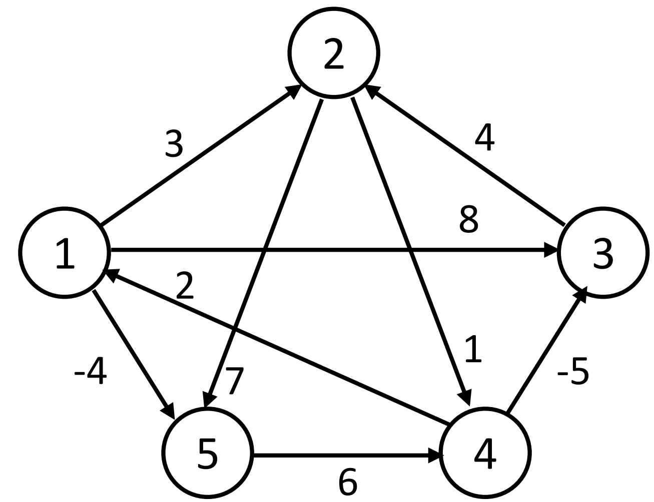
All Pairs Shortest Paths

Floyd-Warshall Algorithm

Floyd-Warshall Algorithm

- **Given:**
  - Directed, weighted graph G = (V, E)
  - Negative-weight edges may be present
  - No negative-weight cycles could be present in the graph
- **Compute:**
  - The shortest paths between all pairs of vertices in a graph

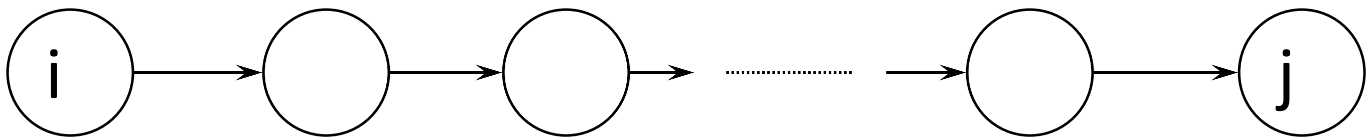


Floyd-Warshall Algorithm

- Represent the directed, edge-weighted graph in adjacency-matrix form.
- W= matrix of weights = $\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$
- w_{ij} is the weight of edge (i, j), or ∞ if there is no such edge.
- Return a matrix D, where each entry d_{ij} is $\delta(i,j)$.
- Could also return a predecessor matrix, P, where each entry p_{ij} is the predecessor of j on the shortest path from i.

Floyd-Warshall: Idea

- Consider *intermediate vertices* of a path:

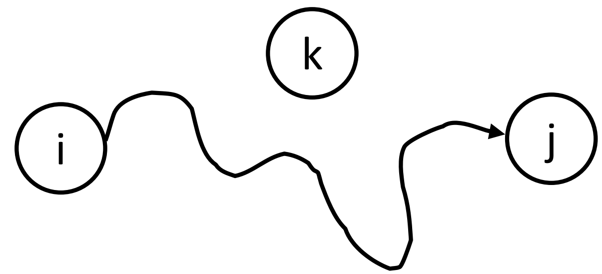


- Say we know the length of the shortest path from i to j whose intermediate vertices are only those with numbers {1,2,...,k-1} Call this length $d_{ij}^{(k-1)}$.

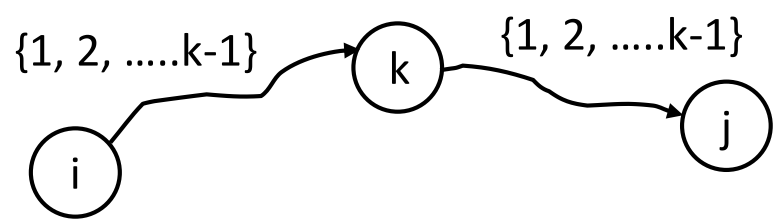
- Now how can we extend this from k-1 to k? In other words, we want to compute d_{ij}^{k} . Can we use $d_{ij}^{(k-1)}$, and if so how?

The Structure of a Shortest Path

- k is not an intermediate vertex of the shortest path p
  - Shortest path from i to j with intermediate vertices from {1, 2, …, k} is a shortest path from i to j with intermediate vertices from {1, 2, …, k - 1}



- k is an intermediate vertex of the shortest path p
  - p_1 is a shortest path from i to k
  - p_2 is a shortest path from k to j
  - k is not intermediary vertex of p_1, p_2
  - p_1 and p_2 are shortest paths from i to k with vertices from {1, 2, …, k - 1}



Floyd-Warshall Idea

- Thus,

$$d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$$

$$\text{Also, } \; d_{ij}^{(0)} = w_{ij}$$

- When k = |V|, we're done.

Dynamic Programming

- Floyd-Warshall is a *dynamic programming* algorithm:
  - Compute and store solutions to sub-problems. Combine those solutions to solve larger sub-problems.
  - Here, the sub-problems involve finding the shortest paths through a subset of the vertices.

Floyd-Warshall Algorithm

Floyd-Warshall(W)

$n \leftarrow rows[W]$ // number of vertices

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

for $k \leftarrow 1$ to n

 do for $i \leftarrow 1$ to n

  do for $j \leftarrow 1$ to n

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

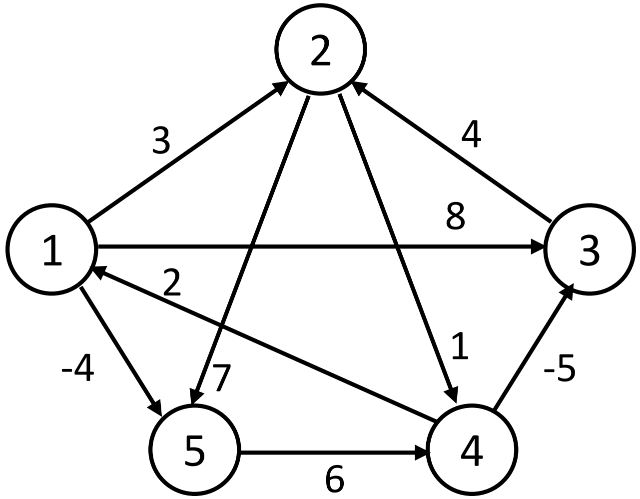
return $D^{(n)}$

Running Time: O(n^3)

Example of Floyd-Warshall

$D^{(0)} = W$

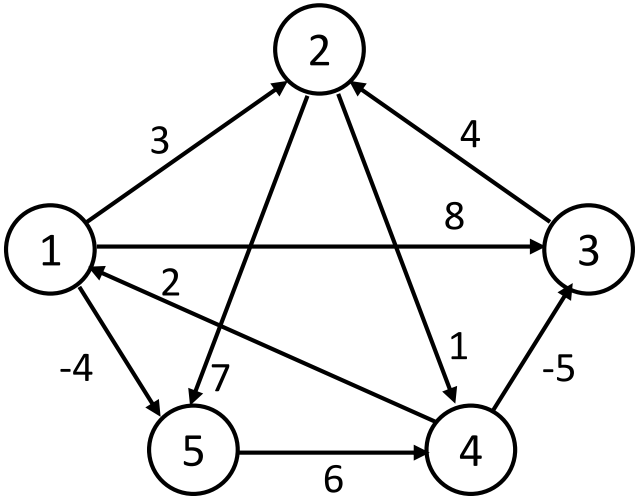
	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	∞	-5	0	∞
5	∞	∞	∞	6	0



Example of Floyd-Warshall

$D^{(1)}$

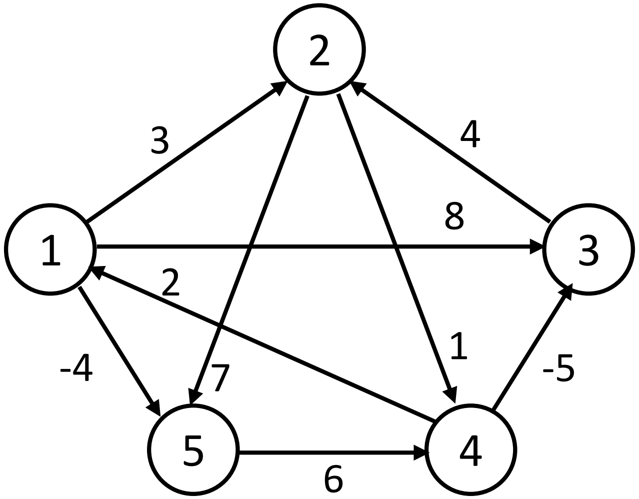
	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	5	-5	0	-2
5	∞	∞	∞	6	0



Example of Floyd-Warshall

$D^{(2)}$

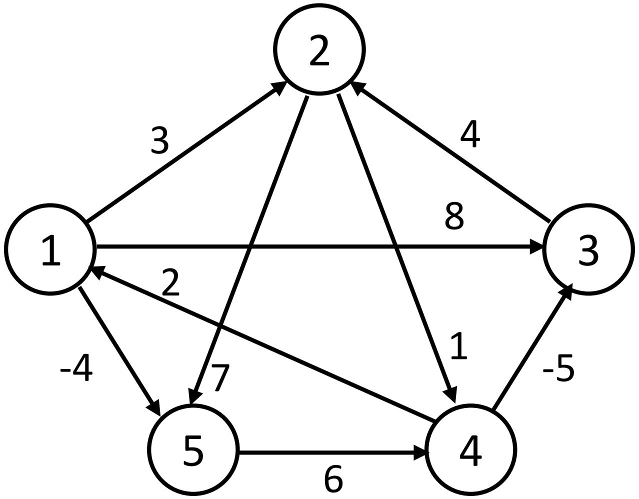
	1	2	3	4	5
1	0	3	8	4	-4
2	∞	0	∞	1	7
3	∞	4	0	5	11
4	2	5	-5	0	-2
5	∞	∞	∞	6	0



Example of Floyd-Warshall

$D^{(3)}$

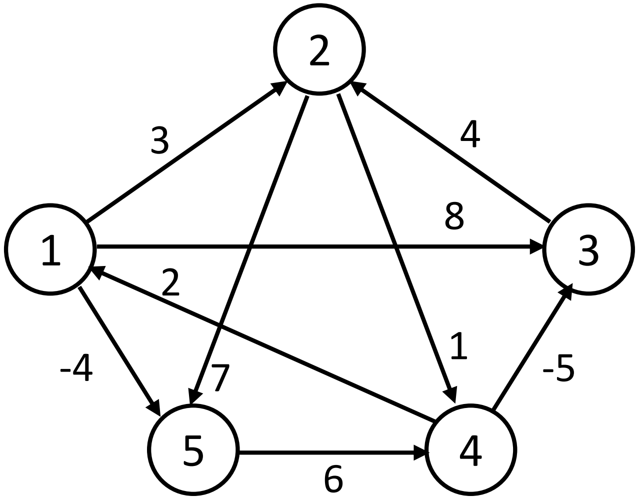
	1	2	3	4	5
1	0	3	8	4	-4
2	∞	0	∞	1	7
3	∞	4	0	5	11
4	2	-1	-5	0	-2
5	∞	∞	∞	6	0



Example of Floyd-Warshall

$D^{(4)}$

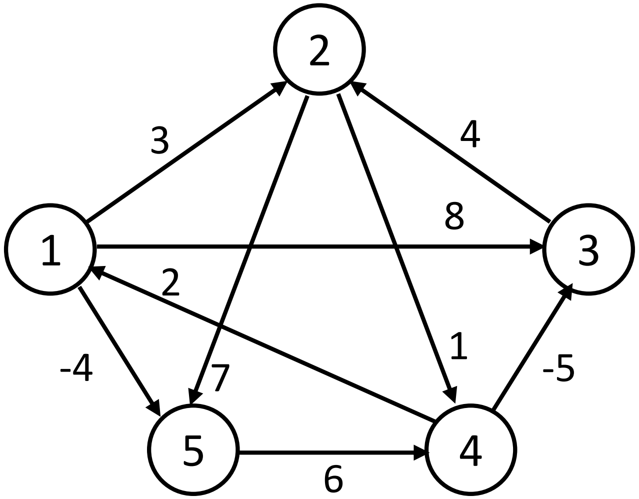
	1	2	3	4	5
1	0	3	-1	4	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2
5	8	5	1	6	0



Example of Floyd-Warshall

$D^{(5)}$

	1	2	3	4	5
1	0	1	-3	2	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2
5	8	5	1	6	0



Computing Predecessor Matrix

- *How do we compute the predecessor matrix?*
  - Initialization:

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

  - Updating:

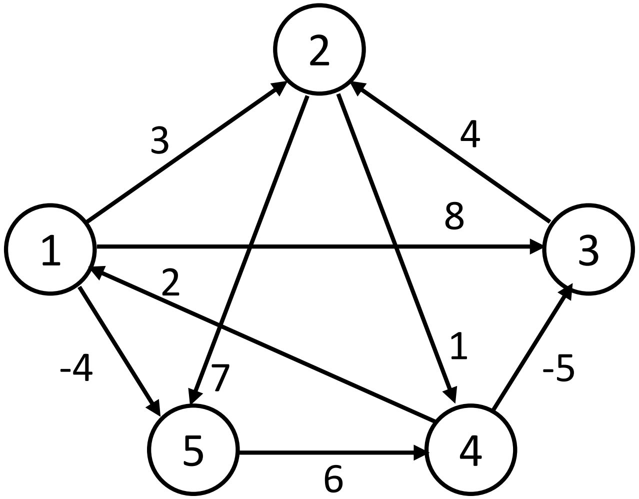
$$p^{(k)}(i,j) = p^{(k-1)}(i,j) \text{ if}(d^{(k-1)}(i,j) \leq d^{(k-1)}(i,k)+(d^{(k-1)}(k,j)$$

$$p^{(k-1)}(k,j) \text{ if}(d^{(k-1)}(i,j) > d^{(k-1)}(i,k)+(d^{(k-1)}(k,j)$$

Example of Floyd-Warshall

$P^{(5)}$

	1	2	3	4	5
1	-	3	4	5	1
2	4	-	4	2	1
3	4	3	-	2	1
4	4	3	4	-	1
5	4	3	4	5	-



Source: 5, Destination: 1

Shortest path: 8

Path: 5 …1 : 5…4…1: 5->4…1: 5->4->1

Source: 1, Destination: 3

Shortest path: -3

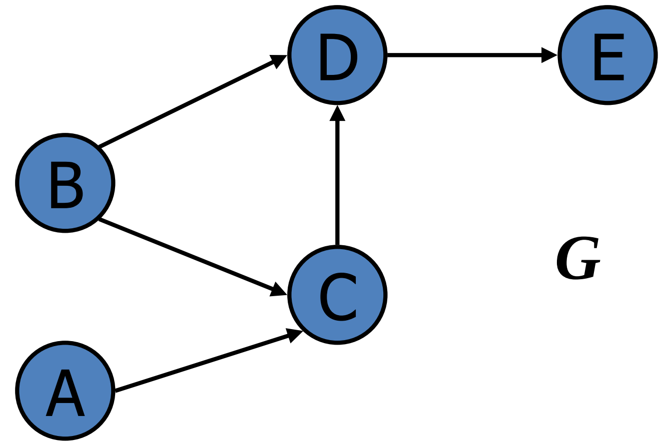
Path: 1 …3 : 1…4…3: 1…5…4…3: 1->5->4->3

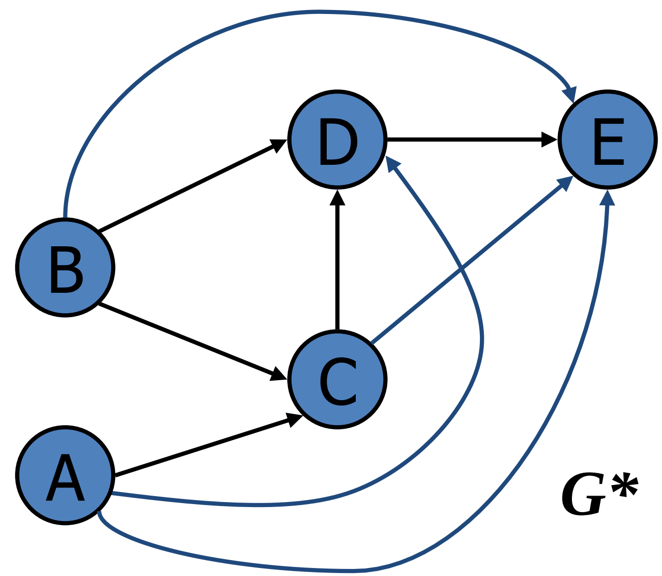
PrintPath for Warshall’s Algorithm

```
PrintPath(s, t)
{
  if(P[s][t]==nil) {print("No path");return;}
  else if (P[s][t]==s){
      print(s);
  }
  else{
      print_path(s,P[s][t]);
      print_path(P[s][t], t);
  }
}
Print (t) at the end of the PrintPath(s,t)
```

Transitive Closure

- Given a digraph ***G***, the transitive closure of ***G*** is the digraph ***G****** such that
  - ***G****** has the same vertices as ***G***
  - if ***G*** has a directed path from ***u*** to ***v*** (***u*** $\neq$ ***v***), ***G****** has a directed edge from ***u*** to ***v***
- The transitive closure provides reachability information about a digraph





Transitive closure of the graph

- Input:
  - Un-weighted graph G: $W[i][j] = 1$, if $(i,j) \in E$, $W[i][j] = 0$ otherwise.
- Output:
  - $T[i][j] = 1$, if there is a path from i to j in G, $T[i][j] = 0$ otherwise.
- Algorithm:
  - Just run Floyd-Warshall with weights 1, and make $T[i][j] = 1$, whenever $D[i][j] < \infty$.
  - More efficient: use only Boolean operators

Transitive Closure Algorithm

```
Transitive-Closure(W[1..n][1..n])
T ← W    // T^(0)
for k ← 1 to n do // compute T^(k)
    for i ←1 to n do
        for i ←1 to n do
            T[i][j] ← T[i][j] ∨ (T[i][k] ∧ T[k][j])
return T
```

All Pairs Shortest Paths

Johnson's algorithm for Sparse Graphs

Johnson’s Algorithm

- The Floyd-Warshall algorithm takes $O(n^3)$ for sparse graph or dense graph.
- Can we find better algorithm for sparse graph?
  - Use single source shortest path algorithm.
    - No negative edges: use Dijkstra n times $\longrightarrow \Theta(n^2 \lg n)$.
    - Problem if we have negative weight edges

      $\longrightarrow$ Cannot use Dijkstra

      $\longrightarrow$ Using Bellman-Ford is not good enough $\longrightarrow \Theta(n^3)$.
- Johnson’s algorithm combines Bellman-Ford and Dijkstra.

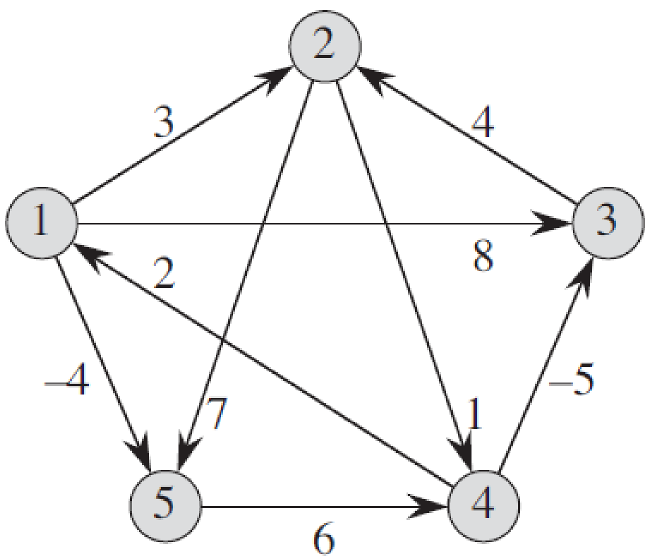
Johnson's Algorithm- Idea

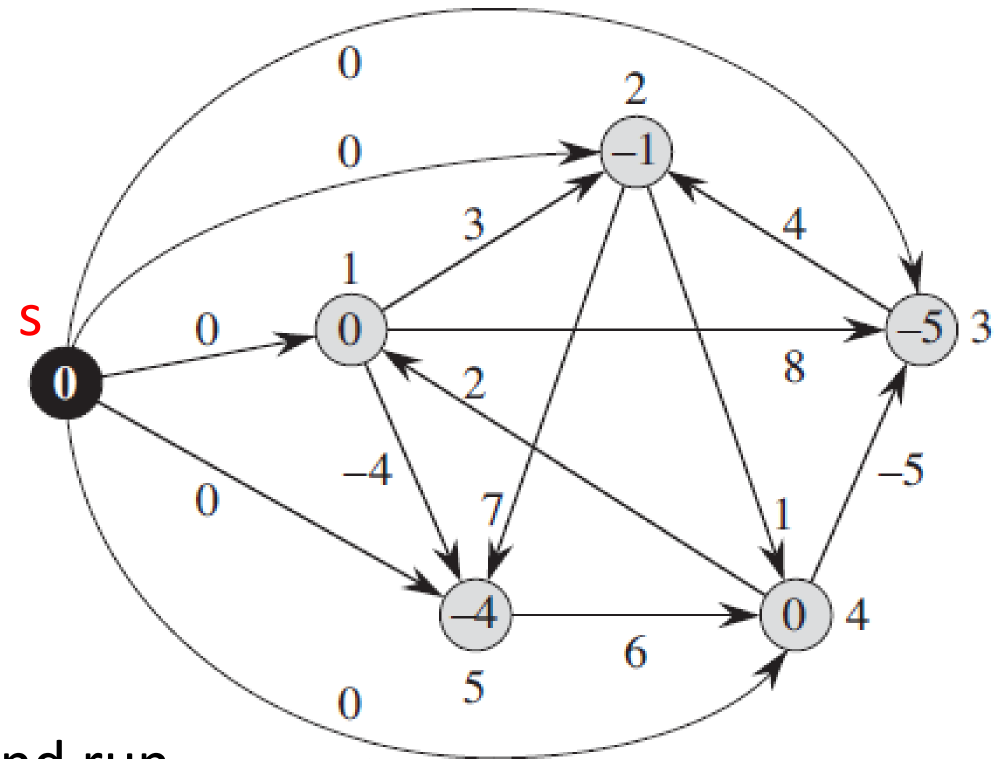
- Transforms any negative-weighted graph into a graph with positive weights.
- Such that the minimum path using the new weights is the same path for the original weights.
- (G, W) => (G, W_1)
  - w(u, v) can be negative
  - w_1(u, v) >= 0 for all (u, v)
  - p is a minimum path u..v in (G, W) => p is also a minimum path u..v in (G, W_1)

How to Compute W_1?

- New idea: build a new graph $G'(V', E')$
  - $V' = V \cup \{s\}$
  - $E' = E \cup \{(s, v)$ for all $v \in V\}$
  - $w'(s, v) = 0$ for all $v \in V$
  - $w'(u, v) = w(u, v)$ for all $u,v \in V$
- Run Bellman-Ford on G' from s
  - The result is $h(v)= \delta(s, v)$ in G' for all $v \in V$
  - $h(v)$ may be positive or negative
  - We can also detect the negative cycles in G'
- The new weight function for G is:
  - $w_1(u, v) = w(u, v) + h(u) - h(v) >= 0$ for all $(u, v) \in E$

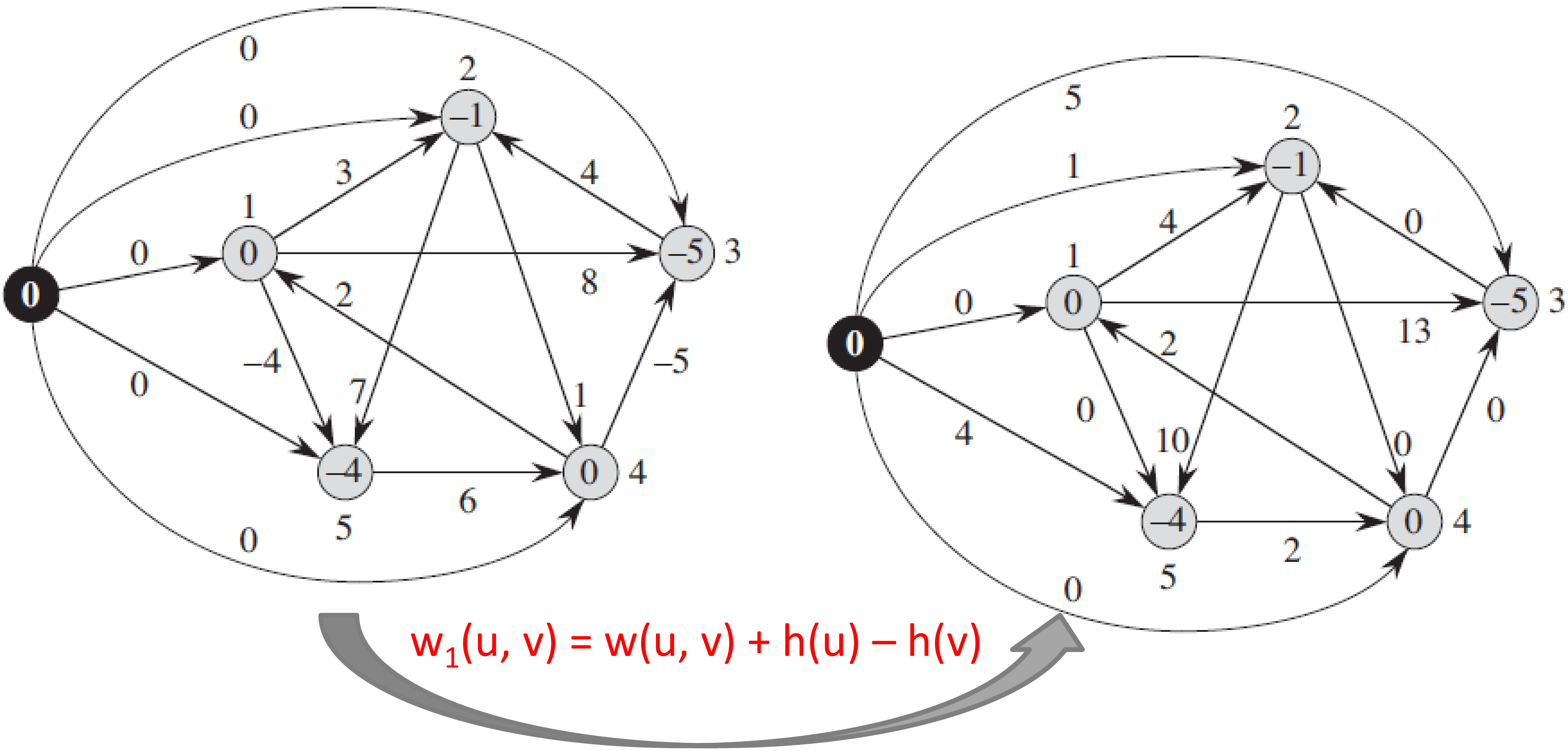
Example





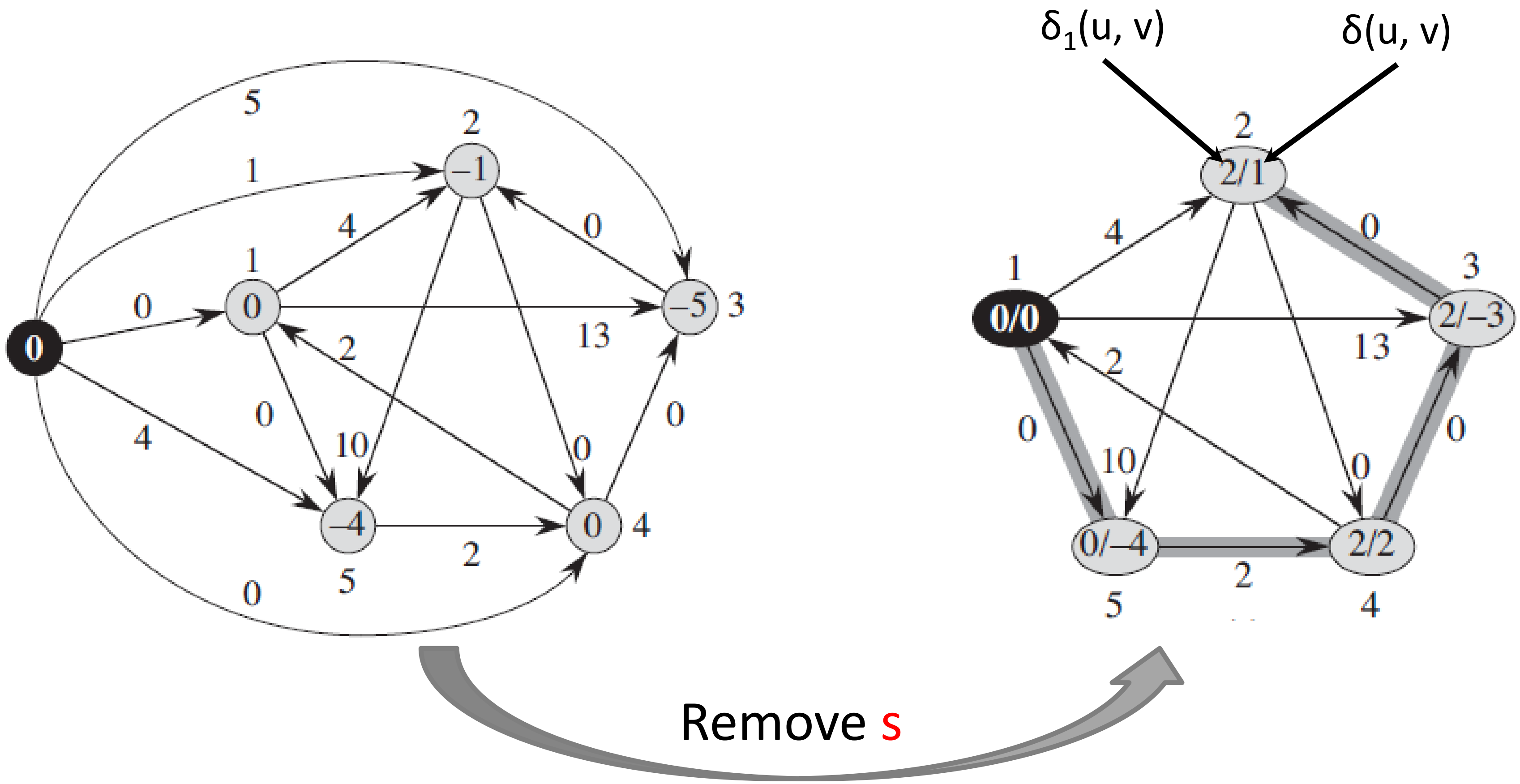
Add s and run Bellman-Ford on the new graph

Example



$$w_1(u, v) = w(u, v) + h(u) - h(v)$$

Example

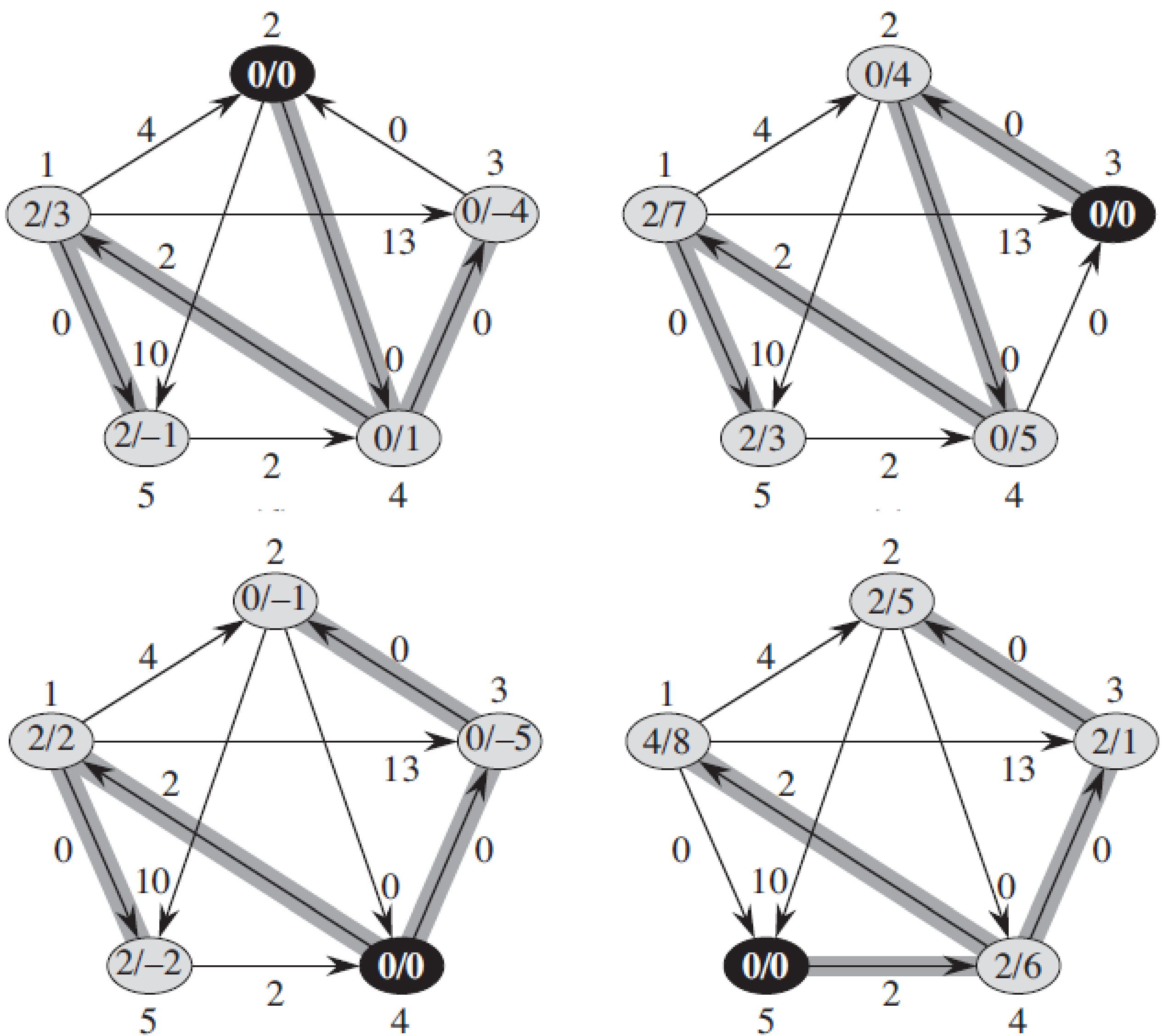


Run Dijkstra from each vertex => $\delta_1(u, v)$

Recompute the distances:

$$\delta(u, v) = \delta 1(u, v) + h(v) - h(u)$$

Example



	1	2	3	4	5
1	0	1	-3	2	-4
2	3	0	-4	1	-1
3	7	4	0	5	3
4	2	-1	-5	0	-2
5	8	5	1	6	0

Pseudocode

JOHNSON(G,w)

```
compute G′, where G′.V = G.V ∪ {s},
    G′.E = G.E ∪ {(s, v) : v ∈ G.V}, and
    w(s, v) = 0 for all v ∈ G.V
if BELLMAN-FORD(G′, w, s) == FALSE
    print "the input graph contains a negative-weight cycle"
else for each vertex v ∈ G′.V
        set h(v) to the value of δ(s, v)
            computed by the Bellman-Ford algorithm
    for each edge (u, v) ∈ G′.E
        ŵ(u, v) = w(u, v) + h(u) − h(v)
    let D = (d_uv) be a new n × n matrix
    for each vertex u ∈ G.V
        run DIJKSTRA(G, ŵ, u) to compute δ̂(u, v) for all v ∈ G.V
        for each vertex v ∈ G.V
            d_uv = δ̂(u, v) + h(v) − h(u)
    return D
```

Analysis

JOHNSON(G,w)

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 $v \in G.V$
if BELLMAN-FORD(G', w, s) == FALSE O(VE)
print "the input graph contains a negative-weight cycle"
else for each vertex $v \in G'.V$ O(V)
set $h(v)$ to the value of $\delta(s, v)$
computed by the Bellman-Ford algorithm
for each edge $(u, v) \in G'.E$ O(E)
$\hat{w}(u, v) = w(u, v) + h(u) - h(v)$
let $D = (d_{uv})$ be a new $n \times n$ matrix
for each vertex $u \in G.V$ O(V)
run DIJKSTRA$(G, \hat{w}, u)$ to compute $\hat{\delta}(u, v)$ for all $v \in G.V$ O(VlgV)
for each vertex $v \in G.V$ O(V)
$d_{uv} = \hat{\delta}(u, v) + h(v) - h(u)$
return D

Time complexity: $O(V^2 lgV+VE)$ => $O(V^2 lgV)$ for sparse graph