Introduction to Algorithms

CS 430

Lecture 23-24

ILLINOIS INSTITUTE OF TECHNOLOGY

By Lan Yao

Outlines

- Shortest path
  - Preliminaries
  - Bellman-Ford algorithm
  - Dijkstra’s algorithm
  - All pairs shortest paths

Shortest Paths*

How to find the shortest route between two points on a map.

Input:

- Directed graph $G = (V, E)$
- Weight function $w : E \to \mathbf{R}$

Weight of path $p = \langle v_0, v_1, \ldots, v_k \rangle$

$$= \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

$=$ sum of edge weights on path p .

Shortest-path weight u to v:

$$\delta(u, v) = \begin{cases} \min\left\{w(p) : u \overset{p}{\leadsto} v\right\} & \text{if there exists a path } u \leadsto v \text{ ,} \\ \infty & \text{otherwise .} \end{cases}$$

Shortest path u to v is any path p such that $w(p) = \delta(u, v)$.

Variants*

- ***Single-source:*** Find shortest paths from a given ***source*** vertex $s \in V$ to every vertex $v \in V$.
- ***Single-destination:*** Find shortest paths to a given destination vertex.
- ***Single-pair:*** Find shortest path from u to v. No way known that's better in worst case than solving single-source.
- ***All-pairs:*** Find shortest path from u to v for all $u, v \in V$.

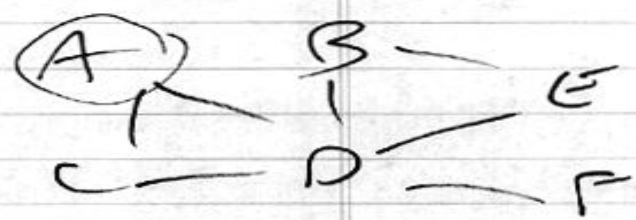
Negative-weight edges*

OK, as long as no negative-weight cycles are reachable from the source.

- If we have a negative-weight cycle, just keep going around it, and get $w(s, v) = -\infty$ for all v on the cycle.
- But OK if the negative-weight cycle is not reachable from the source.
- Some algorithms work only if there are no negative-weight edges in the graph.

BRUTE FORCE

TRY ALL POSSIBILITIES
FOR PATHS, PICK SHORTEST.



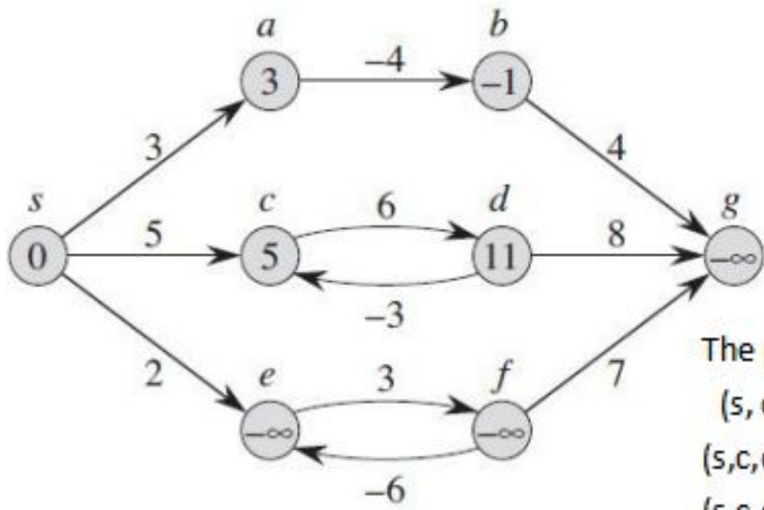
0 A

1 AC AD

2 ACD ADB ADC? ADE? ADF

3 ~~ACDA~~ ACDB ACDE ACDF

~~ADCA~~ A



The path from s to c:

(s, c)

(s,c,d,c)

(s,c,d,c,d,c,d,c,….) and each of the loop of cdc contributes 3 to the weight.

Then (s,c) is the shortest path from s to c.

The path from s to e:

(s, e)

(s,e,f,e)

(s,e,f,e,f,e,f,e,….) and each of the loop of efe contributes -3 to the weight.

Then (s,e,f,e, …) is the shortest path from s to e.

The path from s to g:

(s,a,b,g)

(s, c,d,g)

(s,c,d,c,d,…,g)

(s,e,f,g)

(s,e,f,e,f,e,f,e,….).

Can a shortest path contain a cycle?--NO!

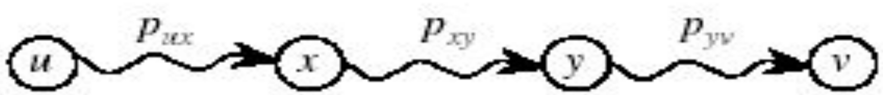
- When it contains a negative cycle: suppose there is a shortest path (s,d) containing a negative cycle and w(p)=105. If w(cycle)=-10, then w(p')=95<the weight of the shortest path 105. Then p is not the shortest path.
- When it contains a positive cycle: suppose there is a shortest path p= $(v_0, v_1, \ldots, v_k)$ containing a positive cycle and the cycle c= $(v_i, v_{i+1}, \ldots, v_j)$ and i=j. w(c)>0. p'=$(v_0, v_1, \ldots, v_i, v_{j+1}, v_{j+2}, \ldots, v_k)$ is the remaining path after c has be deleted from p. w(p')=w(p)-w(c)<w(p). It shows that p' is the shorter path than p and p is not the shortest path.
- When it contains a Zero weight cycle: when w(c)=0, w(p')=w(p)-w(c)=w(p). It shows that if we remove all the zero weight cycles, the weight remains the same. The shortest path does not contains a zero weight cycle.

We can assume that when we are seeking for a shortest paths,they do NOT have cycles and they are simple paths.

Optimal substructure *

Lemma
Any subpath of a shortest path is a shortest path.

Proof Cut-and-paste.



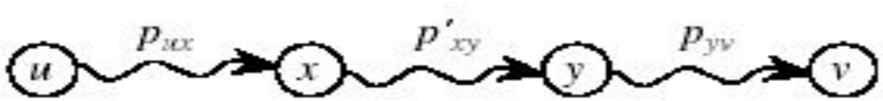
Suppose this path p is a shortest path from u to v.

Then $\delta(u, v) = w(p) = w(p_{ux}) + w(p_{xy}) + w(p_{yv})$.

Now suppose there exists a shorter path $x \overset{p'_{xy}}{\leadsto} y$.

Then $w(p'_{xy}) < w(p_{xy})$.

Construct p':



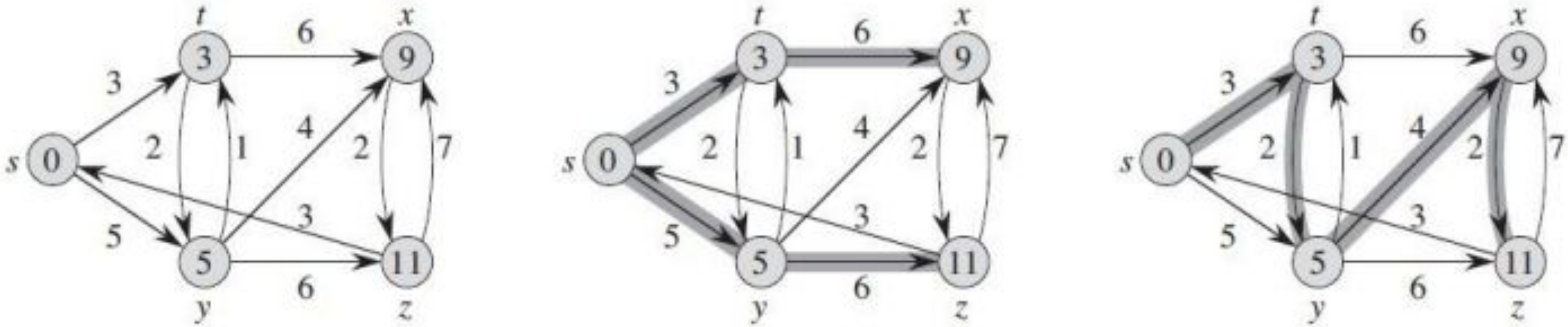
Output of single-source shortest-path algorithm

For each vertex $v \in V$:

$d[v] = \delta(s, v)$, Initially, $d[v]=\infty$, Reduces as algorithms progress. But always maintain $d[v] \geq \delta(s, v)$. Call $d[v]$ a ***shortest-path estimate***.

$\pi[v]$ = predecessor of v on a shortest path from s, If no predecessor, $\pi[v]$ = NIL, π induces a tree—***shortest-path tree***

The algorithms differ in the order and how many times they relax each edge.



(a) (b) (c)

A shortest path tree rooted at s is a directed subgraph $G'=(V',E')$ where $V' \subseteq V$ and $E' \subseteq E$ such that:

- V' is the set of vertices reachable from s in G;
- G' forms the rooted tree with root s;
- For all $v \in V'$, the unique simple path from s to v in G' is the shortest path from s to v in G.

Initialization

All the shortest-paths algorithms start with INIT-SINGLE-SOURCE.

INIT-SINGLE-SOURCE*(V, s)*

for each $v \in V$

    $d[v] \leftarrow \infty$

    $\pi[v] \leftarrow$ NIL

$d[s] \leftarrow 0$

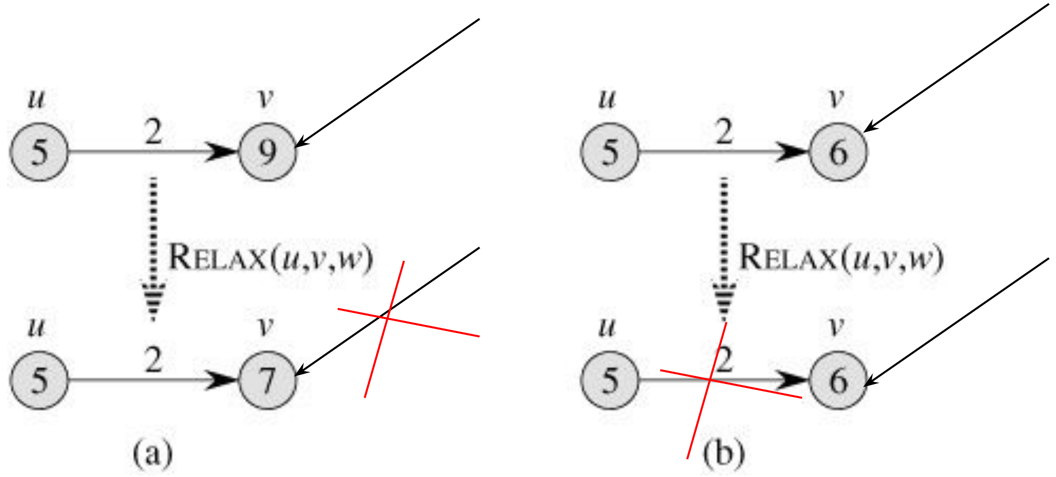
Relaxing an edge *(u, v)* - Can we improve the shortest-path estimate (best seen so far) for *v* by going through *u* and taking *(u, v)*?

RELAX*(u, v,w)*

if $d[u] + w(u, v) <= d[v]$

then $d[v] \leftarrow d[u] + w(u, v)$

$\pi[v] \leftarrow u$



The Bellman-Ford algorithm

- Allows negative-weight edges.
- Computes $d[v]$ and $\pi[v]$ for all $v \in V$.
- Returns TRUE if no negative-weight cycles reachable from s, FALSE otherwise.

BELLMAN-FORD(V, E, w, s)

INIT-SINGLE-SOURCE(V, s)

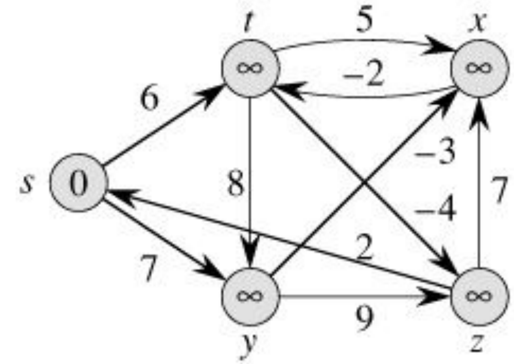
for $i \leftarrow 1$ to $|V|$-1
    for each edge $(u, v) \in E$ **// all edges, in any order**
        RELAX(u,v,w)

for each edge $(u, v) \in E$
    if $d[v] > d[u] + w(u, v)$
        then return FALSE

return TRUE

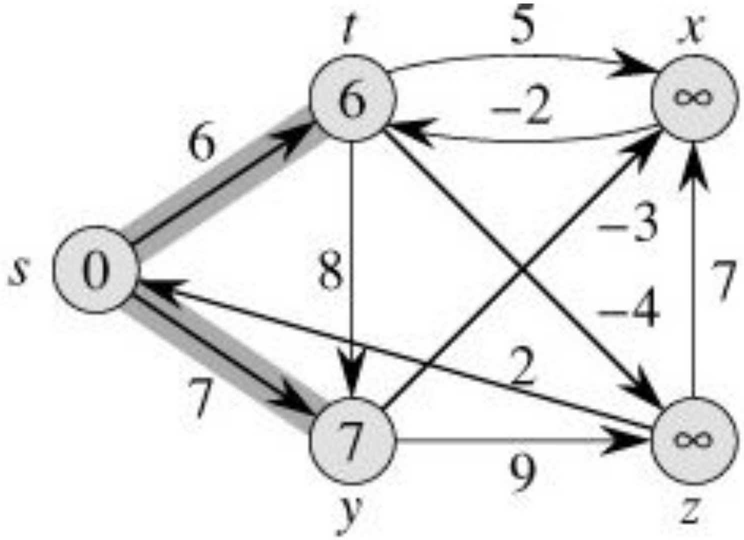
The first **for** loop relaxes all edges $|V|$-1 times.
$\boldsymbol{O}(VE+E)=O(VE)$

$= O(V^3)$

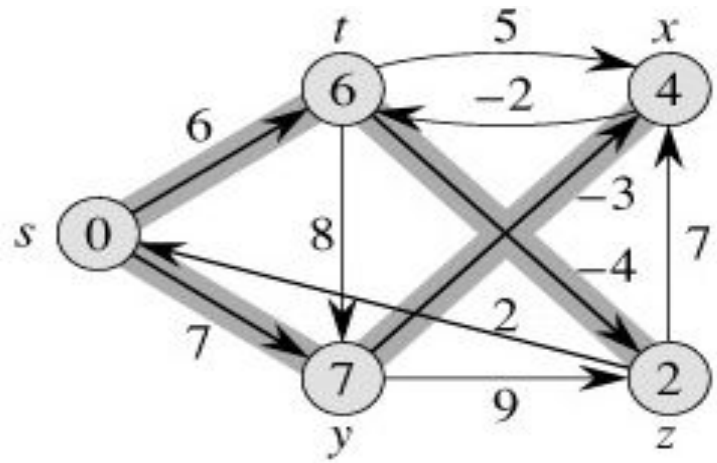


(a)

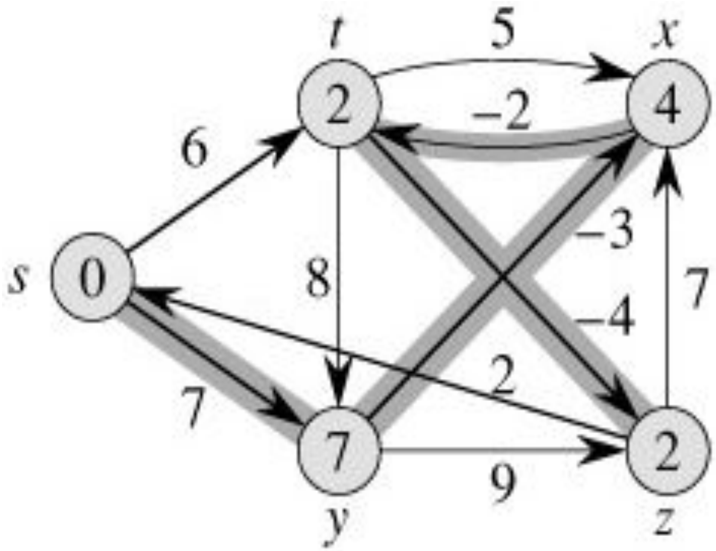
The execution of the Bellman-Ford algorithm. The source is vertex *s*. The *d* values are shown within the vertices, and shaded edges indicate predecessor values: if edge (*u, v*) is shaded, then π[*v*] = *u*. In this particular example, each pass relaxes the edges in the order (*t, x*), (*t, y*), (*t, z*), (*y, x*), (*y, z*),(*x, t*), (*z, x*), (*z, s*), (*s, t*), (*s, y*). (a) The situation just before the first pass over the edges. (b)-(e) The situation after each successive pass over the edges.



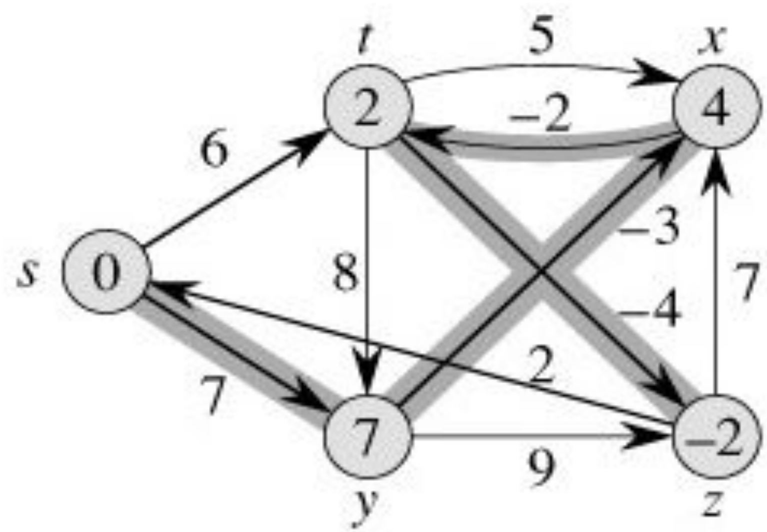
(b)



(c)



(d)



(e)

Review -1

- Term Explanation
  - DAG
  - Topological Sort
  - Shortest Path
- Does a shortest path have a cycle? Why?
- True or false: any subpath of a shortest path is a shortest path.

Review -2

- When looking for the shortest path, why RELAX?

- Options:What is Bellman-Ford designed for and what does it return?
  - A. for single source single destination shortest path and returns false if there is no negative cycle
  - B. for single source multiple destination shortest paths and returns false if there is no negative cycle
  - C. for single source single destination shortest path and returns True if there is no negative cycle
  - D. for single source multiple destination shortest paths and returns True if there is no negative cycle

Dijkstra’s algorithm

No negative-weight *edges*.

Essentially a weighted version of breadth-first search.

- Instead of a FIFO queue, uses a priority queue.
- Keys are shortest-path weight estimates ($d[v]$).

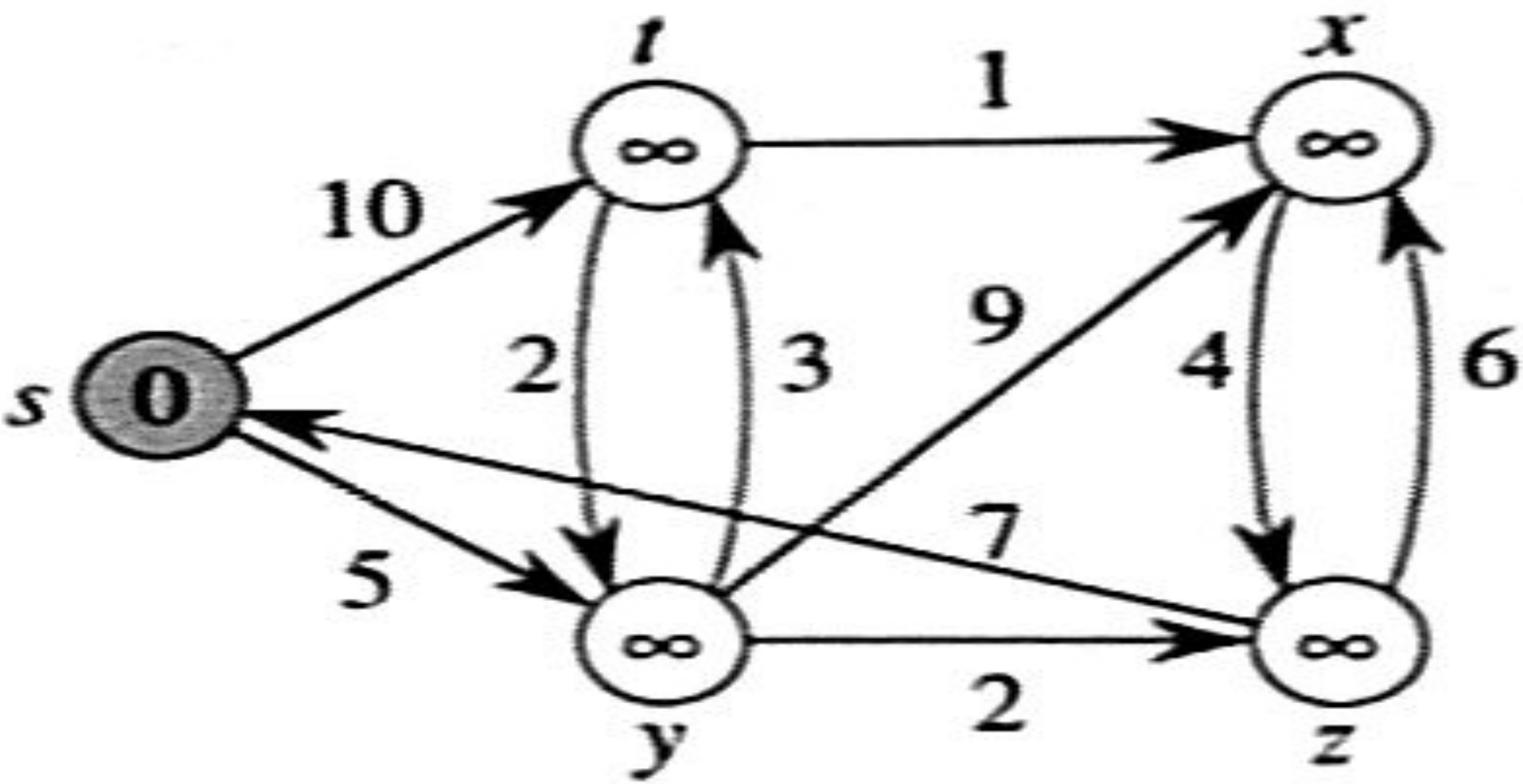
Have two sets of vertices:

- S = vertices whose final shortest-path weights are determined,
- Q = priority queue = V-S.

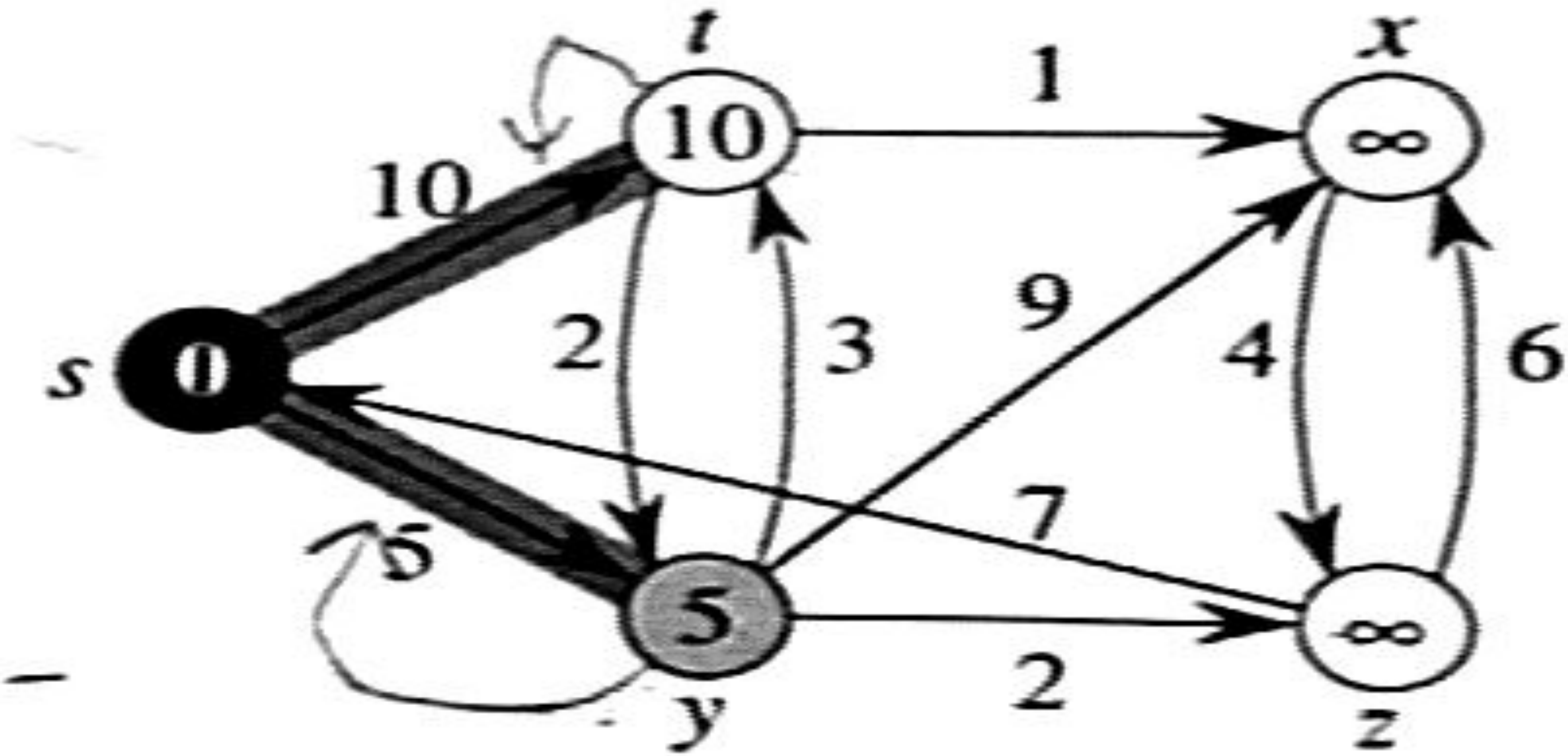
DIJKSTRA(V, E, w, s)

```
INIT-SINGLE-SOURCE(V, s)
S ← empty set
Q ← V   // i.e., insert all vertices into Q by "d" values
while Q not empty
      u ← EXTRACT-MIN(Q)
      S ← S U {u}
      for each vertex v ∈ Adj[u]
            RELAX(u,v,w)
```

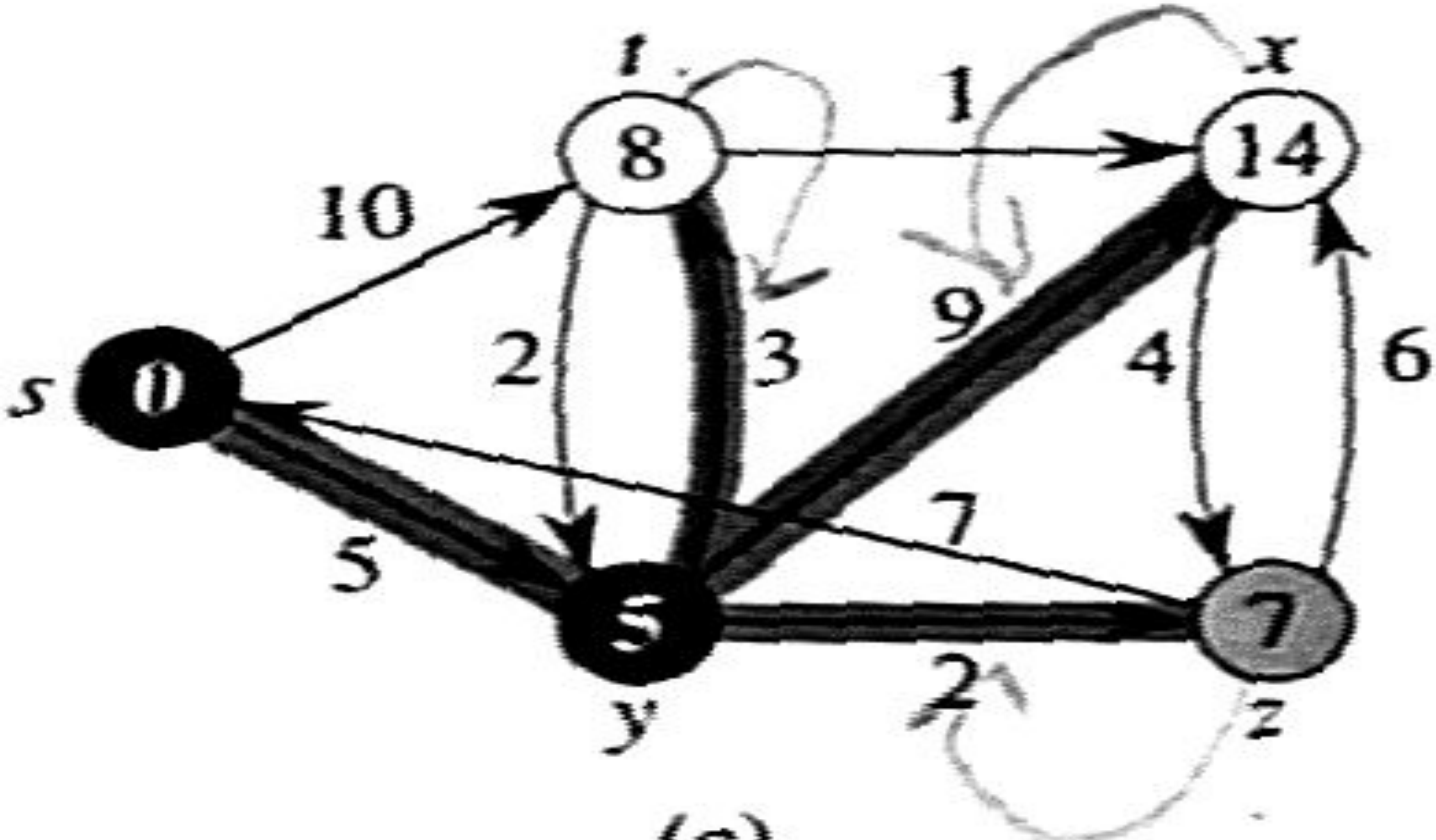
Dijkstra's algorithm can be viewed as greedy, since it always chooses the "lightest" ("closest"?) vertex in $V - S$ to add to S.



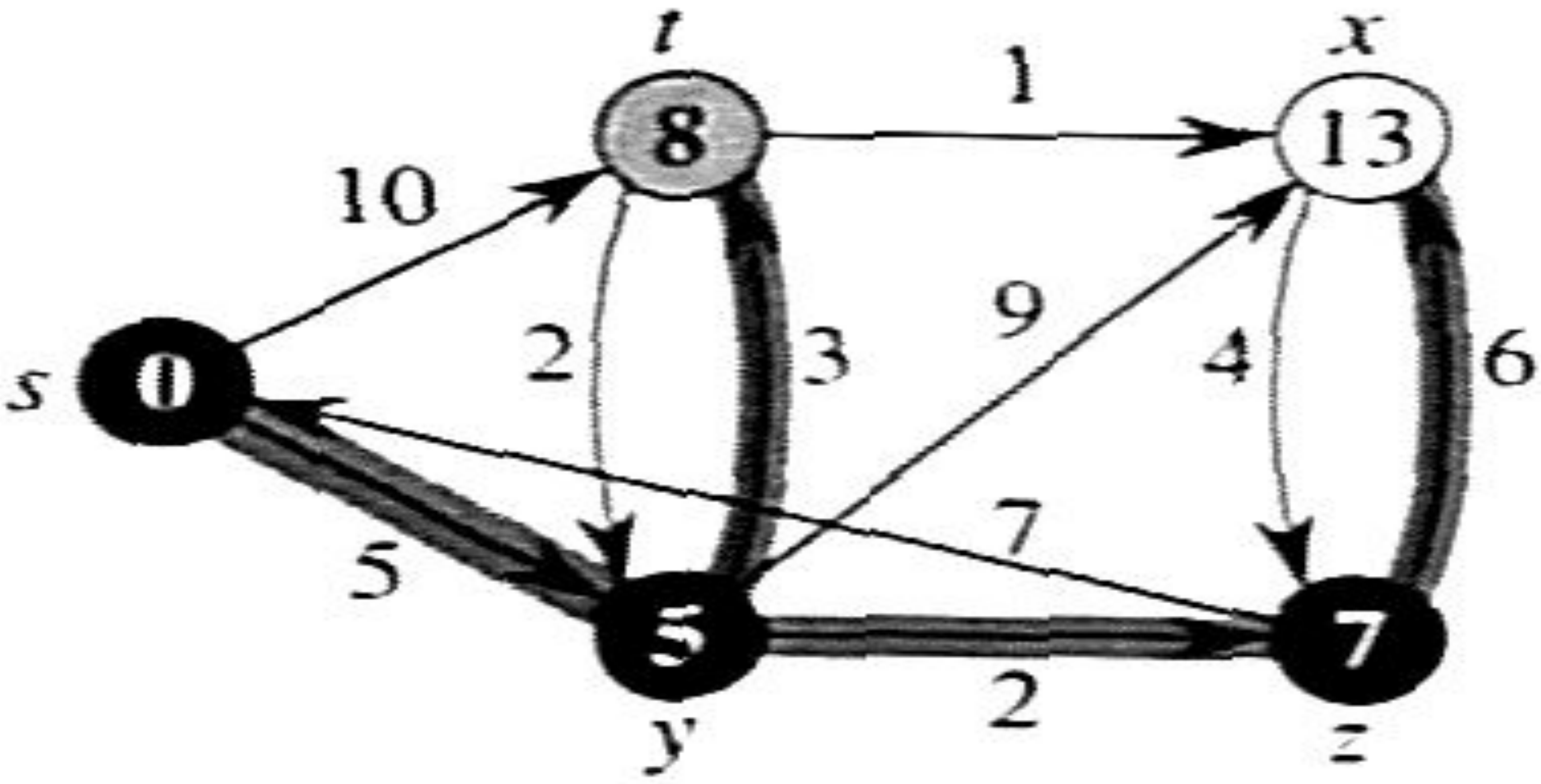
(a)



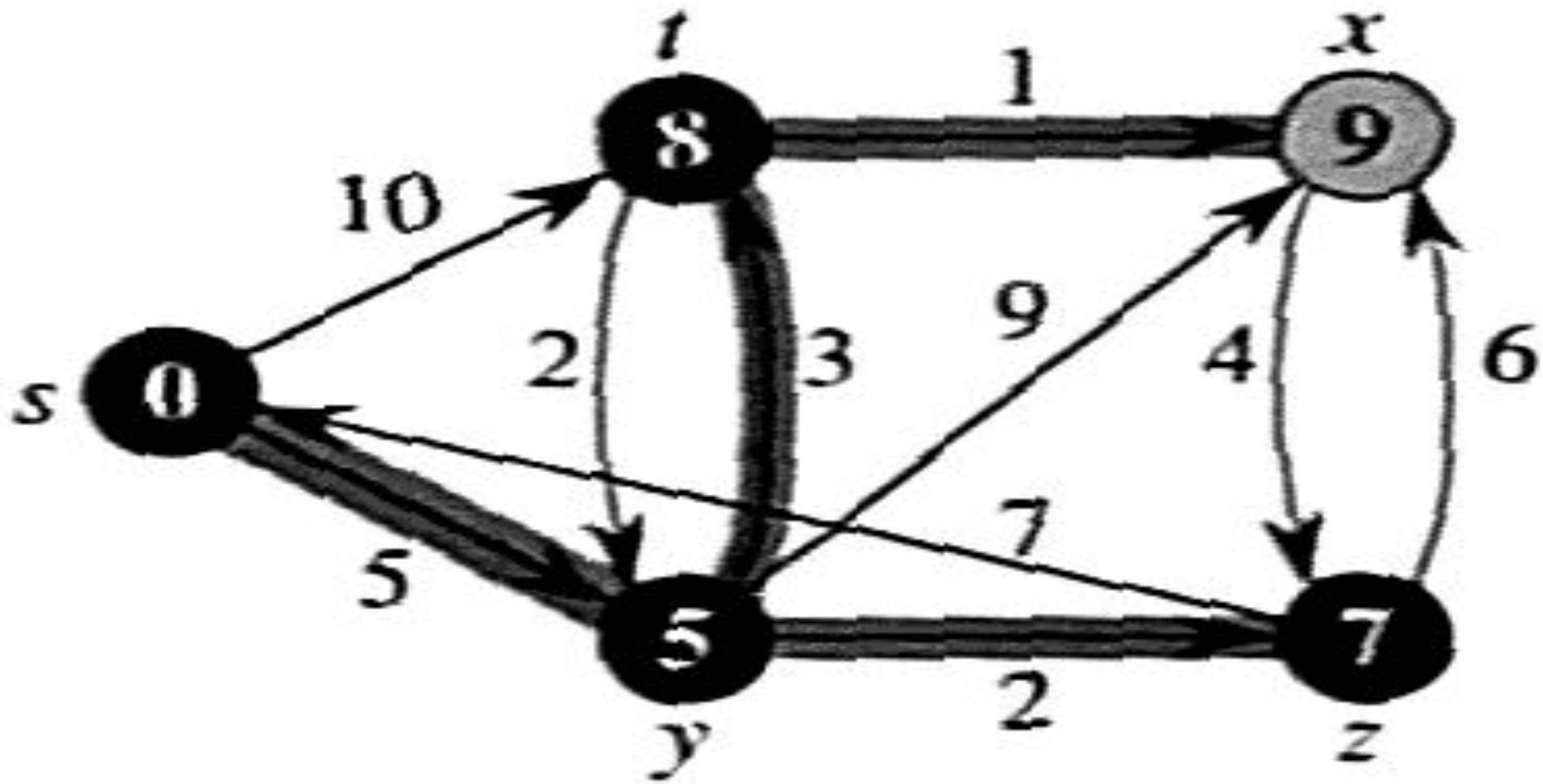
(b)



(c)



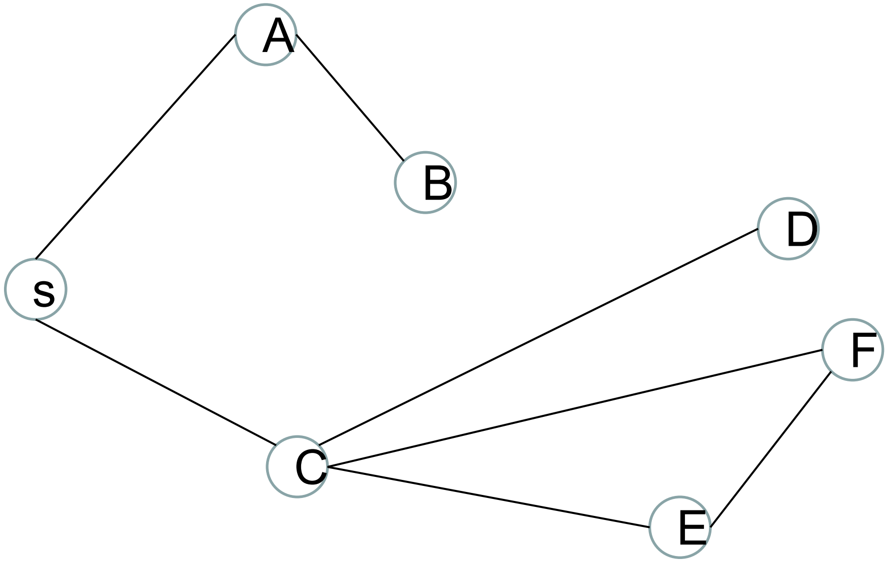
(d)



(e)

Prove Greedy Choice for Dijkstra

- Greedy Choice – pick the vertex with the smallest shortest path estimate (not including the vertices we are done with)
- Assume we have a solution: we know the shortest path from s to every other vertex. “S” is the set of edges in the solution. If S does not contain the greedy choice at the last step, we can remove the non-greedy last edge added to S and add the greedy choice to S and get just as good a solution.



G=(V, E);
s is the source;
Assume we have S as the shortest paths from s to any other vertices;
Wv is the weight of edge (s, v);

$\overline{uv}$ is the weight of (u,v).

Suppose that Greedy choice at some previous step was not chosen and it results in s-c-e-f as a shortest path in S. We replace this non-greedy choice by a greedy choice at vertex c, which is c-f, then we have:

$$W_c + \overline{CF} < W_c + \overline{CE} + \overline{EF}$$

It is a contradiction with the fact that all paths in S are shortest path.

Analysis: It depends on implementation of priority queue.

If binary heap, each operation takes $O(\lg V)$ time, overall $O(E \lg V)$.

All-Pairs Shortest Paths

- Given a directed graph $G = (V, E)$, weight function $w : E \rightarrow \mathbf{R}$, $|V| = n$.
- Goal: create an $n \times n$ matrix of shortest-path distances $\delta(u, v)$.
- Could run BELLMAN-FORD once from each vertex:
  - $O(V^2E)$ which is $O(V^4)$ if the graph is ***dense*** ($E =\sim V^2$).
- If no negative-weight edges, could run Dijkstra's algorithm once from each vertex:
  - $O(V E \lg V)$ with binary heap—$O(V^3 \lg V)$ if dense
- We'll see how to do in $O(V^3)$ in all cases, with no fancy data structure.

Shortest paths and matrix multiplication

- Assume that G is given as adjacency matrix of weights: $W = (w_{ij})$, with vertices numbered 1 to n.

$$w_{ij} = \begin{cases} 0 & \text{if } i = j , \\ \text{weight of } (i, j) & \text{if } i \neq j, (i, j) \in E , \\ \infty & \text{if } i \neq j, (i, j) \notin E . \end{cases}$$

- Output matrix $D = (d_{ij})$, where $d_{ij} = \delta(i, j)$.
  - All simple shortest paths contain <=n-1 edges
- Will use dynamic programming at first.

Shortest paths and matrix multiplication

- ***Optimal substructure:*** Recall: subpaths of shortest paths are shortest paths.
- ***Recursive solution:*** Let $l^{(m)}_{ij}$ = weight of shortest path from *i to j* that contains ≤ *m* edges.

$$l_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j , \\ \infty & \text{if } i \neq j . \end{cases}$$

m=0, there is a shortest path from *i to j* with ≤ *m* edges if and only if *i =j*

$$
\begin{aligned}
m &\geq 1 \\
\Rightarrow l_{ij}^{(m)} &= \min\left(l_{ij}^{(m-1)}, \min_{1 \leq k \leq n}\{l_{ik}^{(m-1)} + w_{kj}\}\right) \\
&\qquad (k \text{ is all possible predecessors of } j) \\
&= \min_{1 \leq k \leq n}\{l_{ik}^{(m-1)} + w_{kj}\} \\
&\qquad \text{since } w_{jj} = 0 \text{ for all } j.
\end{aligned}
$$

- Observe when *m* = 1, must have $l^{(1)}_{ij} = w_{ij}$
- Conceptually, when the path is restricted to at most 1 edge, the weight of the shortest path from *i to j* must be w_{ij}

$$
\begin{aligned}
l_{ij}^{(1)} &= \min_{1 \leq k \leq n}\{l_{ik}^{(0)} + w_{kj}\} \\
&= l_{ii}^{(0)} + w_{ij} \qquad (l_{ii}^{(0)} \text{ is the only non-}\infty \text{ among } l_{ik}^{(0)}) \\
&= w_{ij}\,.
\end{aligned}
$$

- $\delta(i, j) = l_{ij}^{(n-1)}$

Compute a solution bottom-up:

Compute $L^{(1)}, L^{(2)}, \ldots, L^{(n-1)}$

- Start with $L^{(1)} = W$, since $l^{(1)}_{ij} = w_{ij}$
- Go from $L^{(m-1)}$ to $L^{(m)}$:

EXTEND*(L,W, n)*

create *L'* an *n* ×*n* matrix

for *i* $\leftarrow$ 1 to *n*

for *j* $\leftarrow$ 1 to *n*

$l'_{ij} \leftarrow \infty$

for *k* $\leftarrow$ 1 to *n*

$l'_{ij} \leftarrow \min(l'_{ij}, l_{ik} + w_{kj})$

return *L'*

Runtime Complexity: $O(n^3)$

SLOW-APSP*(W, n)*

$L^{(1)} \leftarrow W$

for $m \leftarrow$ 2 to n -1

  $L^{(m)} \leftarrow$EXTEND$(L^{(m-1)}, W, n)$

return $L^{(n-1)}$

Time:

EXTEND: (n^3).

SLOW-APSP: (n^4).

Observation: EXTEND

EXTEND*(L, W, n)*

create *L'* an *n* × *n* matrix

for *i* ← 1 to *n*

for *j* ← 1 to *n*

$l'_{ij} \leftarrow \infty$

for *k* ← 1 to *n*

$l'_{ij} \leftarrow \min(l'_{ij},\ l_{ik} + w_{kj})$

return *L'*

$$L^{(1)} = L^{(0)} \bullet W = W$$

$$L^{(2)} = L^{(1)} \bullet W = W^{2}$$

$$L^{(n-1)} = L^{(n-2)} \bullet W = W^{n-1}$$

Why do we care?

Because our goal is to compute $L^{(n-1)}$ as fast as we can. Don't need to compute

all the intermediate $L^{(1)}, L^{(2)}, L^{(3)}, \ldots, L^{(n-2)}$.

Suppose we had a matrix A and we wanted to compute A^{n-1} (like calling EXTEND n-1 times).

Could compute $A, A^2, A^4, A^8, \ldots$

If we knew $A^m = A^{n-1}$ for all $m \geq n-1$, could just finish with A^r, where r is the smallest power of 2 that's ≥ n-1.

$$r = 2^{\lceil \lg(n-1) \rceil}$$

FASTER-APSP*(W, n)*

$L^{(1)} \leftarrow W$

$m \leftarrow 1$

while $m < n-1$

  $L^{(2m)} \leftarrow$ **EXTEND**$(L^{(m)}, L^{(m)}, n)$

  $m \leftarrow 2m$

return $L^{(m)}$

OK to overshoot, since products don't change after $L^{(n-1)}$

Time: $(n^3 \lg n)$.

Floyd-Warshall algorithm

- A different dynamic-programming approach.
- Optimal substructure
  - For path $p = \{v_i, v_1, v_2, \ldots, v_k, v_j\}$, an ***intermediate vertex*** is any vertex of *p* other than v_i or v_j
  - Let $d(k)_{ij}$ = shortest-path weight of any path from *i to j* with all intermediate vertices in $\{1, 2, \ldots, k\}$.

Consider a shortest path from *i to j* with all intermediate vertices in {1, 2, . . . , *k*}:

- If *k* is not an intermediate vertex, then all intermediate vertices of *p* are in {1, 2, . . . , *k*-1}
- If *k* is an intermediate vertex:

Recursive formulation

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 , \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 1 . \end{cases}$$

(Have $d^{(0)}_{ij} = w_{ij}$ because can't have intermediate vertices⇒≤1 edge.)

Want ($d^{(n)}_{ij}$), since all vertices numbered <= *n*

Compute bottom-up

Compute in increasing order of *k*:

FLOYD-WARSHALL*(W, n)*

$D(0) \leftarrow W$

for $k \leftarrow 1$ **to** n

  for $i \leftarrow 1$ **to** n

    for $j \leftarrow 1$ **to** n

      $d^{(k)}_{i\,j} \leftarrow \min\{ d^{(k-1)}_{ij}, d^{(k-1)}_{ik} + d^{(k-1)}_{kj} \}$

return $D^{(n)}$

Time: (n^3)