Advanced Algorithms Analysis and Design

By

Nazir Ahmad Zafar

Lecture No 37

The Floyd-Warshall Algorithm
and
Johnson's Algorithm

Intermediate Vertices

- Vertices in G are given by:
  $V = \{1, 2, \ldots, n\}$
- Consider a path $p = \langle v_1, v_2, \ldots, v_l \rangle$
  - An **intermediate** vertex of p is any vertex in set $\{v_2, v_3, \ldots, v_{l-1}\}$

Example 1

If $p = \langle 1, 2, 4, 5 \rangle$ then

I.V. = $\{2, 4\}$

Example 2

If $p = \langle 2, 4, 5 \rangle$ then

I.V. = $\{4\}$

The Floyd Warshall Algorithm

1. Structure of a Shortest Path

- Let $V = \{1, 2,..., n\}$ be a set of vertices of G
- Consider subset $\{1, 2,..., k\}$ of vertices for some k
- *Let p be a* shortest paths from i to j with all intermediate vertices in the set $\{1, 2,..., k\}$.
- It exploits a relationship between path p and shortest paths from i to j with all intermediate vertices in the set $\{1, 2,..., k - 1\}$.
- The relationship depends on whether or not k is an intermediate vertex of path p.
- For both cases optimal structure is constructed

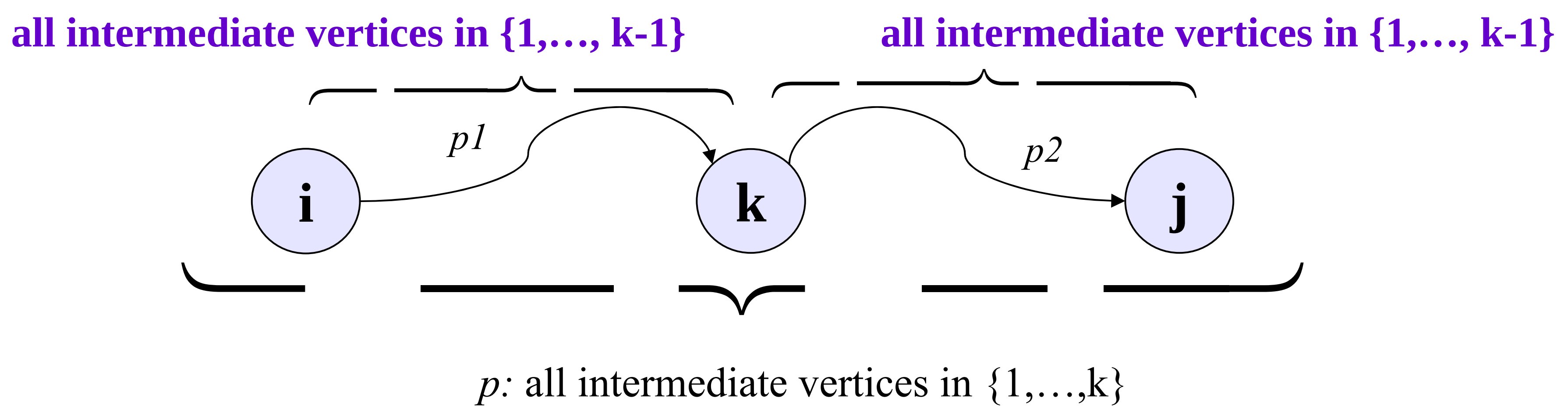
1. The Structure of Shortest Path

k not an I. vertex. of path p

- Shortest path i to j with I.V. from {1, 2, …, k} is shortest path i to j with I.V. from {1, 2, …, k - 1}

k an intermediate vertex of path p

- p1 is a shortest path from i to k
- p2 is a shortest path from k to j
- k is neither intermediate vertex of p1 nor of p2
- p1, p2 shortest paths i to k with I.V. from: {1, 2, …, k - 1}



2. A Recursive Solution

- Let $d_{ij}^{(k)}$ = be the weight of a shortest path from vertex *i* to vertex *j* for which all intermediate vertices are in the set {1, 2,. . ., *k*}.

- Now $D^{(n)} = (d_{i,j}^{(n)})$,
- Base case $d_{i,j}^{(0)}) = w_{i,j}$
- $D^{(0)} = (w_{i,j}) = W$
- The recursive definition is given below

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

3. The Floyd Warshall Algorithm

FLOYD-WARSHALL (W)

$n \leftarrow rows[W]$

$D^{(0)} \leftarrow W$

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

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

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

      do $d_{ij}^{(k)} \leftarrow \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$

return $D^{(n)}$

$\boxed{\textbf{Total Running Time} = \Theta(n^3)}$

Constructing a Shortest Path

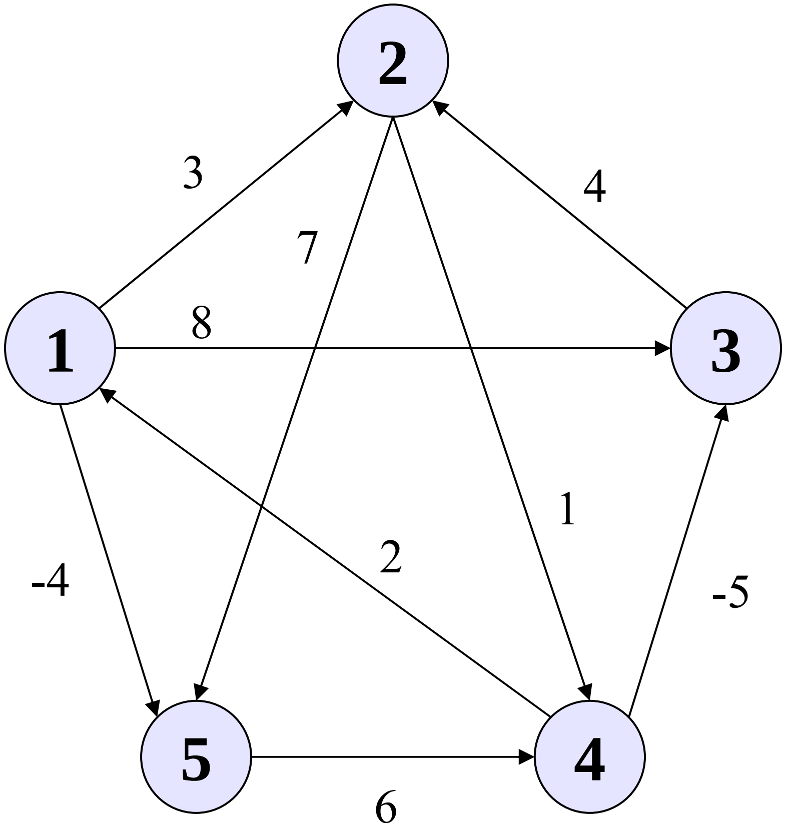
- One way is to compute matrix *D* of SP weights and then construct predecessor matrix Π from *D* matrix.
  - *It takes $O(n^3)$*
- A recursive formulation of: $\pi_{ij}^{(k)}$
  - $k = 0$, shortest path from *i* to *j* has no intermediate vertex

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

  - For $k \geq 1$

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

Example: Floyd Warshall Algorithm



$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

Adjacency Matrix of given Graph

Example: Floyd Warshall Algorithm

For $k = 0$

$$D^{(0)} = \begin{pmatrix} \mathbf{0} & \mathbf{3} & \mathbf{8} & \boldsymbol{\infty} & \mathbf{-4} \\ \boldsymbol{\infty} & 0 & \infty & 1 & 7 \\ \boldsymbol{\infty} & 4 & 0 & \infty & \infty \\ \mathbf{2} & \infty & -5 & 0 & \infty \\ \boldsymbol{\infty} & \infty & \infty & 6 & 0 \end{pmatrix} \qquad \Pi^{(0)} = \begin{pmatrix} NIL & 1 & 1 & NIL & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & NIL & NIL \\ 4 & NIL & 4 & NIL & NIL \\ NIL & NIL & NIL & 5 & NIL \end{pmatrix}$$

Example: Floyd Warshall Algorithm

For k = 1

$d_{3,4}^{(1)} = \min (d_{3,4}^{(0)}, d_{3,\mathbf{1}}^{(0)} + d_{\mathbf{1},4}^{(0)})$
$= \min (\infty, \infty + \infty) = \infty$

$$D^{(0)} = \begin{pmatrix} \mathbf{0} & \mathbf{3} & \mathbf{8} & \boldsymbol{\infty} & \mathbf{-4} \\ \boldsymbol{\infty} & 0 & \infty & 1 & 7 \\ \boldsymbol{\infty} & 4 & 0 & \infty & \infty \\ \mathbf{2} & \infty & -5 & 0 & \infty \\ \boldsymbol{\infty} & \infty & \infty & 6 & 0 \end{pmatrix} \Rightarrow D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \mathbf{5} & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$d_{4,2}^{(1)} = \min (d_{4,2}^{(0)}, d_{4,\mathbf{1}}^{(0)} + d_{\mathbf{1},2}^{(0)})$
$= \min (\infty, 2 + 3) = 5$

Example: Floyd Warshall Algorithm

For k = 1

$$D^{(1)} = \begin{pmatrix} 0 & \mathbf{3} & 8 & \infty & -4 \\ \mathbf{0} & \boldsymbol{\infty} & \mathbf{8} & \boldsymbol{\infty} & \mathbf{-4} \\ \infty & \boldsymbol{\infty} & 0 & \infty & \infty \\ 2 & \mathbf{2} & -5 & 0 & -2 \\ \infty & \boldsymbol{\infty} & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(1)} = \begin{pmatrix} NIL & 1 & 1 & NIL & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & NIL & NIL \\ 4 & 1 & 4 & NIL & 1 \\ NIL & NIL & NIL & 5 & NIL \end{pmatrix}$$

Example: Floyd Warshall Algorithm

For k = 2

$$d_{4,3}^{(2)} = \min\left(d_{4,3}^{(1)}, d_{4,2}^{(1)} + d_{\mathbf{2},3}^{(1)}\right)$$
$$= \min(-5, 5 + \infty) = -5$$

$$D^{(1)} = \begin{pmatrix} 0 & \mathbf{3} & 8 & \infty & -4 \\ \mathbf{0} & \boldsymbol{\infty} & \mathbf{8} & \boldsymbol{\infty} & \mathbf{-4} \\ \infty & \boldsymbol{\infty} & 0 & \infty & \infty \\ 2 & \mathbf{2} & -5 & 0 & -2 \\ \infty & \boldsymbol{\infty} & \infty & 6 & 0 \end{pmatrix} \Rightarrow D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & \mathbf{4} & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \mathbf{5} & \mathbf{11} \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$d_{1,4}^{(2)} = \min\left(d_{1,4}^{(1)}, d_{1,\mathbf{2}}^{(1)} + d_{\mathbf{2},4}^{(1)}\right)$$
$$= \min(\infty, 3 + 1) = 4$$

Example: Floyd Warshall Algorithm

For $k = 2$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & \mathbf{8} & 4 & -4 \\ \infty & 0 & \mathbf{\infty} & 1 & 7 \\ \mathbf{0} & \mathbf{4} & \mathbf{0} & \mathbf{5} & \mathbf{11} \\ 2 & 5 & \mathbf{-5} & 0 & -2 \\ \infty & \infty & \mathbf{\infty} & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} NIL & 1 & 1 & 2 & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & 2 & 2 \\ 4 & 1 & 4 & NIL & 1 \\ NIL & NIL & NIL & 5 & NIL \end{pmatrix}$$

Example: Floyd Warshall Algorithm

For k = 3

$$D^{(2)} = \begin{pmatrix} 0 & 3 & \mathbf{8} & 4 & -4 \\ \infty & 0 & \boldsymbol{\infty} & 1 & 7 \\ \mathbf{0} & \mathbf{4} & \mathbf{0} & \mathbf{5} & \mathbf{11} \\ 2 & \mathbf{5} & \mathbf{-5} & 0 & -2 \\ \infty & \infty & \boldsymbol{\infty} & 6 & 0 \end{pmatrix} \Rightarrow D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & \mathbf{-1} & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$d_{4,2}{}^{(3)} = \min (d_{4,2}{}^{(2)}, d_{4,\mathbf{3}}{}^{(2)} + d_{\mathbf{3},2}{}^{(2)})$
$= \min (5, -5 + 4) = -1$

Example: Floyd Warshall Algorithm

For *k* = 3

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & \mathbf{4} & -4 \\ \infty & 0 & \infty & \mathbf{1} & 7 \\ \infty & 4 & 0 & \mathbf{5} & 11 \\ \mathbf{2} & \mathbf{-1} & \mathbf{-5} & \mathbf{0} & \mathbf{-2} \\ \infty & \infty & \infty & \mathbf{6} & 0 \end{pmatrix} \quad \Pi^{(3)} = \begin{pmatrix} NIL & 1 & 1 & 2 & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & 2 & 2 \\ 4 & 3 & 4 & NIL & 1 \\ NIL & NIL & NIL & 5 & NIL \end{pmatrix}$$

Example: Floyd Warshall Algorithm

For k = 4

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & \mathbf{4} & -4 \\ \infty & 0 & \infty & \mathbf{1} & 7 \\ \infty & 4 & 0 & \mathbf{5} & 11 \\ \mathbf{2} & \mathbf{-1} & \mathbf{-5} & \mathbf{0} & \mathbf{-2} \\ \infty & \infty & \infty & \mathbf{6} & 0 \end{pmatrix} \Rightarrow D^{(4)} = \begin{pmatrix} 0 & 3 & \mathbf{-1} & 4 & -4 \\ \mathbf{3} & 0 & \mathbf{-4} & 1 & \mathbf{-1} \\ \mathbf{7} & 4 & 0 & 5 & \mathbf{3} \\ 2 & -1 & -5 & 0 & -2 \\ \mathbf{8} & \mathbf{5} & \mathbf{1} & 6 & 0 \end{pmatrix}$$

$d_{5,2}{}^{(4)} = \min\,(d_{5,2}{}^{(3)},\ d_{5,\mathbf{4}}{}^{(3)} + d_{\mathbf{4},2}{}^{(3)})$
$= \min\,(\infty,\ 6 + (-1)) = 5$

Example: Floyd Warshall Algorithm

For k = 4

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & \mathbf{-4} \\ 3 & 0 & -4 & 1 & \mathbf{-1} \\ 7 & 4 & 0 & 5 & \mathbf{3} \\ 2 & -1 & -5 & 0 & \mathbf{-2} \\ \mathbf{8} & \mathbf{5} & \mathbf{1} & \mathbf{6} & \mathbf{0} \end{pmatrix} \quad \Pi^{(4)} = \begin{pmatrix} NIL & 1 & 4 & 2 & 1 \\ 4 & NIL & 4 & 2 & 1 \\ 4 & 3 & NIL & 2 & 1 \\ 4 & 3 & 4 & NIL & 1 \\ 4 & 3 & 4 & 5 & NIL \end{pmatrix}$$

The Floyd Warshall Algorithm

For *k* = 5

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & \mathbf{-4} \\ 3 & 0 & -4 & 1 & \mathbf{-1} \\ 7 & 4 & 0 & 5 & \mathbf{3} \\ 2 & -1 & -5 & 0 & \mathbf{-2} \\ \mathbf{8} & \mathbf{5} & \mathbf{1} & \mathbf{6} & \mathbf{0} \end{pmatrix} \Rightarrow D^{(5)} = \begin{pmatrix} 0 & \mathbf{1} & \mathbf{-3} & \mathbf{2} & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

$d_{1,2}{}^{(5)} = \min\,(d_{1,2}{}^{(4)}, d_{1,\mathbf{5}}{}^{(4)} + d_{\mathbf{5},2}{}^{(4)})$
$= \min\,(3, (-4) + (5)) = 1$

The Floyd Warshall Algorithm

For *k* = 5

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(5)} = \begin{pmatrix} NIL & 3 & 4 & 5 & 1 \\ 4 & NIL & 4 & 2 & 1 \\ 4 & 3 & NIL & 2 & 1 \\ 4 & 3 & 4 & NIL & 1 \\ 4 & 3 & 4 & 5 & NIL \end{pmatrix}$$

Existence of Shortest Paths Between any Pair

Transitive Closure

- Given a directed graph $G = (V, E)$ with vertex set $V = \{1, 2, ..., n\}$, we may wish to find out whether there is a path in G from i to j for all vertex pairs $i, j \in V$.
- The **transitive closure** of G is defined as the graph $G^* = (V, E^*)$, where $E^* = \{(i, j)$: there is a path from vertex i to vertex j in $G\}$.
- One way is to assign a weight of 1 to each edge of E and run the Floyd-Warshall algorithm.
  - If there is a path from vertex i to j, then $d_{ij} < n$
  - Otherwise, we get $d_{ij} = \infty$.
  - The running time is $\Theta(n^3)$ time

Existence of Shortest Paths Between any Pair

Substitution

- Substitute logical operators, $\vee$ (for min) and $\wedge$ (for +) in the Floyd-Warshall algorithm
  - Running time: $\Theta(n^3)$ which saves time and space
  - A recursive definition is given by
  - K = 0

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

  - For k $\geq$ 1

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge d_{kj}^{(k-1)} \right)$$

Transitive Closure

TRANSITIVE-CLOSURE(G)

```
n ← |V[G]|
for i ← 1 to n
    do for j ← 1 to n
        do if i = j or (i, j) ∈ E[G]
            then t_ij^(0) ← 1
            else t_ij^(0) ← 0
for k ← 1 to n
    do for i ← 1 to n
        do for j ← 1 to n
            do t_ij^(k) ← t_ij^(k-1) ∨ (t_ik^(k-1) ∧ d_kj^(k-1))
return T^(n)
```

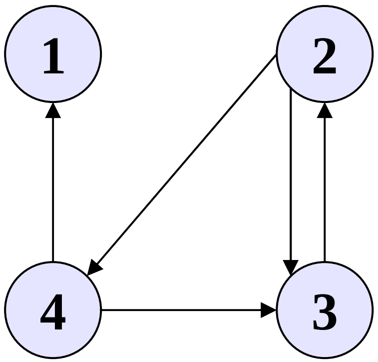
$$n \leftarrow |V[G]|$$
$$t_{ij}^{(k)} \leftarrow t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge d_{kj}^{(k-1)} \right)$$

Total Running Time = $\Theta(n^3)$

Transitive Closure

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

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



Johnson's Algorithm

- For sparse graphs, Johnson's Algorithm is asymptotically better than
  - Repeated squaring of matrices and
  - The Floyd-Warshall algorithm.
- It uses as subroutines both
  - Dijkstra's algorithm and
  - The Bellman-Ford algorithm.
- It returns a matrix of shortest-path weights for all pairs of vertices OR
- Reports that the input graph contains a negative-weight cycle.
- This algorithm uses a technique of **reweighting**.

Johnson's Algorithm

Re-weighting

- The technique of **reweighting** works as follows.
  - If all edge weights are nonnegative, find shortest paths by running Dijkstra's algorithm, with Fibonacci heap priority queue, once for each vertex.
  - If *G* has negative-weight edges, we simply compute a new set of nonnegative edges weights that allows us to use the same method.
- New set of edge weights must satisfy the following
  - For all pairs of vertices $u, v \in V$, a shortest path from u to v using weight function w is also a shortest path from u to v using weight function w'.
  - For all (u, v), new weight $w'(u, v)$ is nonnegative

δ, δ' Preserving Shortest Paths by Re-weighting

- From the lemma given in the next slide shows, it is easy to come up with a re-weighting of the edges that satisfies the first property above.
- We use δ to denote shortest-path weights derived from weight function w
- And δ' to denote shortest-path weights derived from weight function w'.
- And then we will show that, for all (*u*, *v*), new weight *w*' (*u*, *v*) is nonnegative.

Re-weighting does not change shortest paths

Lemma Statement

- Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \rightarrow R$, let $h : V \rightarrow R$ be any function mapping vertices to real numbers.
- For each edge (u, v) E, define

$$w'(u, v) = w(u, v) + h(u) - h(v)$$

- Let $p = \langle v_0, v_1, \ldots, v_k \rangle$ be any path from vertex v_0 to vertex v_k. Then p is a shortest path from v_0 to v_k with weight function w if and only if it is a shortest path with weight function w'.
- That is, $w(p) = \delta(v_0, v_k)$ if and only if $w'(p) = \delta'(v_0, v_k)$.
- Also, G has a negative-weight cycle using weight function w if and only if G has a negative-weight cycle using weight function w'.

Proof: Lemma

We start by showing that

$$w'(p) = w(p) + h(v_0) - h(v_k)$$

We have

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

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

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

$$= \sum_{i=1}^{k} w(p) + h(v_0) - h(v_k)$$

- Therefore, any path p from v_0 to v_k has w'(p) = w(p) + $h(v_0)$ – $h(v_k)$.
- If one path from v_0 to v_k is shorter than another using weight function w, then it is also shorter using w'.
- Thus,

  $w(p) = \delta(v_0, v_k,) \Leftrightarrow w'(p) = \delta'(v_0, v_k,)$.

Proof: Lemma

- Finally, we show that G has a negative-weight cycle using weight function w if and only if G has a negative-weight cycle using weight function w'.
- Consider any cycle

  $c = \langle v_0, v_1, \ldots, v_k \rangle$, where $v_0 = v_k$.
- Now

  $$w'(c) = w(c) + h(v_0) - h(v_k) = w(c),$$
- And thus c has negative weight using w if and only if it has negative weight using w'.
- It completes the proof of the theorem

Producing nonnegative weights by re-weighting

Next we ensure that second property holds i.e. $w'(u, v)$ to be nonnegative for all edges $(u, v) \in E$.

- Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \to R$, we make a new graph $G' = (V', E')$, where $V' = V \cup \{s\}$ for some new vertex $s \notin V$ and
- $E' = E \cup \{(s, v) : v \in V\}$.
- Extend weight function w so that $w(s, v) = 0$ for all $v \in V$.
- Note that because s has no edges that enter it, no shortest paths in G', other than those with source s, contain s.
- Moreover, G' has no negative-weight cycles if and only if G has no negative-weight cycles.

Producing nonnegative weights by re-weighting

- Now suppose that G and G' have no negative-weight cycles.
- Let us define $h(v) = \delta(s, v)$ for all $v \in V'$.
- By triangle inequality, we have

  $h(v) \leq h(u) + w(u, v), \quad \forall\ (u, v) \in E'.$ (1)
- Thus, if we define the new weights w', we have

  $w'(u, v) = w(u, v) + h(u) - h(v) \geq 0.$ by (1)
- And the second property is satisfied.

Johnson’s Algorithm

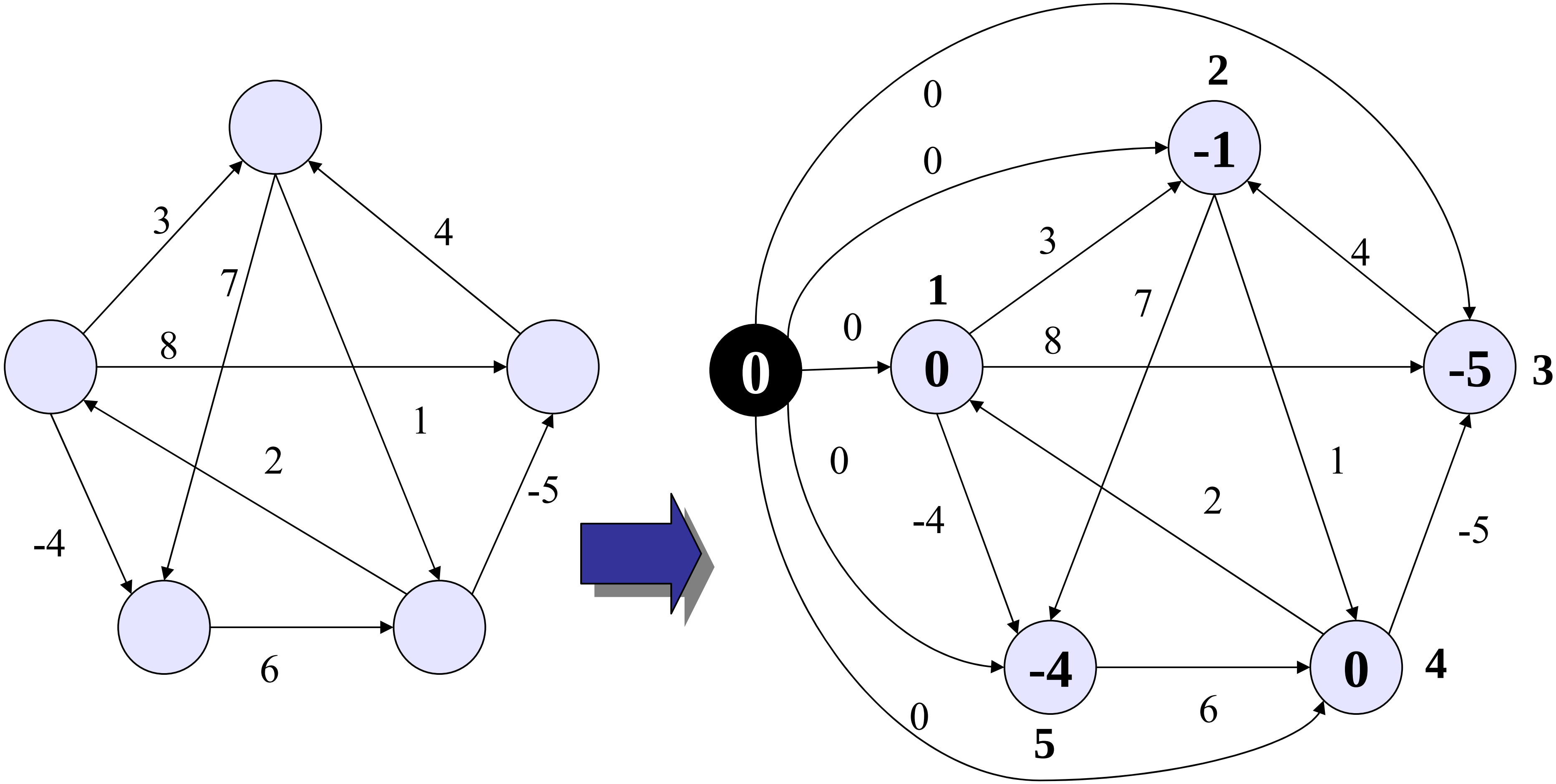
JOHNSON (G)

compute G', where $V[G'] = V[G] \cup \{s\}$,
$E[G'] = E[G] \cup \{(s, v) : v \in V[G]\}$, and
$w(s, v) = 0$ for all $v \in V[G]$

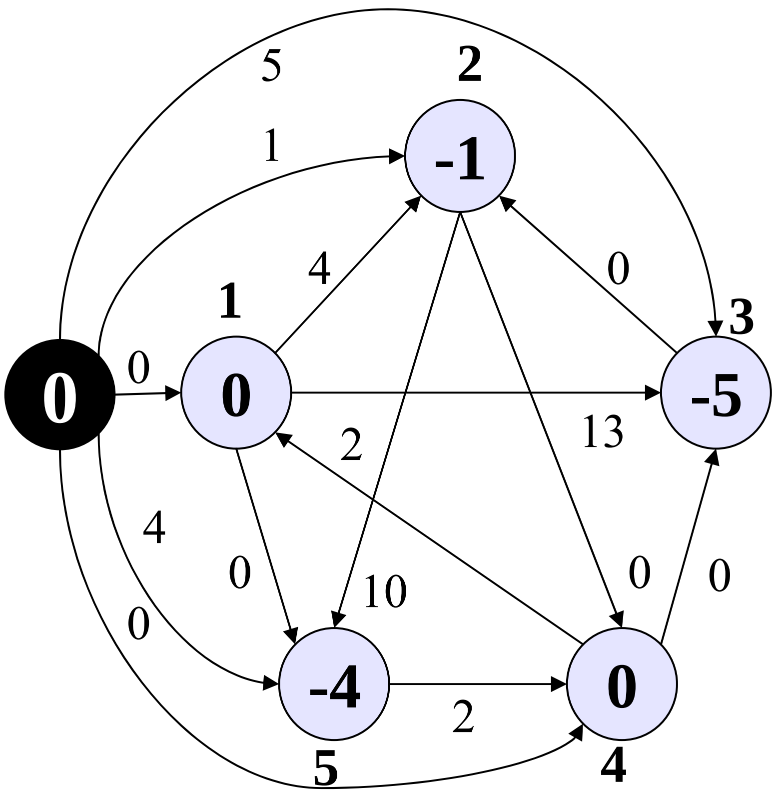
if BELLMAN-FORD(G', w, s) = FALSE
  then print “the input graph contains a negative-weight cycle”
  else for each vertex $v \in V[G']$
      do set $h(v)$ to the value of $\delta(s, v)$
        computed by the Bellman-Ford algorithm
    for each edge $(u, v) \in E[G']$
      do $\hat{w}(u, v) \leftarrow w(u, v) + h(u) - h(v)$
    for each vertex $u \in V[G]$
      do run DIJKSTRA(G, $\hat{w}$, u) to compute $\hat{\delta}(u, v)$ for all $v \in V[G]$
        for each vertex $v \in V[G]$
          do $d_{uv} \leftarrow \hat{\delta}(u, v) + h(v) - h(u)$
    return D

Total Running Time = $O(V^2 \lg V + VE)$

Johnson’s Algorithm



Bellman-Ford algorithm is used to determine $\delta(s, v)$ for all $v \in V$ e.g., $\delta(s, 3) = -5$ path: <s, 4, 3>



$\hat{w}(0,1) \leftarrow w(0,1) + h(0) - h(1) = (0 + 0 - (-0)) = 5$

$\hat{w}(0,2) \leftarrow w(0,2) + h(0) - h(2) = (0 + 0 - (-1)) = 1$

$\hat{w}(0,3) \leftarrow w(0,3) + h(0) - h(3) = (0 + 0 - (-5)) = 5$

$\hat{w}(0,4) \leftarrow w(0,4) + h(0) - h(4) = (0 + 0 - (-0)) = 0$

$\hat{w}(0,5) \leftarrow w(0,5) + h(0) - h(5) = (0 + 0 - (-4)) = 4$

$\hat{w}(1,2) \leftarrow w(1,2) + h(1) - h(2) = (3 + 0 - (-1)) = 4$

$\hat{w}(1,3) \leftarrow w(1,3)+ h(1) - h(3) = (8 + 0 - (-5)) = 13$

$\hat{w}(1,5) \leftarrow w(1,5) + h(1) - h(5) = (-4 + 0 - (-4)) = 0$

$\hat{w}(2,4) \leftarrow w(2,4) + h(2) - h(4) = (1 + (-1) - 0) = 0$

$\hat{w}(2,5) \leftarrow w(2,5) + h(2) - h(5) = (7+(-1)-(-4))= 10$

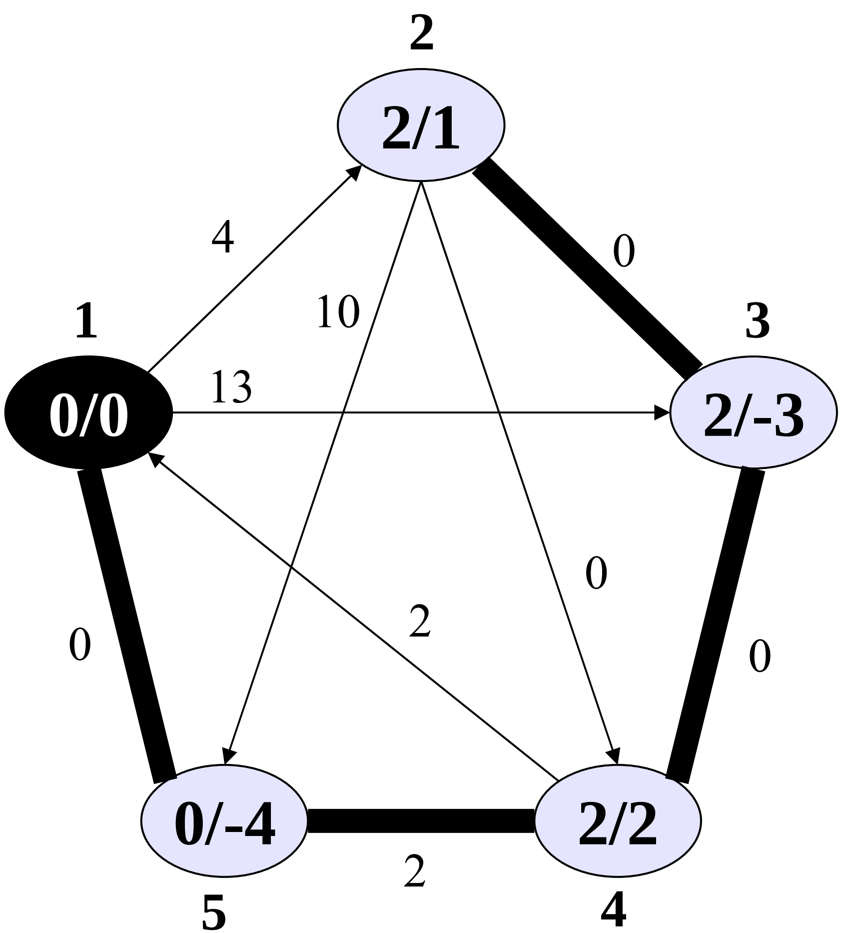
$\hat{w}(3,2) \leftarrow w(3,2) + h(3) - h(2) = (4+(-5)-(-1)) = 0$

$\hat{w}(4,1) \leftarrow w(4,1) + h(4) - h(1) = (2+0-0) = 2$

$\hat{w}(4,3) \leftarrow w(4,3) + h(4) - h(3) = (-5+0-(-5)) = 0$

$\hat{w}(5,4) \leftarrow w(5,4) + h(5) - h(4) = (6+(-4)-0)= 2$

Johnson's Algorithm



Applying Dijkstra's Algorithm on vertex 1

$\hat{\delta}(1,5) \leftarrow 0,$

$\delta(1,5) \leftarrow -4$

$d(1,5) \leftarrow \delta(1,5) = -4$

$\hat{\delta}(5,4) \leftarrow 2,$

$\delta(5,4) \leftarrow 2$

$d(5,4) \leftarrow \delta(5,4) = 2$

$\hat{\delta}(4,3) \leftarrow 2,$

$\delta(4,3) \leftarrow -3$

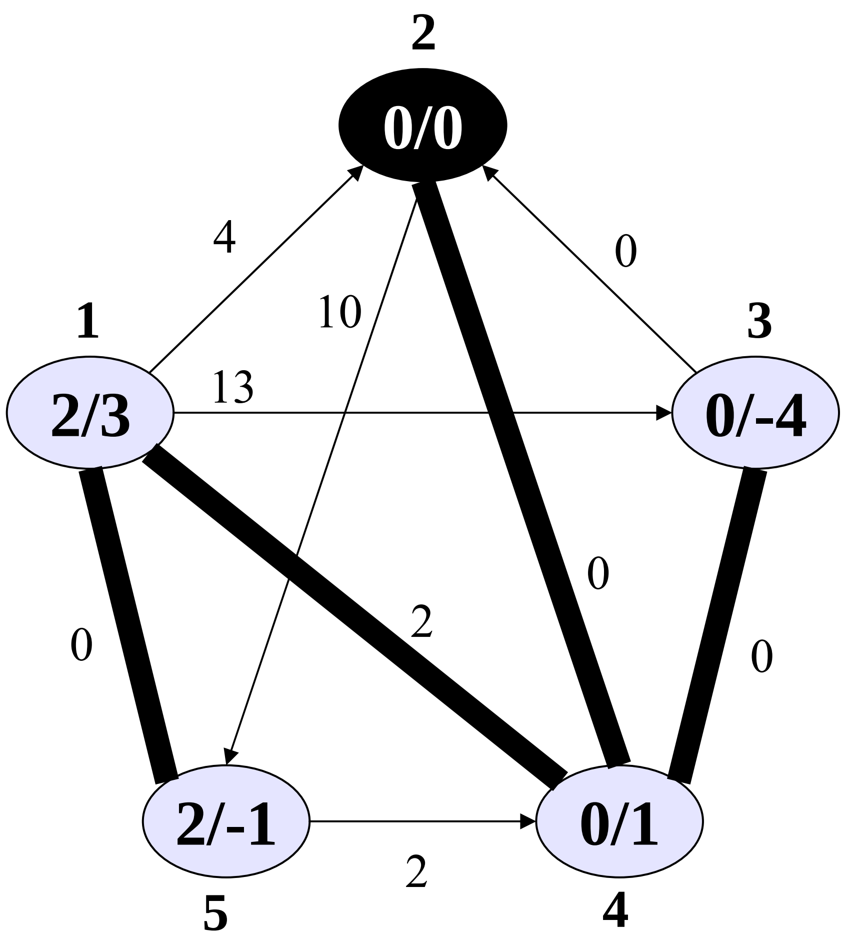
$d(4,3) \leftarrow \delta(4,3) = -3$

$\hat{\delta}(3,2) \leftarrow 2,$

$\delta(3,2) \leftarrow 1$

$d(3,2) \leftarrow \delta(3,2) = 1$

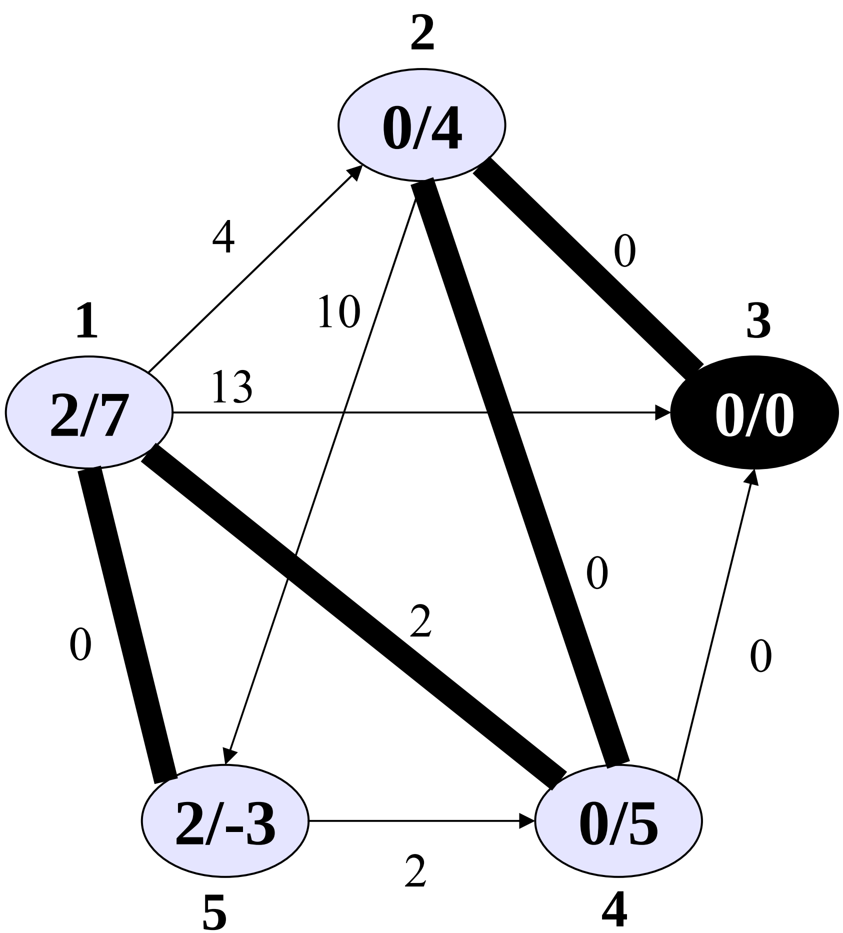
Johnson's Algorithm



Applying Dijkstra's Algorithm on vertex 2

$\hat{\delta}(2,4) \leftarrow 0,$
$\delta(2,4) \leftarrow 1$
$d(2,4) \leftarrow \delta(2,4) = 1$
$\hat{\delta}(4,1) \leftarrow 2,$
$\delta(4,1) \leftarrow 3$
$d(4,1) \leftarrow \delta(4,1) = 3$
$\hat{\delta}(4,3) \leftarrow 0,$
$\delta(4,3) \leftarrow -4$
$d(4,3) \leftarrow \delta(4,3) = -4$
$\hat{\delta}(1,5) \leftarrow 2,$
$\delta(1,5) \leftarrow -1$
$d(1,5) \leftarrow \delta(1,5) = -1$

Johnson's Algorithm



Applying Dijkstra's Algorithm on vertex 3

$\hat{\delta}(3,2) \leftarrow 0,$

$\delta\,(3,2) \leftarrow 4$

$d\,(3,2) \leftarrow \delta(3,2) = 4$

$\hat{\delta}(2,4) \leftarrow 0,$

$\delta\,(2,4) \leftarrow 5$

$d\,(2,4) \leftarrow \delta(2,4) = 5$

$\hat{\delta}(4,1) \leftarrow 2,$

$\delta\,(4,1) \leftarrow 7$

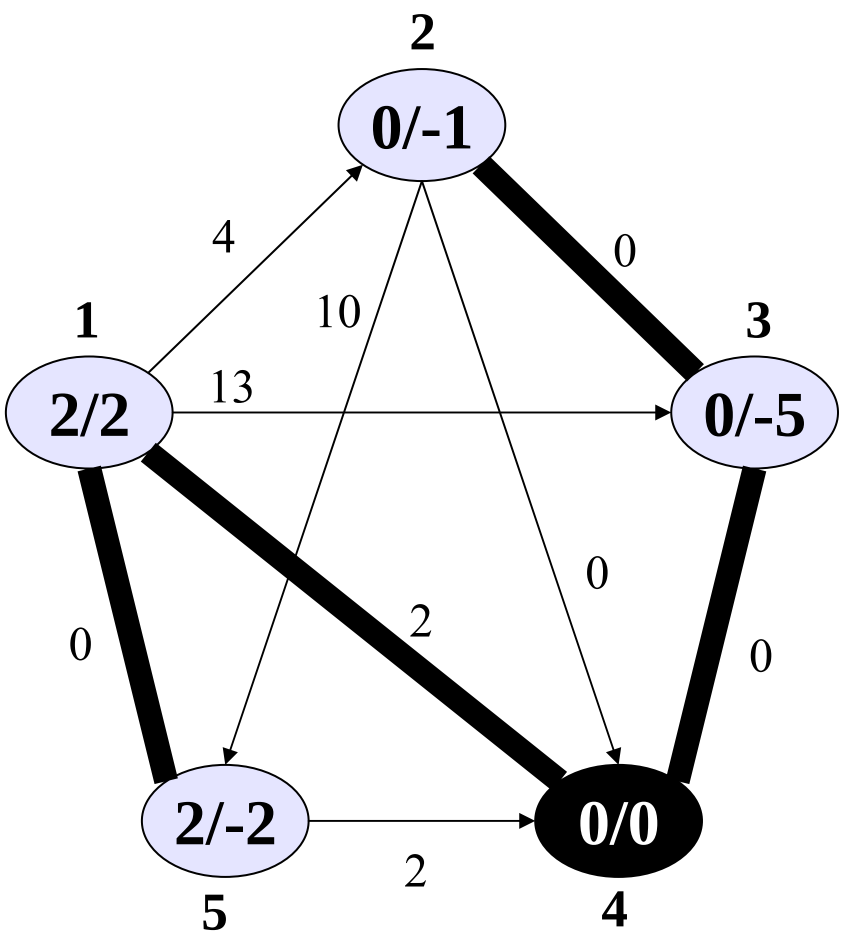
$d\,(4,1) \leftarrow \delta(4,1) = 7$

$\hat{\delta}(1,5) \leftarrow 2,$

$\delta\,(1,5) \leftarrow 3$

$d\,(1,5) \leftarrow \delta(1,5) = 3$

Johnson's Algorithm



Applying Dijkstra's Algorithm on vertex 4

$\hat{\delta}(4,1) \leftarrow 2,$

$\delta(4,1) \leftarrow 2$

$d(4,1) \leftarrow \delta(4,1) = 2$

$\hat{\delta}(4,3) \leftarrow 0,$

$\delta(4,3) \leftarrow -5$

$d(4,3) \leftarrow \delta(4,3) = -5$

$\hat{\delta}(1,5) \leftarrow 2,$

$\delta(1,5) \leftarrow -2$

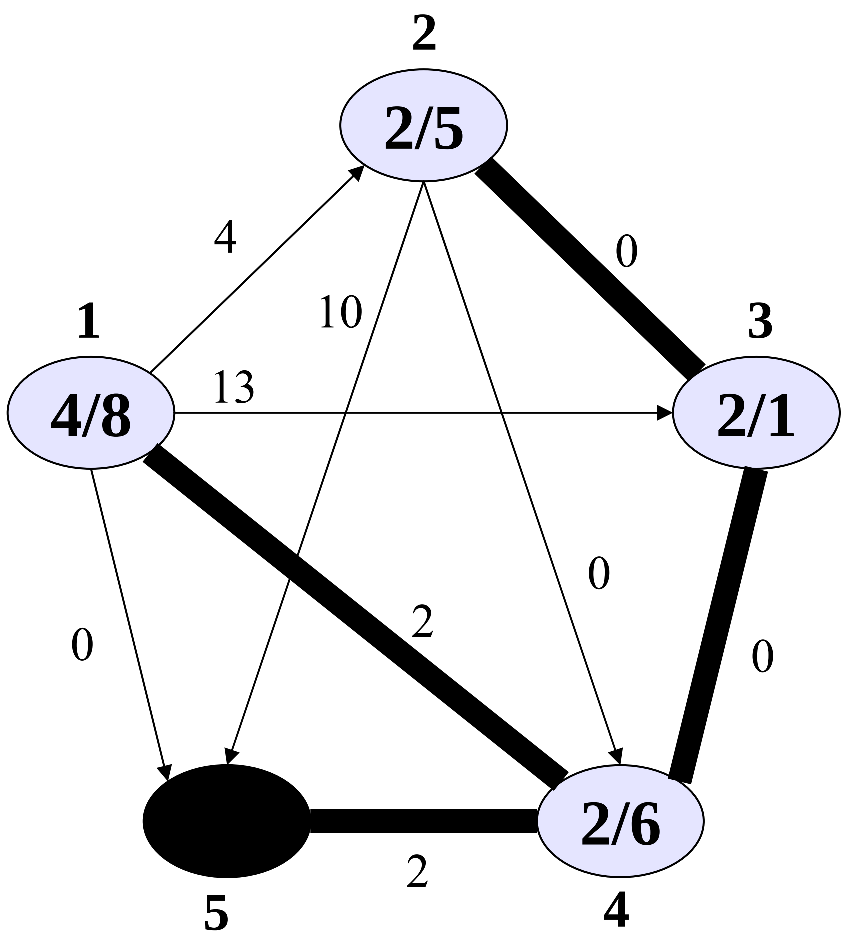
$d(1,5) \leftarrow \delta(1,5) = -2$

$\hat{\delta}(3,2) \leftarrow 0,$

$\delta(3,2) \leftarrow -1$

$d(3,2) \leftarrow \delta(3,2) = -1$

Johnson's Algorithm



Applying Dijkstra's Algorithm on vertex 5

$\hat{\delta}(5,4) \leftarrow 2,$
$\delta(5,4) \leftarrow 6$
$d(5,4) \leftarrow \delta(5,4) = 6$
$\hat{\delta}(4,1) \leftarrow 4,$
$\delta(4,1) \leftarrow 8$
$d(4,1) \leftarrow \delta(4,1) = 8$
$\hat{\delta}(4,3) \leftarrow 2,$
$\delta(4,3) \leftarrow 1$
$d(4,3) \leftarrow \delta(4,3) = 1$
$\hat{\delta}(3,2) \leftarrow 2,$
$\delta(3,2) \leftarrow 5$
$d(3,2) \leftarrow \delta(3,2) = 5$

Conclusion