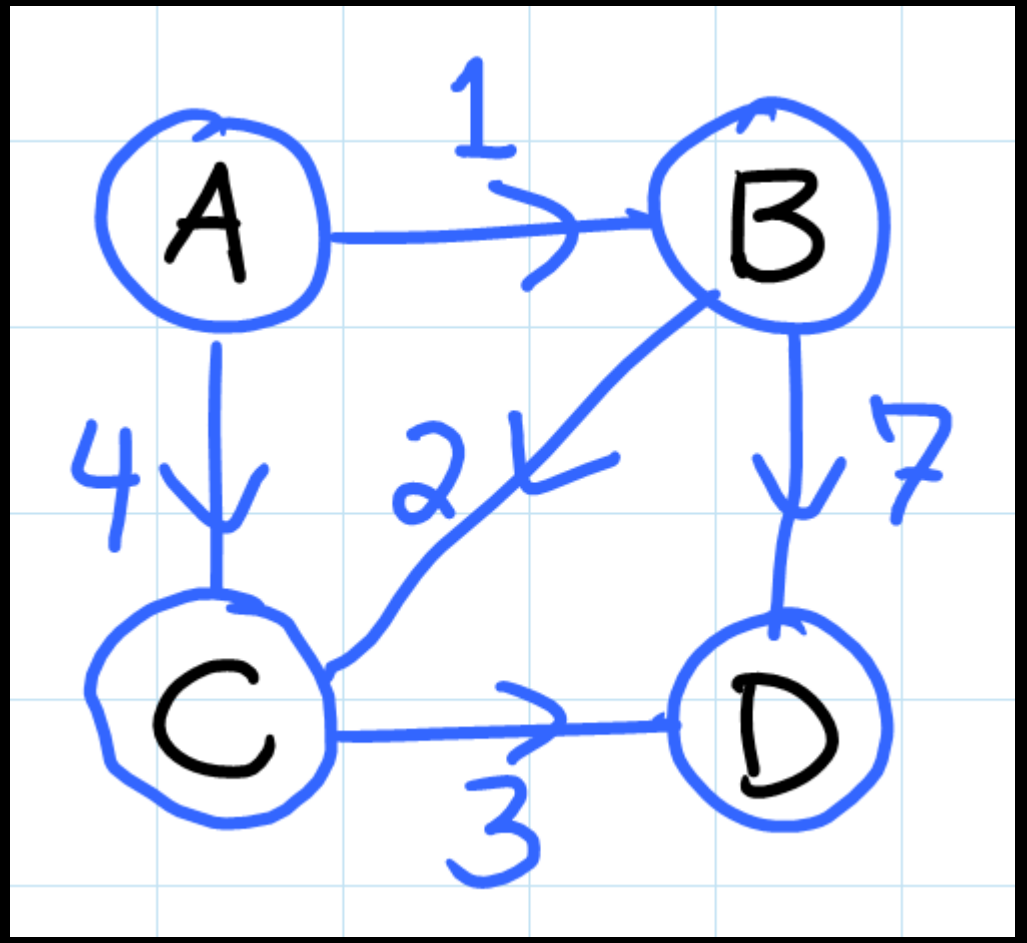


```
IPv4 Route Table
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0    192.168.3.1    192.168.3.41     281
          0.0.0.0          0.0.0.0       10.8.0.1        10.8.0.2     286
         10.8.0.0  255.255.255.252         On-link        10.8.0.2     286
         10.8.0.2  255.255.255.255         On-link        10.8.0.2     286
         10.8.0.3  255.255.255.255         On-link        10.8.0.2     286
        127.0.0.0        255.0.0.0         On-link         127.0.0.1     306
        127.0.0.1  255.255.255.255         On-link         127.0.0.1     306
  127.255.255.255  255.255.255.255         On-link         127.0.0.1     306
      169.254.0.0        255.255.0.0         On-link    169.254.112.65     276
   169.254.112.65  255.255.255.255         On-link    169.254.112.65     276
  169.254.255.255  255.255.255.255         On-link    169.254.112.65     276
      192.168.1.0    255.255.255.0       10.8.0.1        10.8.0.2      30
      192.168.3.0    255.255.255.0         On-link      192.168.3.41     281
    192.168.3.41  255.255.255.255         On-link      192.168.3.41     281
   192.168.3.255  255.255.255.255         On-link      192.168.3.41     281
        224.0.0.0        240.0.0.0         On-link         127.0.0.1     306
        224.0.0.0        240.0.0.0         On-link    169.254.112.65     276
        224.0.0.0        240.0.0.0         On-link        10.8.0.2     286
        224.0.0.0        240.0.0.0         On-link      192.168.3.41     281
  255.255.255.255  255.255.255.255         On-link         127.0.0.1     306
  255.255.255.255  255.255.255.255         On-link    169.254.112.65     276
  255.255.255.255  255.255.255.255         On-link        10.8.0.2     286
  255.255.255.255  255.255.255.255         On-link      192.168.3.41     281
===========================================================================
Persistent Routes:
  Network Address          Netmask  Gateway Address  Metric
          0.0.0.0          0.0.0.0       10.8.0.1  Default
```

```
Adj = {
    'A': ['B', 'C'],
    'B': ['C', 'D'],
    'C': ['D'],
    'D': [],
}
```



```
wAdj = {
    'A': {'B': 1,
          'C': 4},
    'B': {'C': 2,
          'D': 7},
    'C': {'D': 3},
    'D': {},
}
```



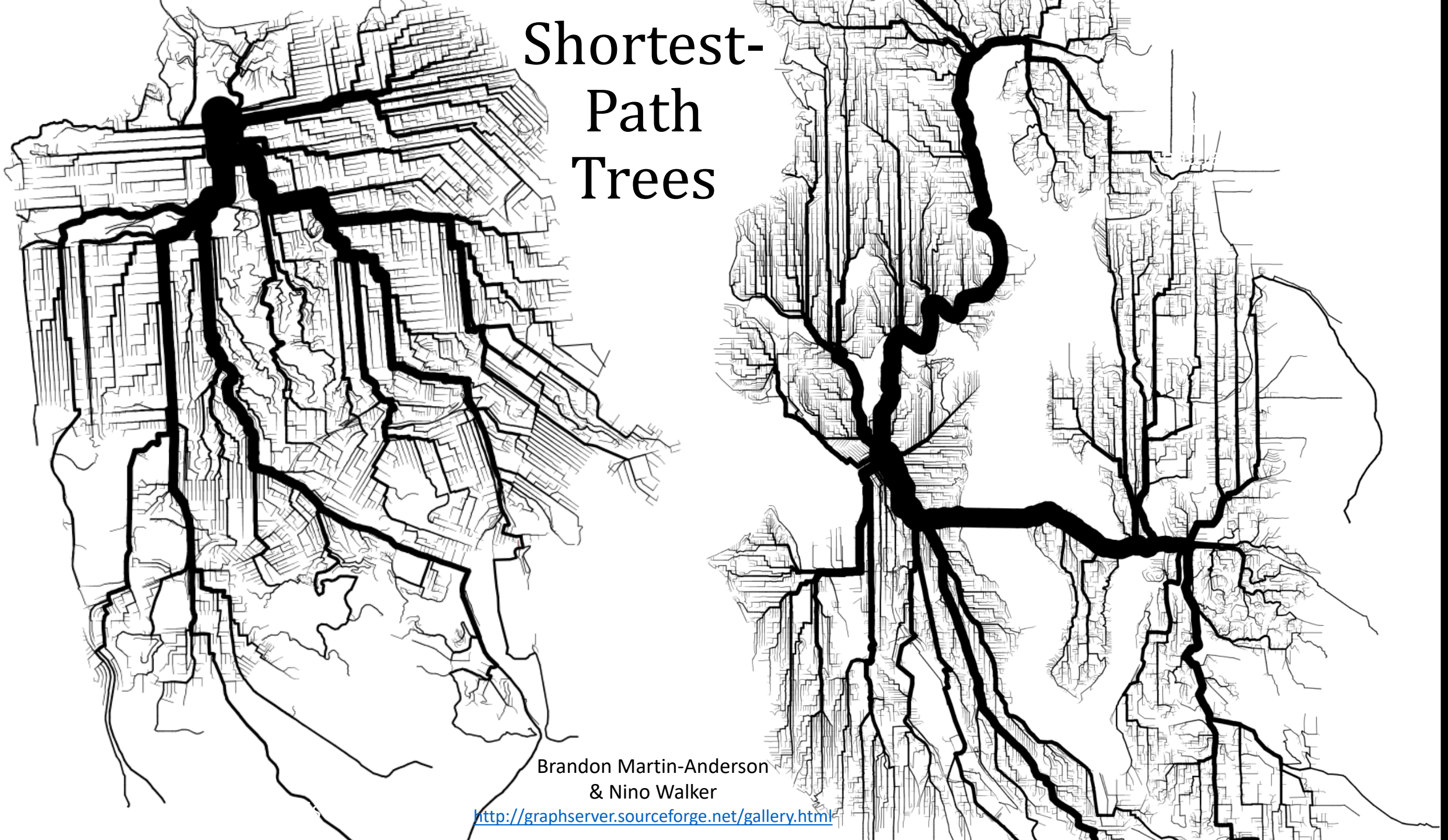
```
WAdj = {
    'A': {'B': 1,
          'C': 4},
    'B': {'C': 2,
          'D': 7},
    'C': {'D': 3},
    'D': {},
}
```

```
def Adj(u):
    return \
        WAdj[u].keys()

def w(u, v):
    return WAdj[u][v]
```

```
def weight(p, w):
    return sum(w(p[i-1], p[i])
               for i in range(1, len(p)))

def brute_force(Adj, w, s, t):
    best = math.inf
    for p in all_paths(Adj, s, t):
        best = min(best, weight(p, w))
    return best
```

SSSP Algorithm	Lecture	Setting	Running Time
BFS	10	$w(e) = 1$ for all $e \in E$	$O(V + E)$
DAG shortest paths	12	G acyclic	$O(V + E)$
Dijkstra	14	$w(e) \geq 0$ for all $e \in E$	$O(V \log V + E)$
Bellman-Ford	13	(general)	$O(VE)$

Shortest-Path Trees



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

```
def relaxation_sssp(Adj, w, s, pick_edge):
    parent = [None] * len(Adj) # shortest-path tree
    parent[s] = s              # s is root
    d = [math.inf] * len(Adj)  # d[v] = δ(s,v)
    d[s] = 0                   # δ(s,s) ≤ 0

    def relax(u, v):
        if d[v] > d[u] + w(u,v):
            d[v] = d[u] + w(u,v)
            parent[v] = u

    while True:
        u, v = pick_edge(Adj, w, s, d)
        if u is None: break
        relax(u, v)
```

```
def BFS(Adj, s): # BFS from s in graph given by Adj
    parent = [None] * len(Adj) # shortest-path tree
    parent[s] = s              # s is root
    d = [math.inf] * len(Adj)  # d[v] = δ(s,v)
    d[s] = 0                   # δ(s,s) = 0
    level = [[s]]     # level 0 = {s}
    while level[-1]: # stop at empty level
        level.append([]) # start next level
        for u in level[-2]:
            for v in Adj[u]: # edge (u,v)
                if parent[v] is None: # v new
                    parent[v] = u
                    d[v] = d[u] + 1 #= len(level)-1
                    level[-1].append(v)
```

Relaxation

Let a solution (temporarily) violate a constraint, and try to fix these violations

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

```
def topological_sort(Adj):
    parent = [None] * len(Adj) # DFS tree
    order = []
    def visit(u):
        for v in Adj[u]:  # edge (u,v)
            if parent[v] is None:
                parent[v] = u
                visit(v)
        order.append(u)
    for s in range(len(Adj)): # try all sources
        if parent[s] is None:
            parent[s] = s
            visit(s)
    order.reverse()
    return order
```

```
def dag_sssp(Adj, w, s):
    parent = [None] * len(Adj) # shortest-path tree
    parent[s] = s              # s is root
    d = [math.inf] * len(Adj)  # d[v] = δ(s,v)
    d[s] = 0                   # δ(s,s) = 0

    def relax(u, v):
        if d[v] > d[u] + w(u,v):
            d[v] = d[u] + w(u,v)
            parent[v] = u

    for u in topological_sort(Adj):
        for v in Adj[u]:
            relax(u, v)
```

```
def dag_order(Adj):
    for u in topological_sort(Adj):
        for v in Adj[u]:
            yield (u, v)

def dag_sssp(Adj, w, s):
    order = dag_order(Adj)
    return relaxation_sssp(Adj, w, s,
        lambda: next(order, (None, None)))

def relaxation_sssp(Adj, w, s, pick_edge): ...
    while True:
        u, v = pick_edge(Adj, w, s, d)
        if u is None: break
        relax(u, v)
```

```
def relaxation_sssp(Adj, w, s, pick_edge):
    parent = [None] * len(Adj) # shortest-path tree
    parent[s] = s              # s is root
    d = [math.inf] * len(Adj)  # d[v] = δ(s,v)
    d[s] = 0                   # δ(s,s) ≤ 0

    def relax(u, v):
        if d[v] > d[u] + w(u,v):
            d[v] = d[u] + w(u,v)
            parent[v] = u

    while True:
        u, v = pick_edge(Adj, w, s, d)
        if u is None: break
        relax(u, v)
```