國立臺灣大學
National Taiwan University

UNIT 6
GRAPHS PART IV:
All-Pairs Shortest Paths

Iris Hui-Ru Jiang
Spring 2024

Department of Electrical Engineering
National Taiwan University

Outline

- Content:
  - All-Pairs Shortest Paths (APSP)
  - Floyd-Warshall Algorithm
  - Transitive-Closure
  - Johnson's Algorithm
- Reading:
  - Chapter 23

All-Pairs Shortest Paths (APSP)

- **The All-Pairs Shortest Path (APSP) Problem**
  - **Given:** A **directed** graph $G = (V, E)$ with edge weights
  - **Goal:** Find a minimum weight path (or cost) between **every pair** of vertices in V.
- **Method 1:** Extends the SSSP algorithms
  - No negative-weight edges: Run Dijkstra's algorithm $|V|$ times, once with each $v \in V$ as the source.
    - Adjacency **list** + Fibonacci heap: $O(V^2 \lg V + VE)$ time.
  - With negative-weight edges: Run the Bellman-Ford algorithm $|V|$ times, once with each $v \in V$ as the source.
    - Adjacency **list**: $O(V^2 E)$ time.
- **Method 2:** Applies the Floyd-Warshall algorithm (negative-weight edges allowed).
  - Adjacency **matrix**: $O(V^3)$ time.
- **Method 3:** Applies Johnson's algorithm for sparse graphs (negative-weight edges allowed).
  - Adjacency **list**: $O(V^2 \lg V + VE)$ time.

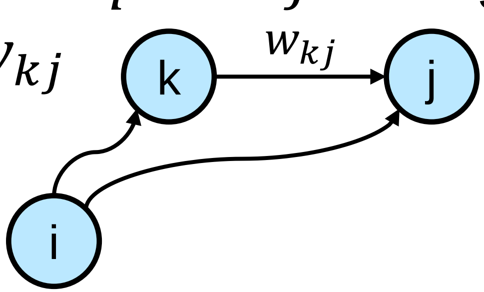
Warm Up

Dynamic Programming for Shortest Paths

- Steps of developing a DP algorithm
  1. Define the subproblem
  2. Characterize the structure of an optimal solution
  3. Recursively define the value of an optimal solution
  4. Compute the value of an optimal solution in a bottom-up fashion
  5. Construct an optimal solution from computed information
- Represent a weighted directed graph by adjacency matrix

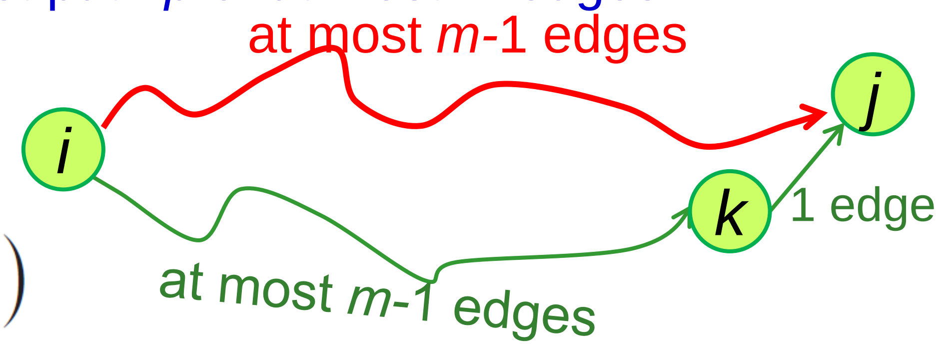
$$W = (w_{ij}) \quad w_{ij} = \begin{cases} 0 & \text{if } i = j\ , \\ \text{the weight of directed edge } (i, j) & \text{if } i \neq j \text{ and } (i, j) \in E\ , \\ \infty & \text{if } i \neq j \text{ and } (i, j) \notin E\ . \end{cases}$$

- By **triangle inequality**, if shortest path $p: i \rightsquigarrow j$ through edge (k, j), then $\delta(i, j) = \delta(i, k) + w_{kj}$



APSP: Induction on Passing Edges

- **Idea: induction on # of passing edges**
  - Let $l_{ij}^{(m)}$ be the weight of shortest path *p* of at most *m* edges



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

$$\begin{aligned} l_{ij}^{(m)} &= \min\left(l_{ij}^{(m-1)}, \min_{1 \le k \le n} \left\{l_{ik}^{(m-1)} + w_{kj}\right\}\right) \\ &= \min_{1 \le k \le n} \left\{l_{ik}^{(m-1)} + w_{kj}\right\}\ . \end{aligned}$$

$$l_{ij}^{(m-1)} = l_{ij}^{(m-1)} + w_{jj}$$

since $w_{jj} = 0$ for all j

  - Shortest path should be simple
    - At most *n*-1 edges

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

EXTEND-SHORTEST-PATHS(L, W)

```
n = L.rows
let L' = (l'_ij) be a new n × n matrix
for i = 1 to n
    for j = 1 to n
        l'_ij = ∞
        for k = 1 to n
            l'_ij = min(l'_ij, l_ik + w_kj)
return L'
```

Adjacency Matrix

Analogy

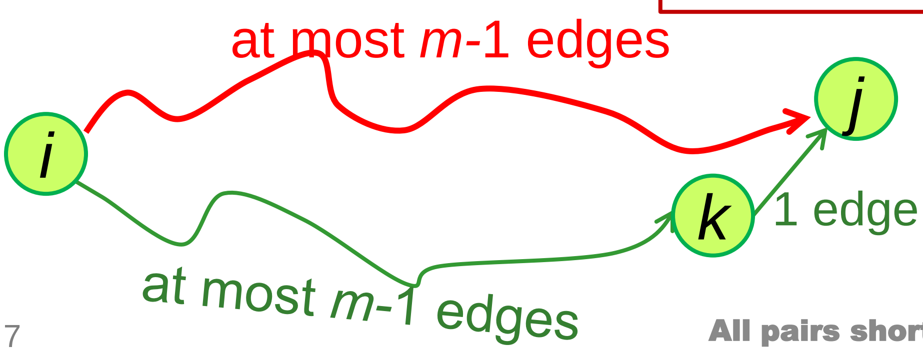
- **Extend-Shortest-Paths**
- **$L^{(m)} = L^{(m\text{-}1)} * W$**

$$l_{ij}^{(m)} = \min_{1 \le k \le n} \{ l_{ik}^{(m-1)} + w_{kj} \}$$

- **Matrix Multiplication**
- **C=A•B**

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

$$\boxed{\begin{aligned} l^{(m-1)} &\to a\,, \\ w &\to b\,, \\ l^{(m)} &\to c\,, \\ \min &\to +\,, \\ + &\to \cdot \end{aligned}}$$



APSP: Induction on Passing Edges

$$
\begin{aligned}
L^{(1)} &= L^{(0)} \cdot W &&= W\,, \\
L^{(2)} &= L^{(1)} \cdot W &&= W^2\,, \\
L^{(3)} &= L^{(2)} \cdot W &&= W^3\,, \\
&\ \ \vdots \\
L^{(n-1)} &= L^{(n-2)} \cdot W &&= W^{n-1}\,.
\end{aligned}
$$

SLOW-ALL-PAIRS-SHORTEST-PATHS(W)

```
n = W.rows
L^(1) = W
for m = 2 to n - 1
    let L^(m) be a new n × n matrix
    L^(m) = EXTEND-SHORTEST-PATHS(L^(m-1), W)
return L^(n-1)
```

$\Theta(n^4)$ time

APSP: Induction on Passing Edges

- Speed up
  - E.g., $7 = 2^2+2^1+2^0$

$$
\begin{aligned}
L^{(1)} &= W\,, \\
L^{(2)} &= W^2 &&= W \cdot W\,, \\
L^{(4)} &= W^4 &&= W^2 \cdot W^2 \\
L^{(8)} &= W^8 &&= W^4 \cdot W^4\,, \\
&\;\;\vdots \\
L^{(2^{\lceil \lg(n-1) \rceil})} &= W^{2^{\lceil \lg(n-1) \rceil}} &&= W^{2^{\lceil \lg(n-1) \rceil - 1}} \cdot W^{2^{\lceil \lg(n-1) \rceil - 1}}
\end{aligned}
$$

FASTER-ALL-PAIRS-SHORTEST-PATHS(W)

```
n = W.rows
L^(1) = W
m = 1
while m < n - 1
    let L^(2m) be a new n × n matrix
    L^(2m) = EXTEND-SHORTEST-PATHS(L^(m), L^(m))
    m = 2m
return L^(m)
```

$\Theta(n^3 \lg n)$ time

Floyd-Warshall's APSP Algorithm

Can we do better?

Overview of Floyd-Warshall's APSP Algorithm

- Applies dynamic programming.
  1. Define the subproblem
  2. Characterize the structure of an optimal solution
  3. Recursively define the value of an optimal solution
  4. Compute the value of an optimal solution in a bottom-up fashion
  5. Construct an optimal solution from computed information
- Uses adjacency matrix A for $G = (V, E)$:

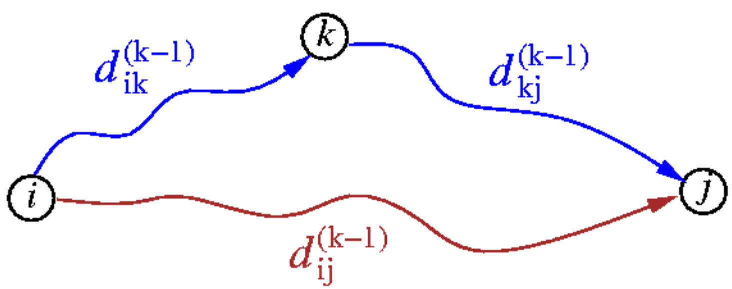
$$A[i,j] = a_{ij} = \begin{cases} 0, & \text{if } i = j \\ w_{ij}, & \text{if } (i,j) \in E \\ \infty, & \text{if } i \neq j \text{ and } (i,j) \notin E \end{cases}$$

- **Goal:** Compute $|V| \times |V|$ matrix D where $D[i, j] = d_{ij}$ is the weight of a shortest i-to-j path.
- Allows negative-weight edges.
- Runs in $O(V^3)$ time.

Shortest-Path Structure

- **Key idea: induction on intermediate vertices**
- **An intermediate vertex** of a simple path $\langle v_1, v_2, \ldots, v_l \rangle$ is any vertex in $\{v_2, v_3, \ldots, v_{l-1}\}$.
- Let $d_{ij}^{(k)}$ be the weight of a shortest path from vertex i to vertex j with all intermediate vertices in $\{1, 2, \ldots, k\}$.
  - The path does not use vertex k: $d_{ij}^{(k)} = d_{ij}^{(k-1)}$.
  - The path uses vertex k: $d_{ij}^{(k)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$
- **Def:** $D^k[i, j] = d_{ij}^{(k)}$ is the weight of a shortest i-to-j path with intermediate vertices in $\{1, 2, \ldots, k\}$.

$$d_{ij}^{(k)} = \begin{cases} w_{ij}, & \text{if } k = 0, \quad \text{no intermediate vertices} \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right), & \text{if } k \geq 1. \end{cases}$$



all intermediate vertices in {1, 2, ..., k−1}

The Floyd-Warshall Algorithm for APSP

Floyd-Warshall(W)
$n = W.rows$ // $W = A$
$D^{(0)} = W$;
for $k = 1$ **to** n
    let $D^{(k)} = (d_{ij}^{(k)})$ be a new n x n matrix
    for $i = 1$ **to** n
        for $j = 1$ **to** n
            $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
return $D^{(n)}$

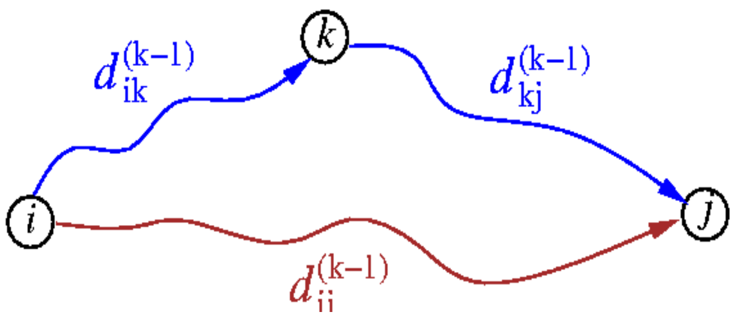
- $D^{(k)}[i, j] = d_{ij}^{(k)}$: weight of a shortest i-to-j path with intermediate vertices in $\{1, 2, \ldots, k\}$.
  - $D^{(0)} = A$: original adjacency matrix (paths are single edges).
  - $D^{(n)} = (d_{ij}^{(n)})$: the final answer ($d_{ij}^{(n)} = \delta(i, j)$).
- Time complexity: $O(V^3)$.
- **Question: How to detect negative-weight cycles?**

Constructing a Shortest Path

- **Predecessor matrix** $\Pi = (\pi_{ij})$: π_{ij} is
  - **NIL** if either $i=j$ or there is no path from i to j, or
  - **Some predecessor of *j*** on a shortest path from i
- Compute $\Pi^{(0)}, \Pi^{(1)}, \ldots, \Pi^{(n)}$, where $\pi_{ij}^{(k)}$ is defined to be the **predecessor of vertex** j on a shortest path from i with all intermediate vertices in $\{1, 2, \ldots, k\}$.

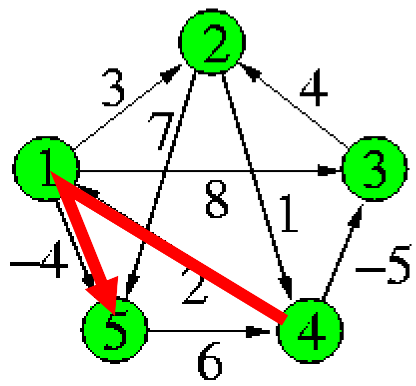
$$\pi_{ij}^{(0)} = \begin{cases} NIL, & \text{if } i = j \text{ or } w_{ij} = \infty, \\ i, & \text{if } i \neq j \text{ and } w_{ij} < \infty. \end{cases}$$

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)}, & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, \\ \pi_{kj}^{(k-1)}, & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}. \end{cases}$$




```
Print-All-Pairs-Shortest-Path(Π, i, j)
if i == j
      print i
elseif π_ij == NIL
      print "no path from i to j exists"
else Print-All-Pairs-Shortest-Path(Π, i, π_ij)
      print j
```

Example: The Floyd-Warshall Algorithm



$$\begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$D^{(0)} \qquad D^{(1)} \qquad D^{(2)}$$

$$\begin{pmatrix} NIL & 1 & 1 & NIL & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & NIL & NIL \\ 4 & NIL & 4 & NIL & NIL \\ NIL & NIL & NIL & 5 & NIL \end{pmatrix} \quad \begin{pmatrix} NIL & 1 & 1 & NIL & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & NIL & NIL \\ 4 & 1 & 4 & NIL & 1 \\ NIL & NIL & NIL & 5 & NIL \end{pmatrix} \quad \begin{pmatrix} NIL & 1 & 1 & 2 & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & 2 & 2 \\ 4 & 1 & 4 & NIL & 1 \\ NIL & NIL & NIL & 5 & NIL \end{pmatrix}$$

$$\Pi^{(0)} \qquad \Pi^{(1)} \qquad \Pi^{(2)}$$

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

$$\begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

$$D^{(3)} \qquad D^{(4)} \qquad D^{(5)}$$

$$\Pi_{ij}^{(k)} = \begin{cases} \Pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \le d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \Pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$

$$\begin{pmatrix} NIL & 1 & 1 & 2 & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & 2 & 2 \\ 4 & 3 & 4 & NIL & 1 \\ NIL & NIL & NIL & 5 & NIL \end{pmatrix} \quad \begin{pmatrix} NIL & 1 & 4 & 2 & 1 \\ 4 & NIL & 4 & 2 & 1 \\ 4 & 3 & NIL & 2 & 1 \\ 4 & 3 & 4 & NIL & 1 \\ 4 & 3 & 4 & 5 & NIL \end{pmatrix} \quad \begin{pmatrix} NIL & 3 & 4 & 5 & 1 \\ 4 & NIL & 4 & 2 & 1 \\ 4 & 3 & NIL & 2 & 1 \\ 4 & 3 & 4 & NIL & 1 \\ 4 & 3 & 4 & 5 & NIL \end{pmatrix}$$

$$\Pi^{(3)} \qquad \Pi^{(4)} \qquad \Pi^{(5)}$$

Transitive Closure of a Directed Graph

- **The Transitive Closure Problem:**
  - **Input:** a directed graph $G = (V, E)$
  - **Output:** a $|V| \times |V|$ matrix T s.t. $T[i, j] = 1$ iff there is a path from i to j in G.

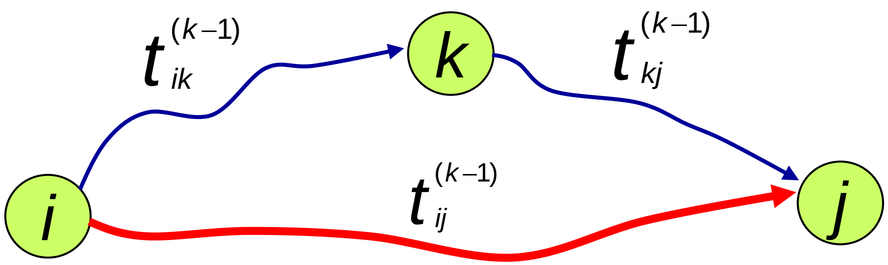
- **Method 1:** Run the Floyd-Warshall algorithm on **unit-weight** graphs. If there is a path from i to j ($T[i, j] = 1$), then $d_{ij} < n$; $d_{ij} = \infty$ otherwise.
  - Runs in $O(V^3)$ time and uses $O(V^2)$ space.
  - Needs to compute lengths of shortest paths.

Transitive Closure of a Directed Graph

- **Method 2:** Define $t_{ij}^{(k)} = 1$ if there exists a path in G from i to j with all intermediate vertices in $\{1, 2, \ldots, k\}$; 0, otherwise.

$$t_{ij}^{(0)} = \begin{cases} 0, & \text{if } i \neq j \text{ and } (i,j) \notin E \\ 1, & \text{if } i = j \text{ or } (i,j) \in E \end{cases}$$

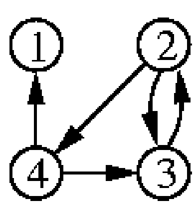
- For $k \geq 1$, $t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)})$.
  – Runs in $O(V^3)$ time and uses $O(V^2)$ space, but only single-bit Boolean values are involved $\Rightarrow$ faster!



Transitive-Closure Algorithm

Transitive-Closure(G)

$n = |G.V|$
let $T^{(0)} = (t_{ij}^{(0)})$ be a new n x n matrix
for $i = 1$ **to** n
    for $j = 1$ **to** n
        if $i == j$ or $(i, j) \in G.E$
            $t_{ij}^{(0)} = 1$
        else $t_{ij}^{(0)} = 0$
for $k = 1$ **to** n
    let $T^{(k)} = (t_{ij}^{(k)})$ be a new n x n matrix
    for $i = 1$ **to** n
        for $j = 1$ **to** n
            $t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}\right)$
return $T^{(n)}$



$$T^{(0)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

$$T^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad T^{(4)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Johnson's Algorithm

Reweighting is possible!

Johnson's Algorithm for Sparse Graphs

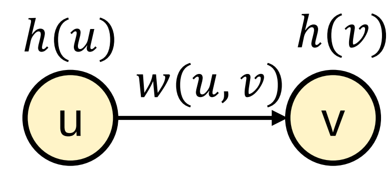
- **Idea:** All edge weights are nonnegative $\Rightarrow$ Dijkstra's algorithm is applicable (+ Fibonacci-heap priority queue: $O(V^2 \lg V + VE)$).
  - **Reweighting:** If there exists some edge weight $w < 0$, reweight w to $\hat{w} \geq 0$.
  - $\hat{w}$ must satisfy two properties:

1. $\forall u, v \in V$, shortest path from u to v w.r.t. $w \Rightarrow$ shortest path from u to v w.r.t. $\hat{w}$.

2. $\forall e \in E$, $\hat{w}(e) \geq 0$.

- **Preserving shortest paths by reweighting:** Given directed $G = (V, E)$ with weight function $w: E \rightarrow \mathbb{R}$, let h: V $\rightarrow \mathbb{R}$. For each edge $(u, v) \in E$, define

$$\hat{w}(u, v) = w(u, v) + h(u) - h(v).$$



Let $p = \langle v_0, v_1, \ldots, v_k \rangle$. Then

1. $w(p) = \delta(v_0, v_k)$ iff $\hat{w}(p) = \hat{\delta}(v_0, v_k)$.

2. G has a negative cycle w.r.t w iff $\hat{G}$ has a negative cycle w.r.t $\hat{w}$.

Donald B. Johnson. 1977. Efficient Algorithms for Shortest Paths in Sparse Networks. J. ACM 24, 1 (Jan 1977), 1-13.

Charles E. Leiserson and James B. Saxe. 1991. Retiming synchronous circuitry. Algorithmica 6, 1-6 (Jun. 1991), 5-35

Preserving Shortest Paths by Reweighting

$$\begin{aligned}\widehat{w}(p) &= \sum_{i=1}^{k} \widehat{w}(v_{i-1}, v_i) \\ &= \sum_{i=1}^{k} (w(v_{i-1}, v_i) + h(v_{i-1}) - h(v_i)) \\ &= \sum_{i=1}^{k} w(v_{i-1}, v_i) + h(v_0) - h(v_k) \\ &= w(p) + h(v_0) - h(v_k).\end{aligned}$$

1. Show that $w(p) = \delta(v_0, v_k) \Rightarrow \widehat{w}(p) = \hat{\delta}(v_0, v_k)$ by contradiction. Suppose $\exists\, p' = \langle v_0, \ldots, v_k \rangle$, $\widehat{w}(p') < \widehat{w}(p)$.

$$\begin{aligned}w(p') + h(v_0) - h(v_k) &= \widehat{w}(p') \\ &< \widehat{w}(p) \\ &= w(p) + h(v_0) - h(v_k)\end{aligned}$$

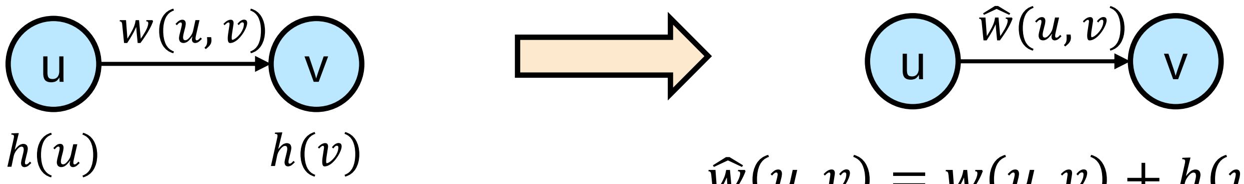
$\Rightarrow$ $w(p') < w(p)$, contradiction!

($\widehat{w}(p) = \hat{\delta}(v_0, v_k) \Rightarrow w(p) = \delta(v_0, v_k)$: similar!)

2. Show that G has a negative cycle w.r.t w iff $\widehat{G}$ has a negative cycle w.r.t $\widehat{w}$. Cycle $c = \langle v_0, v_1, \ldots, v_k = v_0 \rangle \Rightarrow \widehat{w}(c) = w(c) + h(v_0) - h(v_0) = w(c)$.

Outline

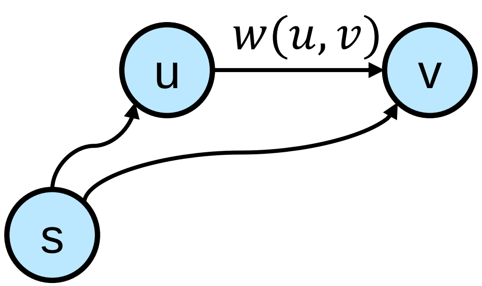
- Johnson's Algorithm idea
  - All edge weights are nonnegative $\Rightarrow$ Dijkstra's algorithm is applicable
  - **Reweighting:** reweight *w* to $\hat{w}$≥ 0 via h: $V \to \mathbb{R}$; $h(u)$ makes u earlier



$$\hat{w}(u,v) = w(u,v) + h(u) - h(v)$$

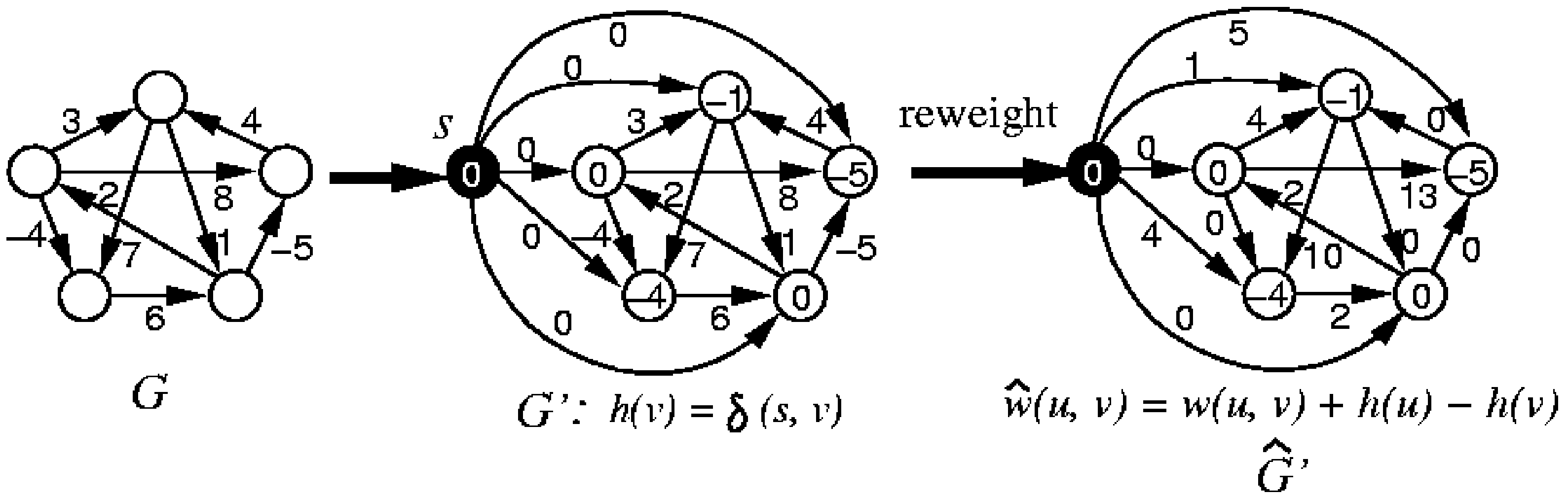
  - $\hat{w}$ must satisfy two properties:
    1. **1.** ∀*u*, *v* ∊ *V*, shortest path from *u* to *v* w.r.t. *w* ⇒ shortest path from *u* to *v* w.r.t. $\hat{w}$.
    2. **2.** ∀*e* ∊ *E*, $\hat{w}(e) \geq 0$.

- **Triangle inequality:**
  - $\forall (u,v) \in E, \delta(s,v) \leq \delta(s,u) + w(u,v)$
  - $w(u,v) + \delta(s,u) - \delta(s,v) \geq 0$
  - Choose $h(u) = \delta(s,u)$



Producing Nonnegative Weights by Reweighting

1. Construct a new graph $G' = (V', E')$, where
   $V' = V \cup \{s\}$
   $E' = E \cup \{(s, v): v \in V\}$
   $w(s, v) = 0, \forall v \in V$
2. G has no negative cycle $\Leftrightarrow G'$ has no negative cycle.
3. If G and G' have no negative cycles, define $h(v) = \delta(s, v), \forall v \in V'$.
4. Define the new weight $\hat{w}$ for $\hat{G}'$: $\hat{w}(u, v) = w(u, v) + h(u) - h(v) \geq 0$
   (since $\delta(s, v) \leq \delta(s, u) + w(u, v) \Rightarrow w(u, v) + \delta(s, u) - \delta(s, v) \geq 0$).



Johnson's Algorithm for APSP

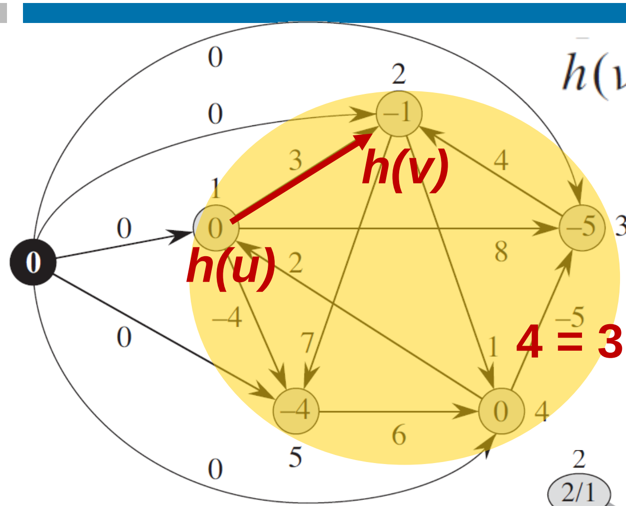
Johnson(G)

1. compute G', where $G'.V = G.V \cup \{s\}$,
   $G'.E = G.E \cup \{(s,v): v \in G.V\}$ and $w(s, v) = 0$ for all $v \in G.V$
2. **if** Bellman-Ford(G', w, s) = FALSE
3. print "the input graph contains a negative-weight cycle"
4. **else for** each vertex $v \in G'.V$
5. set $h(v)$ to the value of $\delta(s,v)$ computed by the Bellman-Ford algorithm
6. **for** each edge $(u,v) \in G'.E$
7. $\hat{w}(u,v) = w(u, v) + h(u) - h(v)$
8. let $D = (d_{uv})$ be a new n x n matrix
9. **for** each vertex $u \in G.V$
10. run Dijkstra(G, $\hat{w}$, u) to compute $\hat{\delta}(u,v)$ for all $v \in G.V$
11. **for** each vertex $v \in G.V$
12. $\boldsymbol{d_{uv} = \hat{\delta}(u, v) + h(v) - h(u)}$
13. **return** D

- Steps: Compute the new graph ⇒ Bellman-Ford ⇒ Reweighting ⇒ Dijkstra's (+ Fibonacci heap) ⇒ Recompute the weights.
- Time complexity: $O(V) + O(VE) + O(E) + O(V^2 \lg V + VE) + O(V^2) = O(V^2 \lg V + VE)$.

Example: Johnson's Algorithm for APSP

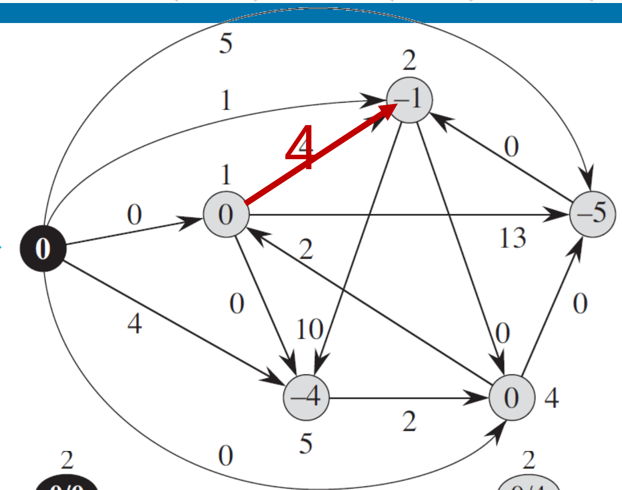
$$\hat{w}(u,v) = w(u,v) + h(u) - h(v)$$



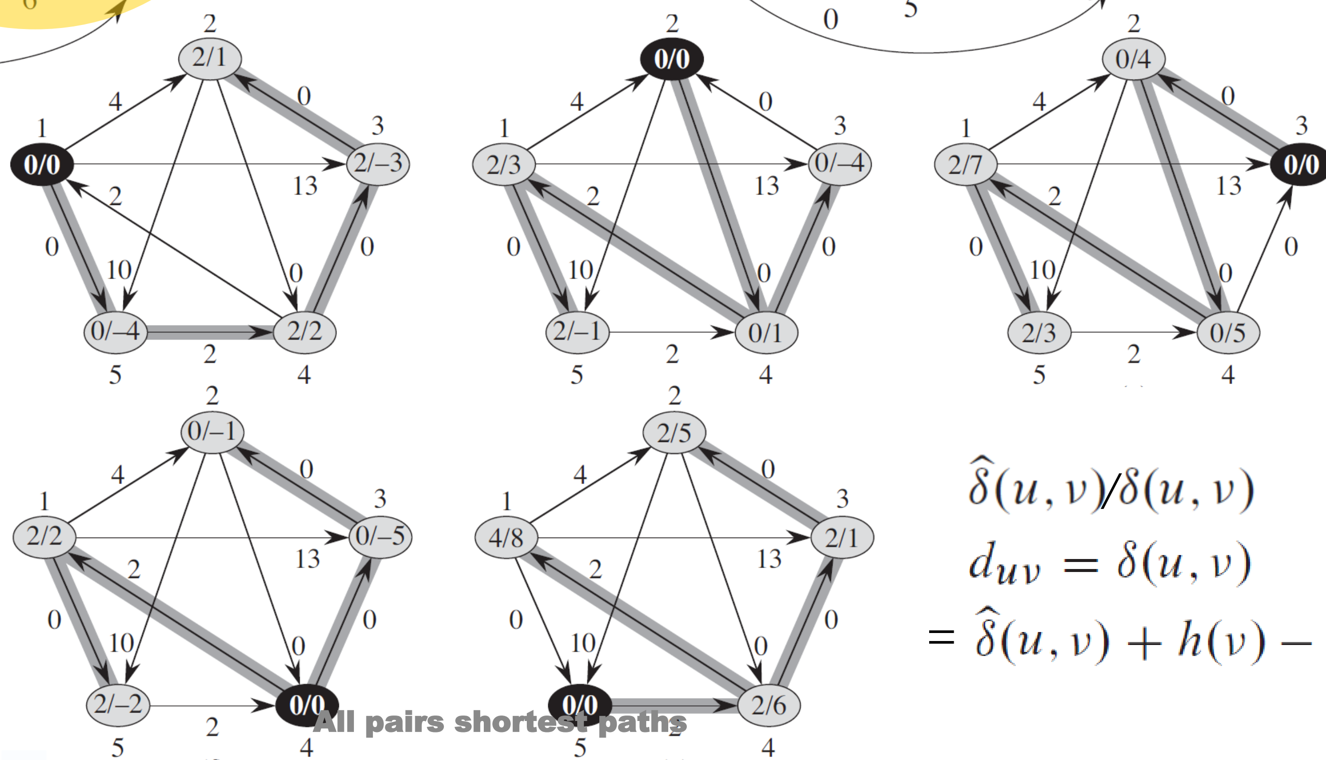
$$h(v) = \delta(s,v)$$

4 = 3 + 0 – (-1)

reweighting



Dijkstra's



$$\hat{\delta}(u,v)/\delta(u,v)$$

$$d_{uv} = \delta(u,v)$$
$$= \hat{\delta}(u,v) + h(v) - h(u).$$

How to Detect Negative Cycles?

- List two methods discussed in lecture.