

CSE 2202
Design and Analysis of
Algorithms – I

All Pair Shortest Path

All-Pairs Shortest Paths - Solutions

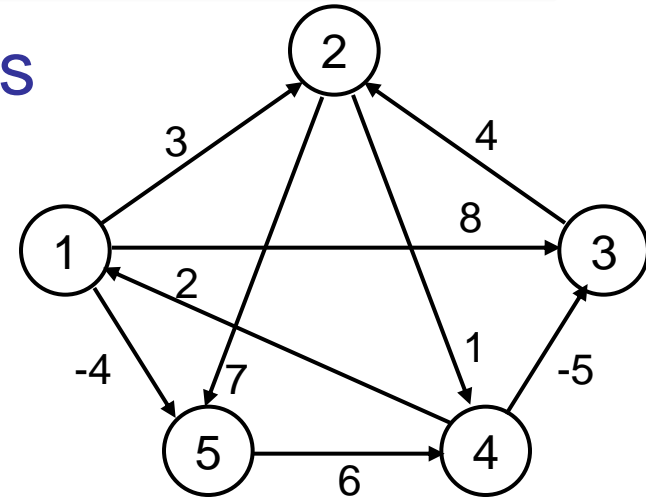
- Run **BELLMAN-FORD** once from each vertex:
 - $O(V^2E)$, which is $O(V^4)$ if the graph is dense
($E = \Theta(V^2)$)
- If no negative-weight edges, could run **Dijkstra's** algorithm once from each vertex:
 - $O(VE \lg V)$ with binary heap, $O(V^3 \lg V)$ if the graph is dense
- We can solve the problem in $O(V^3)$, with no elaborate data structures

All-Pairs Shortest Paths

- Assume the graph (G) is given as adjacency matrix of weights

- $W = (w_{ij})$, $n \times n$ matrix, $|V| = n$
- Vertices numbered 1 to n

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



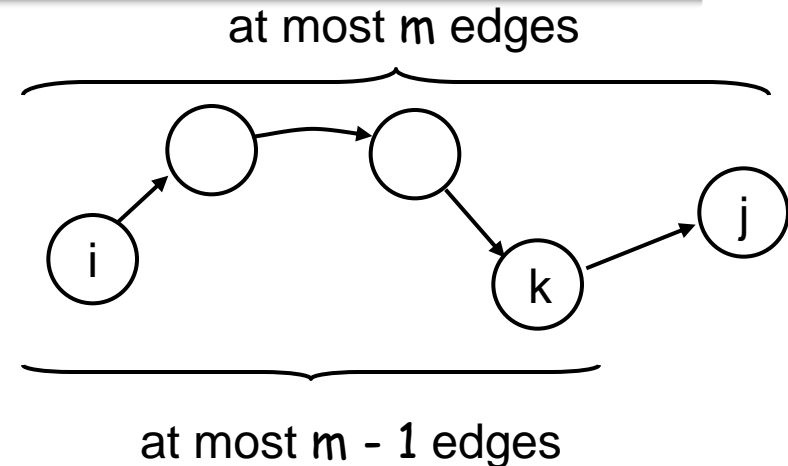
- Output the result in an $n \times n$ matrix
 $D = (d_{ij})$, where $d_{ij} = \delta(i, j)$
- Solve the problem using dynamic programming

Optimal Substructure of a Shortest Path

- All subpaths of a shortest path are shortest paths

- Let p : a shortest path p from vertex i to j that contains at most m edges

- If $i = j$
 - $w(p) = 0$ and p has no edges



- If $i \neq j$: $p = i \xrightarrow{p'} k \rightarrow j$

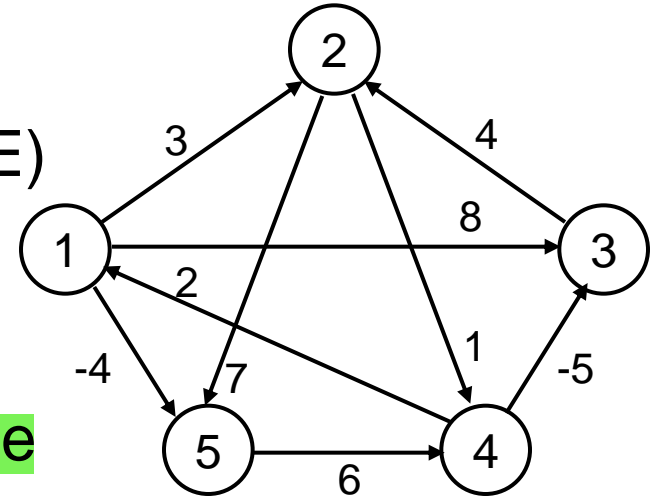
- p' has at most $m-1$ edges
- p' is a shortest path

$$\delta(i, j) = \delta(i, k) + w_{kj}$$

The 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

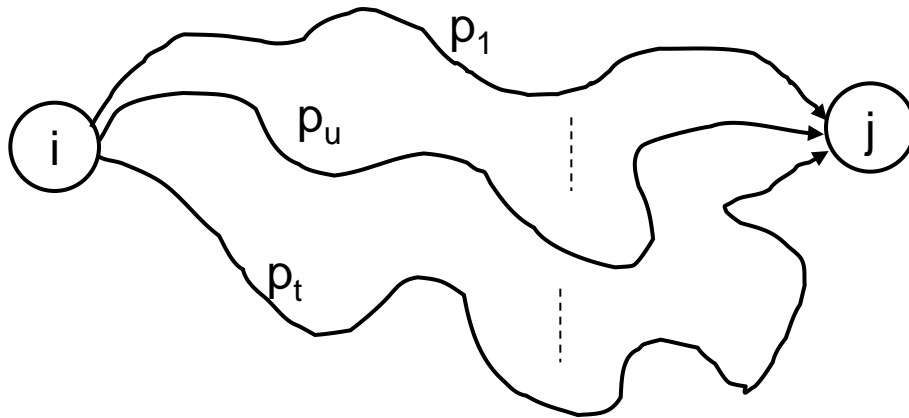


- **Compute:**

- The shortest paths between all pairs of vertices in a graph

The Structure of a Shortest Path

- For any pair of vertices $i, j \in V$, consider all paths from i to j whose intermediate vertices are all drawn from a subset $\{1, 2, \dots, k\}$
 - Find p , a minimum-weight path from these paths

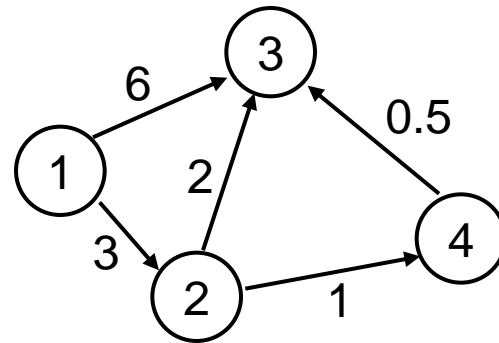


No vertex on these paths has index $> k$

Example

$d_{ij}^{(k)}$ = the weight of a shortest path from vertex i to vertex j with all intermediary vertices drawn from $\{1, 2, \dots, k\}$

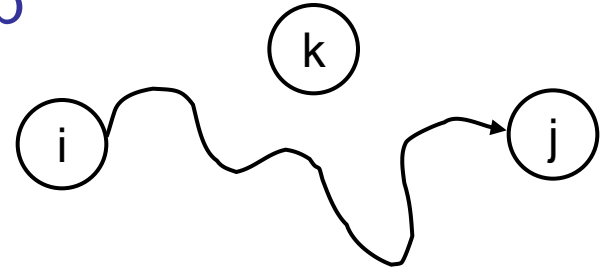
- $d_{13}^{(0)} = 6$
- $d_{13}^{(1)} = 6$
- $d_{13}^{(2)} = 5$
- $d_{13}^{(3)} = 5$
- $d_{13}^{(4)} = 4.5$



The Structure of a Shortest Path

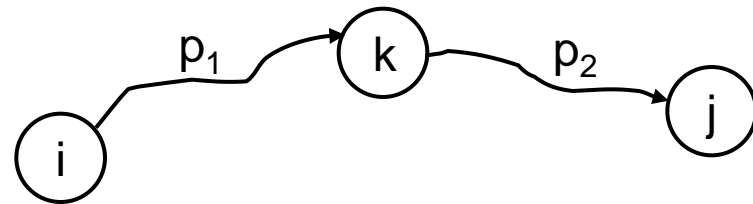
- k is not an intermediate vertex of path p

- Shortest path from i to j with intermediate vertices from $\{1, 2, \dots, k\}$ is a shortest path from i to j with intermediate vertices from $\{1, 2, \dots, k - 1\}$



- k is an intermediate vertex of 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 $\{1, 2, \dots, k - 1\}$



A Recursive Solution (cont.)

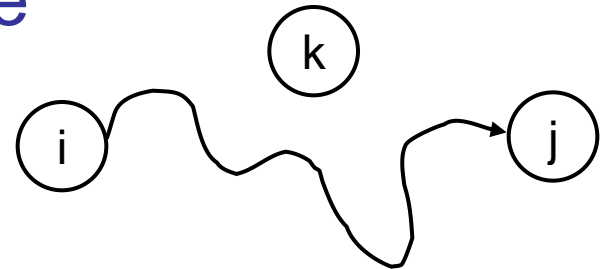
$d_{ij}^{(k)}$ = the weight of a shortest path from vertex i to vertex j with all intermediary vertices drawn from $\{1, 2, \dots, k\}$

- $k = 0$
- $d_{ij}^{(k)} = w_{ij}$

A Recursive Solution (cont.)

$d_{ij}^{(k)}$ = the weight of a shortest path from vertex i to vertex j with all intermediary vertices drawn from $\{1, 2, \dots, k\}$

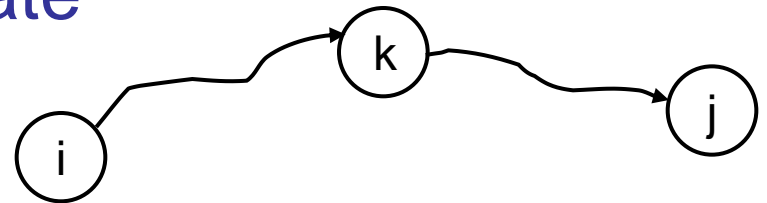
- $k \geq 1$
- **Case 1:** k is not an intermediate vertex of path p
- $d_{ij}^{(k)} = d_{ij}^{(k-1)}$



A Recursive Solution (cont.)

$d_{ij}^{(k)}$ = the weight of a shortest path from vertex i to vertex j with all intermediary vertices drawn from $\{1, 2, \dots, k\}$

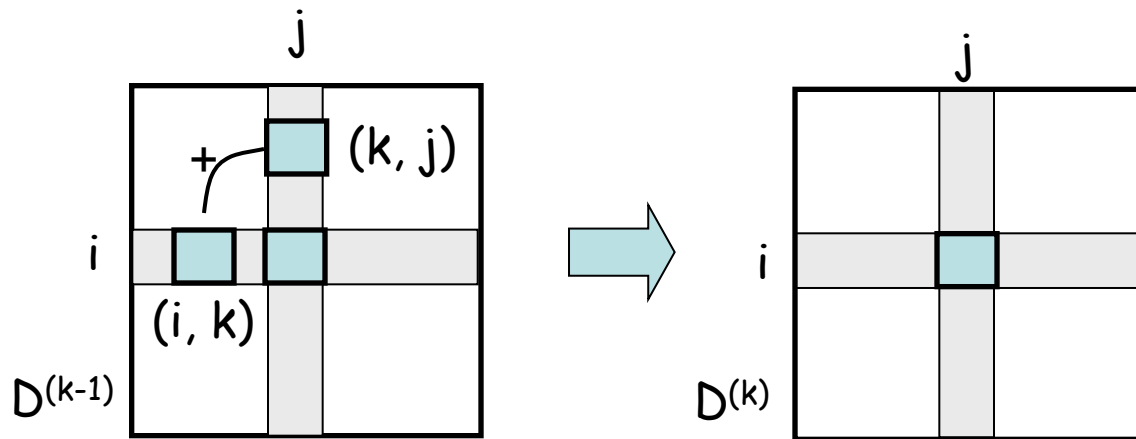
- $k \geq 1$
- **Case 2:** k is an intermediate vertex of path p
- $d_{ij}^{(k)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$



Computing the Shortest Path Weights

- $d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 \\ \min \{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\} & \text{if } k \geq 1 \end{cases}$
- The final solution: $D^{(n)} = (d_{ij}^{(n)})$:

$$d_{ij}^{(n)} = \delta(i, j) \quad \forall i, j \in V$$



The Floyd-Warshall algorithm

```
Floyd-Warshall (W[1..n][1..n])
01 D ← W      // D(0)
02 for k ← 1 to n do // compute D(k)
03     for i ← 1 to n do
04         for j ← 1 to n do
05             if D[i][k] + D[k][j] < D[i][j] then
06                 D[i][j] ← D[i][k] + D[k][j]
07 return D
```

Running Time: $O(n^3)$

Computing predecessor matrix

- *How do we compute the predecessor matrix?*

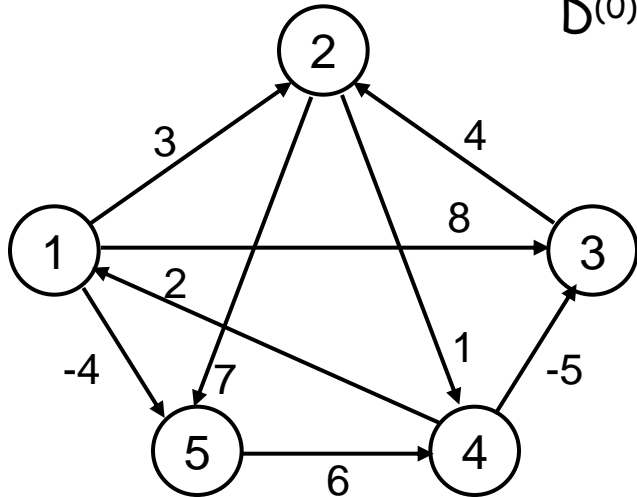
- Initialization:
$$p^{(0)}(i, j) = \begin{cases} \text{nil} & \text{if } i = j \text{ or } w_{ij} = \infty \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty \end{cases}$$

Floyd-Warshall ($W[1..n][1..n]$)

```
01 ...
02 for k ← 1 to n do // compute  $D^{(k)}$ 
03     for i ← 1 to n do
04         for j ← 1 to n do
05             if  $D[i][k] + D[k][j] < D[i][j]$  then
06                  $D[i][j] \leftarrow D[i][k] + D[k][j]$ 
07                  $P[i][j] \leftarrow k$ 
08 return D
```

Example

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



$D^{(0)} = W$

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

$D^{(1)}$

| | 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 |

$D^{(2)}$

| | 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 |

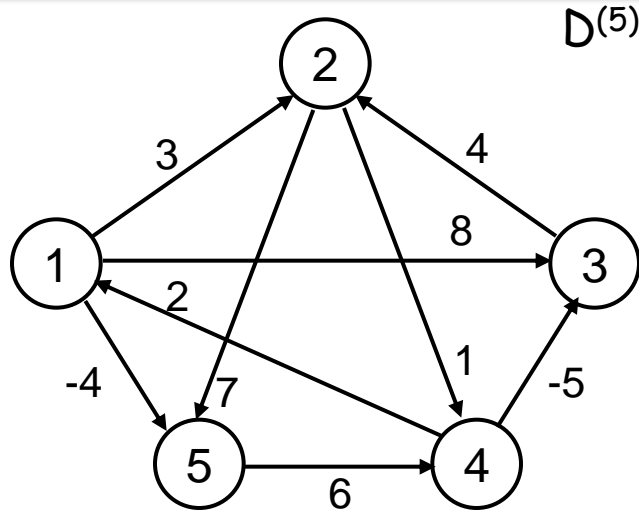
$D^{(3)}$

| | 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 |

$D^{(4)}$

| | 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

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


$D^{(5)}$

| | 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 |

$P^{(5)}$

| | 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

Example

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

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

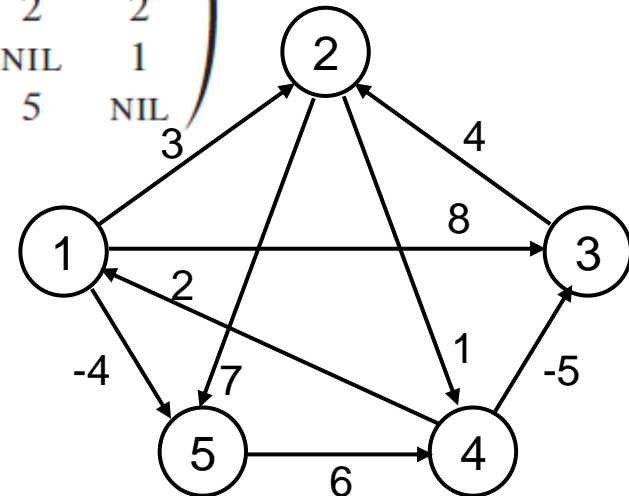
$$\Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

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

$$\Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

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

$$\Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$



Example

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

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

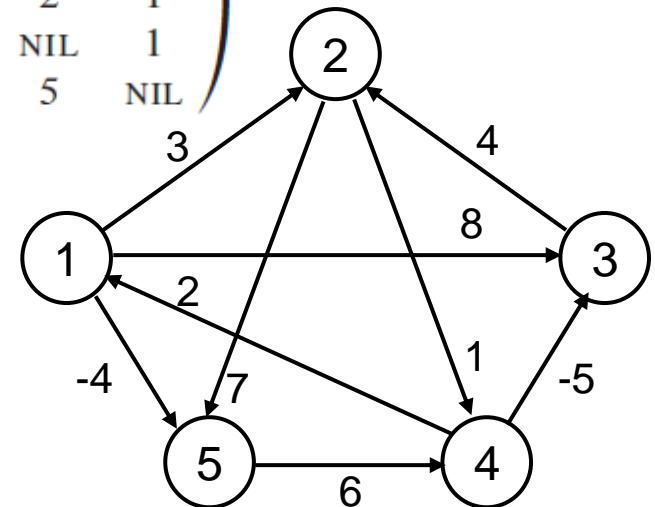
$$\Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

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

$$\Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

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

$$\Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$



Example

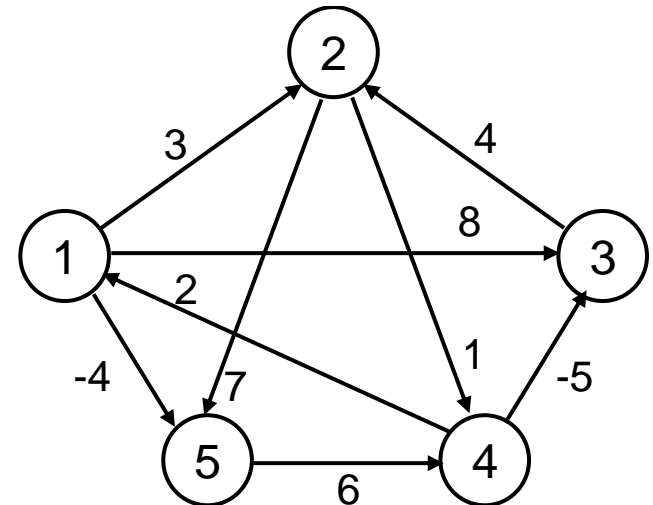
$$d_{ij}^{(k)} = \min \{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}$$

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

$$\Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

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

$$\Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$



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)

Question

- Why should we use $D[i, j]$ instead of $D^{(k)}[i, j]$?
- Exercise:
 - 25.2-4: Memory $O(n^2)$
 - 25.2-6: Negative weight cycle
 - Find the shortest positive cycle

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.

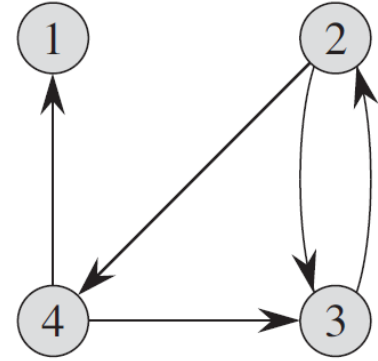
- Algorithm:

- 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 ($W[1..n][1..n]$)

```
01  $T \leftarrow W$       //  $T^{(0)}$ 
02 for  $k \leftarrow 1$  to  $n$  do // compute  $T^{(k)}$ 
03   for  $i \leftarrow 1$  to  $n$  do
04     for  $j \leftarrow 1$  to  $n$  do
05        $T[i][j] \leftarrow T[i][j] \vee (T[i][k] \wedge T[k][j])$ 
06 return  $T$ 
```



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

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

Complexity

- Bellman-Ford algorithm:

Running time: $O(VE)$

- Dijkstra's Algorithm

| Q | Total |
|----------------|------------------|
| array | $O(V^2)$ |
| binary heap | $O(E \lg V)$ |
| Fibonacci heap | $O(V \lg V + E)$ |

Complexity

- Run **BELLMAN-FORD** once from each vertex:
 - $O(V^2E)$, which is $O(V^4)$ if the graph is dense ($E = \Theta(V^2)$)
- If no negative-weight edges, could run **Dijkstra's** algorithm once from each vertex:
 - $O(\underline{VE} \lg V)$ with binary heap, $O(V^3 \lg V)$ if the graph is dense
- We can solve the problem in $O(V^3)$, with no elaborate data structures

Johnson's Algorithm

| Feature | Johnson's Algorithm | Warshall's (Floyd-Warshall) Algorithm |
|------------------|--|--|
| Time Complexity | $O(VE + V^2 \log V)$ | $O(V^3)$ |
| Space Complexity | $O(V + E)$ | $O(V^2)$ |
| Negative Weights | Handles negative weights (without negative cycles) | Handles negative weights (without negative cycles) |
| Optimality | More efficient for sparse graphs | More efficient for dense graphs |

Time Complexity: The main steps in the algorithm are Bellman-Ford Algorithm called once and Dijkstra called V times.

Time complexity of Bellman Ford is **$O(VE)$** and time complexity of Dijkstra is **$O(V \log V)$** . So overall time complexity is **$O(V^2 \log V + VE)$** .

Johnson's Algorithm

Johnson's algorithm uses the technique of **reweighting**

- If all edge weights **w** in a graph G are nonnegative, we can find shortest paths between all pairs of vertices by running Dijkstra's algorithm once from each vertex;
- with the Fibonacci-heap min-priority queue, the running time of this all-pairs algorithm is $O(V^2 \lg V + VE)$.
- If G has negative-weight edges but no negative-weight cycles, we simply compute **a new set of nonnegative edge weights** that allows us to use the same method...

Johnson's Algorithm

The new set of edge weights \hat{w} must satisfy two important properties:

1. For all pairs of vertices $u, v \in V$, a path p is a shortest path from u to v using weight function w if and only if p is also a shortest path from u to v using weight function \hat{w} .
2. For all edges (u, v) , the new weight $\hat{w}(u, v)$ is nonnegative.

As we shall see in a moment, we can preprocess G to determine the new weight function \hat{w} in $O(VE)$ time.

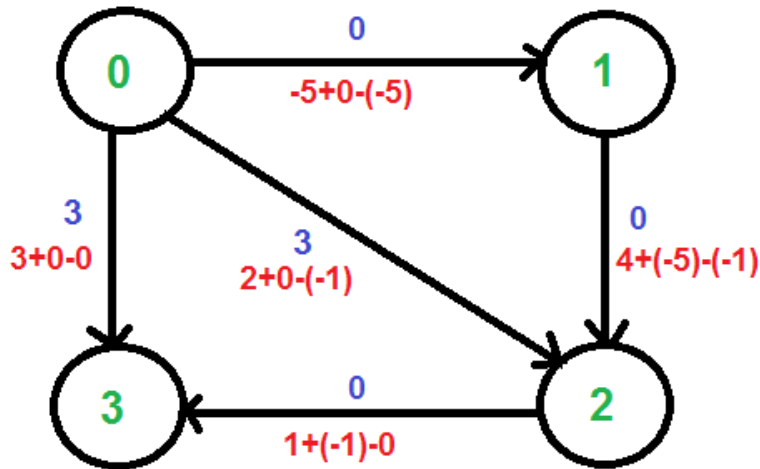
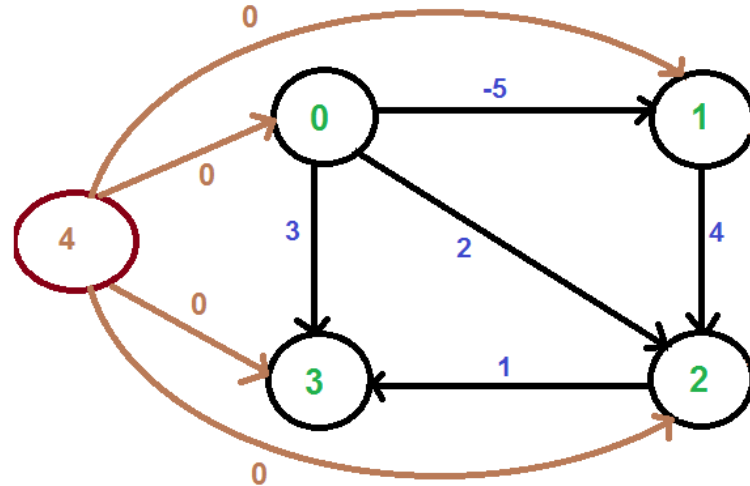
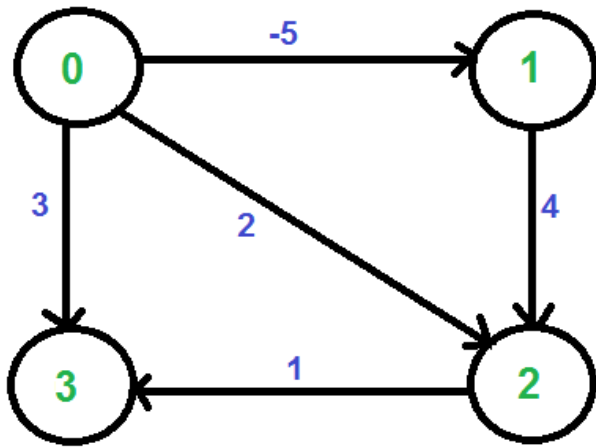
Lemma 25.1 (Reweighting does not change shortest paths)

Johnson's Algorithm

Johnson's algorithm has three main steps.

1. A new vertex is added to the graph, and it is connected by edges of zero weight to all other vertices in the graph.
2. All edges go through a reweighting process that eliminates negative weight edges.
3. The added vertex from step 1 is removed and Dijkstra's algorithm is run on every node in the graph.

Johnson's Algorithm

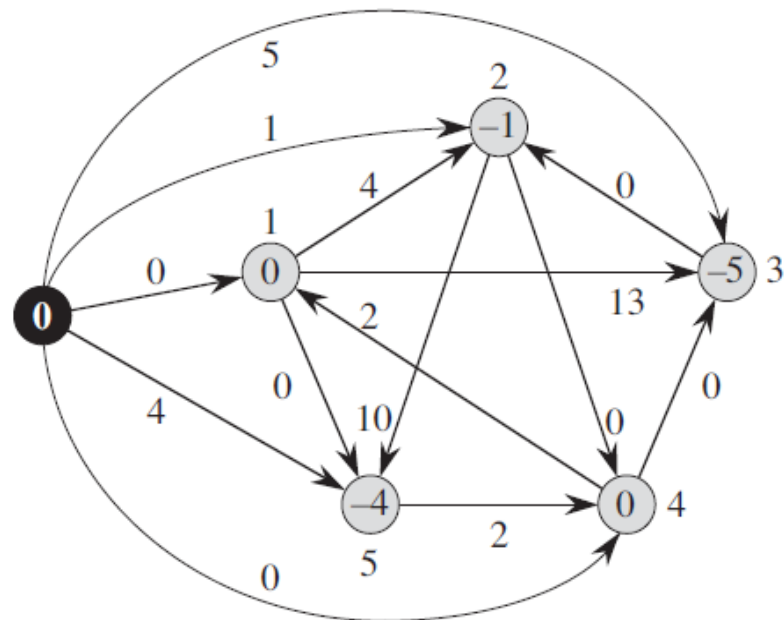
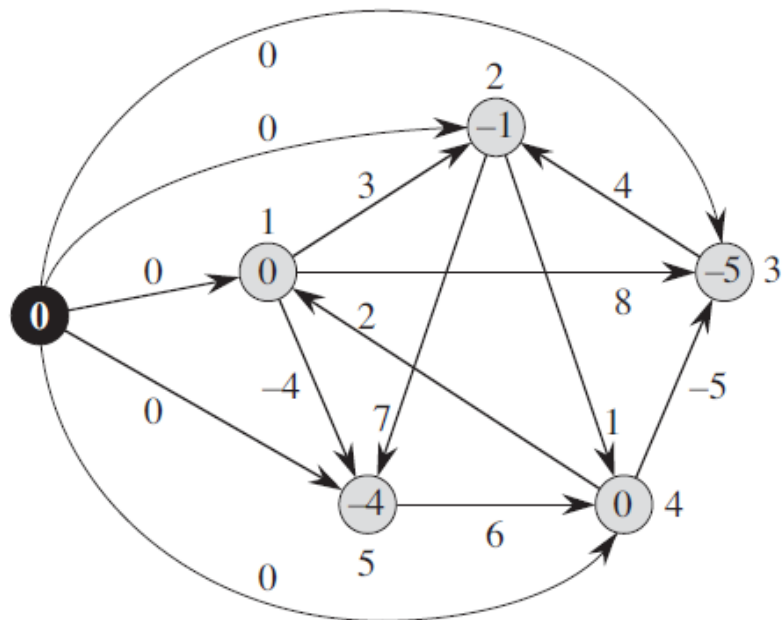


Distances from 4 to 0, 1, 2 and 3 are 0, -5, -1 and 0 respectively.

- We calculate the shortest distances from 4 to all other vertices using Bellman-Ford algorithm.
- The shortest distances from 4 to 0, 1, 2 and 3 are 0, -5, -1 and 0 respectively, i.e., $h[] = \{0, -5, -1, 0\}$.
- Once we get these distances, we remove the source vertex 4 and reweight the edges using following formula.

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

Johnson's Algorithm

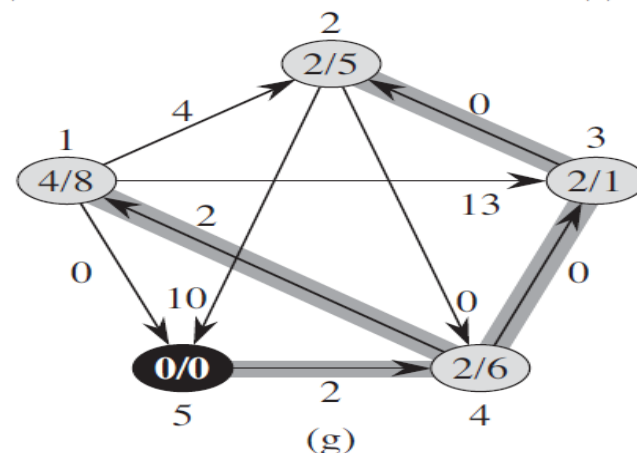
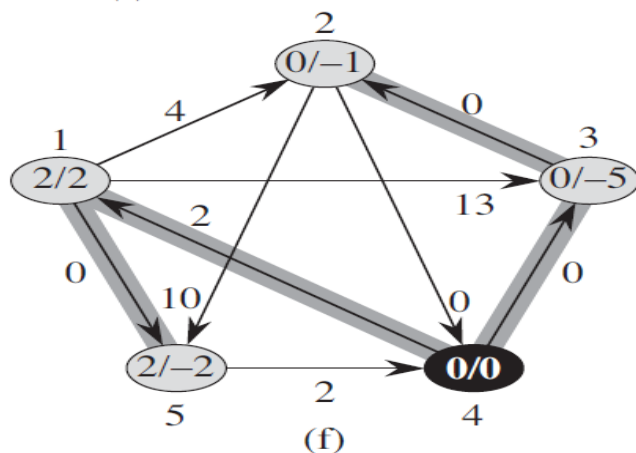
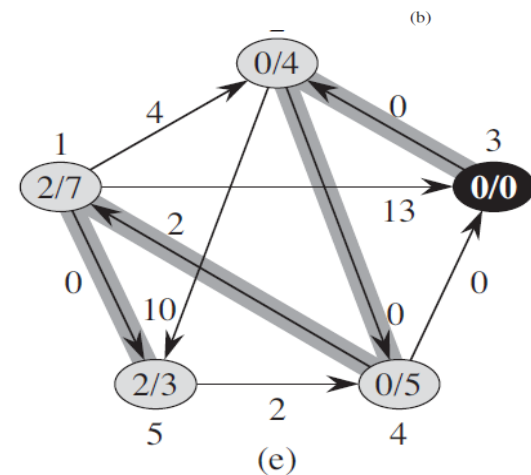
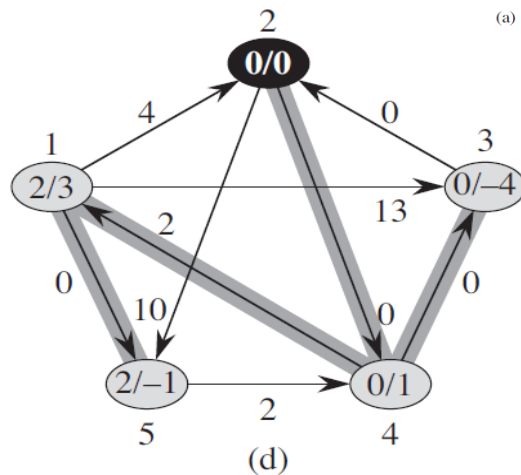
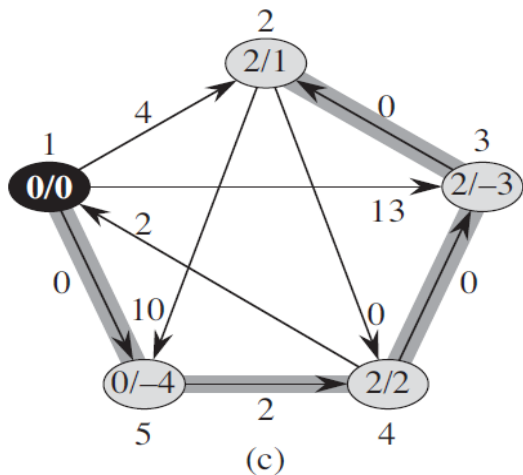
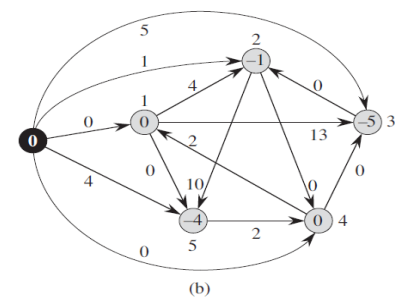
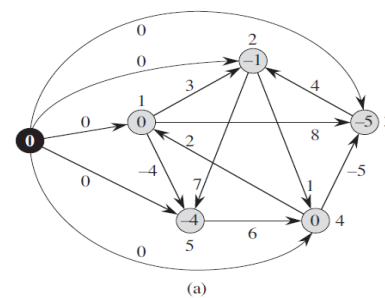


(a) The graph G' with the original weight function w .

The new vertex s is black. Within each vertex v is $h(v) = \delta(s, v)$.

(b) After reweighting each

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



Dijkstra's algorithm on each vertex of G using weight function \hat{w} . In each part, the source vertex u is black, and shaded edges are in the shortest-paths tree computed by the algorithm. Within each vertex v are the values $\hat{\delta}(u, v)$ and $\delta(u, v)$, separated by a slash. The value $d_{uv} = \delta(u, v)$ is equal to $\hat{\delta}(u, v) + h(v) - h(u)$.

JOHNSON(G, w)

```
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  $n \times n$  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          $d_{uv} = \hat{\delta}(u, v) + h(v) - h(u)$   
13     return  $D$ 
```

JOHNSON(G, w)

```
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  $n \times n$  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              $d_{uv} = \hat{\delta}(u, v) + h(v) - h(u)$   
13      return  $D$ 
```

If we implement the min-priority queue in Dijkstra’s algorithm by a Fibonacci heap, Johnson’s algorithm runs in $O(V^2 \lg V + VE)$ time. The simpler binary min-heap implementation yields a running time of $O(VE \lg V)$, which is still asymptotically faster than the Floyd-Warshall algorithm if the graph is sparse.