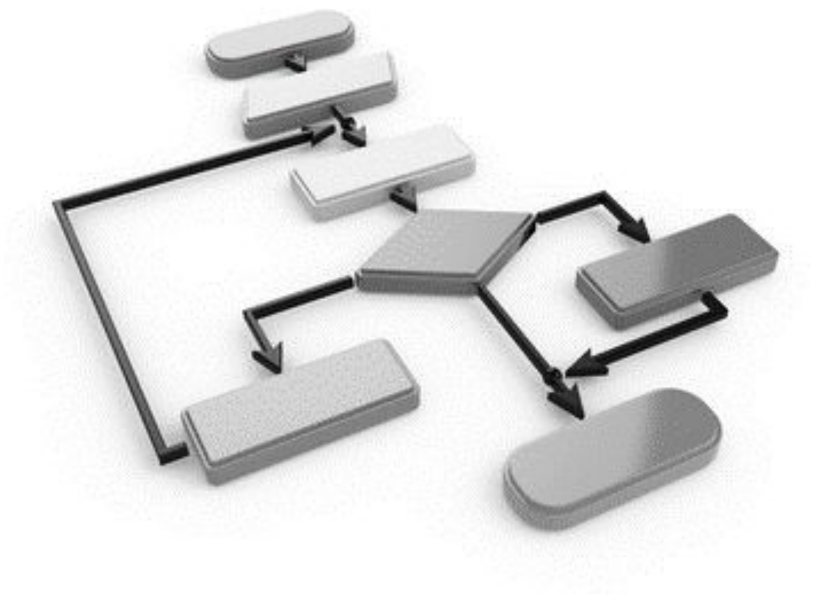
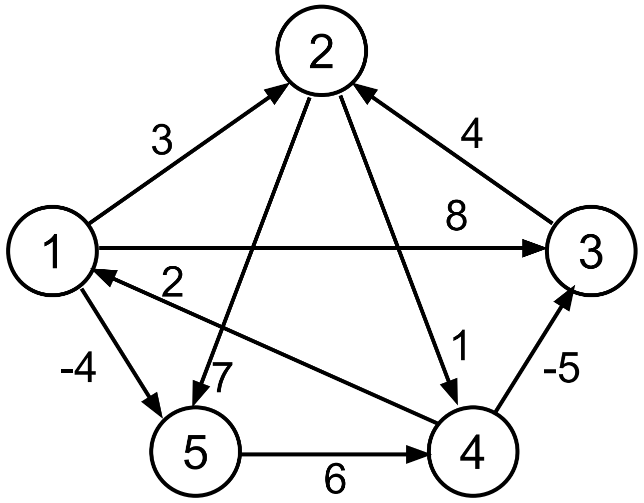


Graph-Based Algorithms

CSE 301: Combinatorial Optimization

All-Pairs Shortest Paths



Given:

Directed graph G = (V, E)

Weight function w : E → **R**

Compute:

The shortest paths between all pairs of vertices in a graph

Representation of the result: an n × n matrix of shortest-path distances δ(u, v)

Dijkstra (G, w, s)

```
1. INITIALIZE-SINGLE-SOURCE(V, s) ⟵ $\Theta(V)$
2. S ← ∅
3. Q ← V[G] ⟵ O(V) build min-heap
4. while Q ≠ ∅ ⟵ Executed O(V) times
5.     do u ← EXTRACT-MIN(Q) ⟵ O(lgV)
6.         S ← S ∪ {u}
7.         for each vertex v ∈ Adj[u]
8.             do RELAX(u, v, w) ⟵ O(E) times; O(lgV)
```

Running time: O(VlgV + ElgV) = O(ElgV)

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

```
1.  INITIALIZE-SINGLE-SOURCE(V, s)        ← Θ(V)
2.  for i ← 1 to |V| - 1                  ← O(V)  ⎫
3.      do for each edge (u, v) ∈ E       ← O(E)  ⎬ O(VE)
4.          do RELAX(u, v, w)                     ⎭
5.  for each edge (u, v) ∈ E              ← O(E)
6.      do if d[v] > d[u] + w(u, v)
7.          then return FALSE
8.  return TRUE
```

Running time: O(VE)

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 are present, we can run **Dijkstra's** algorithm once from each vertex:

$O(VE\lg V)$ with binary heap, which is $O(V^3\lg V)$ if the graph is dense

We can solve the problem in $O(V^3)$, with no elaborate data structures

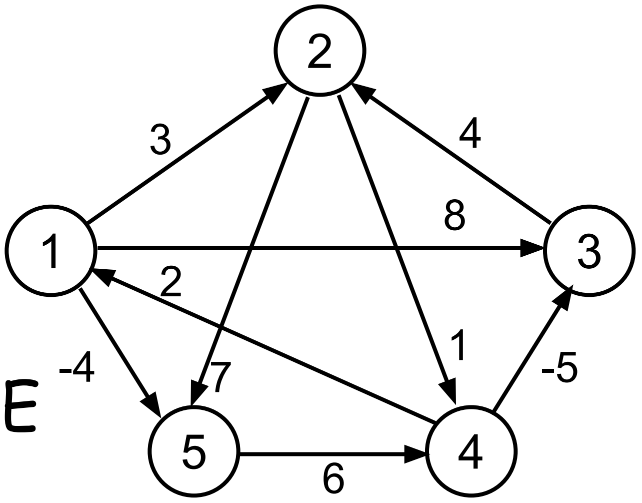
All-Pairs Shortest Paths

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

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

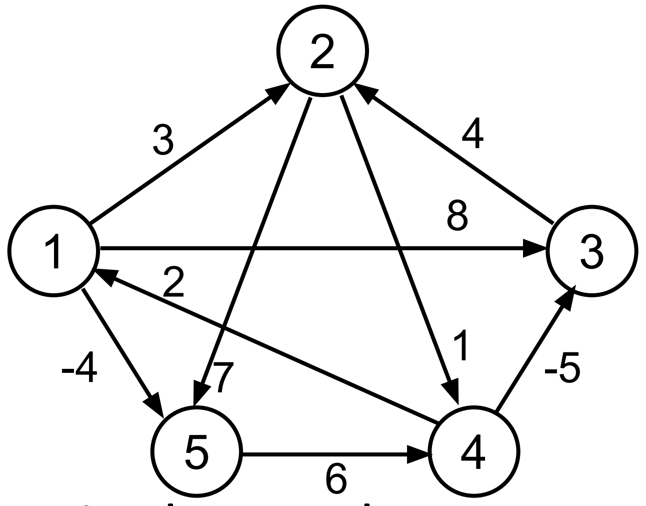


Output the result in an $n \times n$ matrix

$D = (d_{ij})$, where $d_{ij} = \delta(i, j)$

Solve the problem using dynamic programming

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

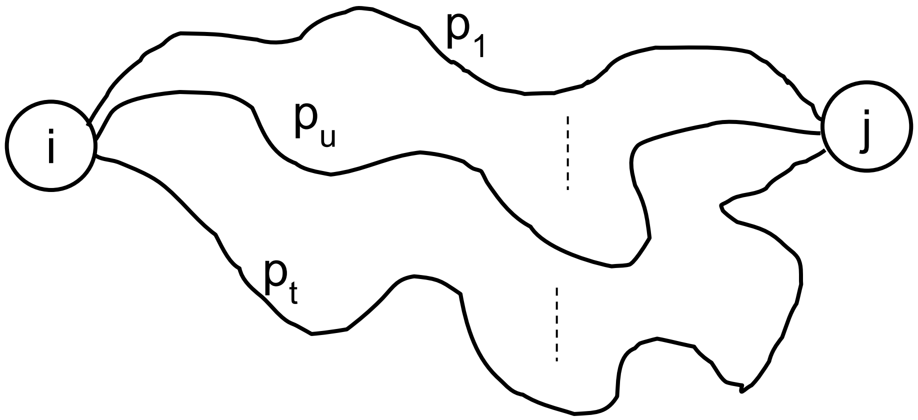
Compute:

The shortest paths between all pairs of vertices in a graph

The Structure of a Shortest Path

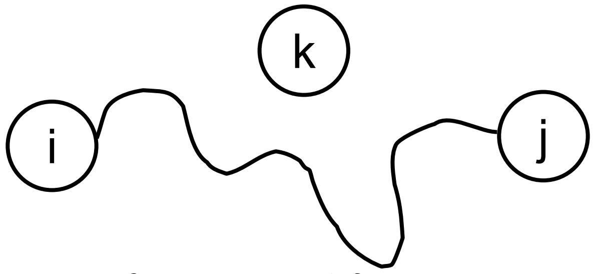
Assume that all vertices are numbered from 1 to V. For any pair of vertices i, j ∈ V, consider all paths from i to j whose intermediate vertices are all drawn from a subset {1, 2, …, k}

Find **p**, a minimum-weight path from these paths



No vertex on these paths has index > k

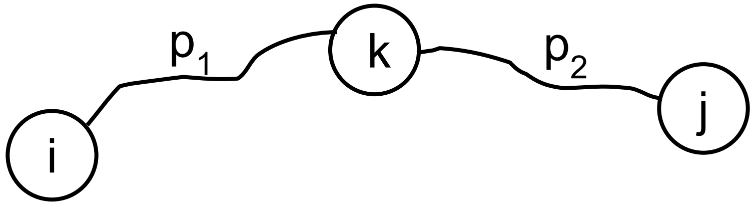
The Structure of a Shortest Path



If k is not an intermediate vertex of path p:

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

If 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 or p_2

So p_1 (respectively, p_2) is a shortest path from i to k (resp., k to j) with vertices from $\{1, 2, \ldots, k - 1\}$

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 $\{1, 2, \ldots, k\}$

For k = 0 (no intermediate vertex is present in SP):

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

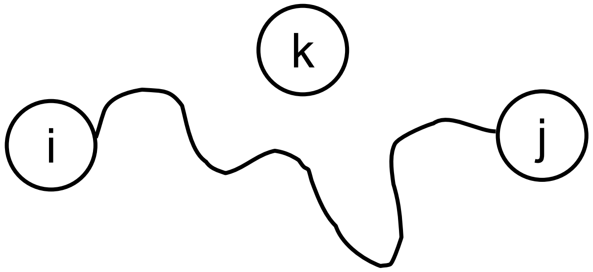
A Recursive Solution (cont.)

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

For k ≥ 1 (One or more intermediate vertex is present in SP):

Case 1: k is not an intermediate vertex of path p

$d_{ij}^{(k)} = d_{ij}^{(k-1)}$



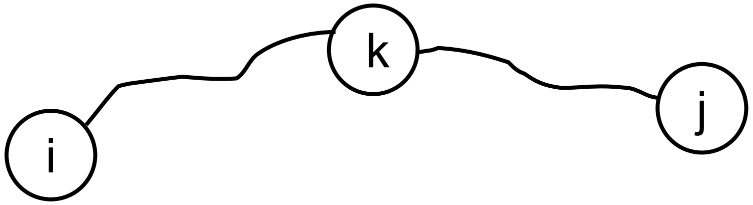
A Recursive Solution (cont.)

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

$k \geq 1$ (One or more intermediate vertex is present in SP):

Case 2: k is an intermediate vertex of path p

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

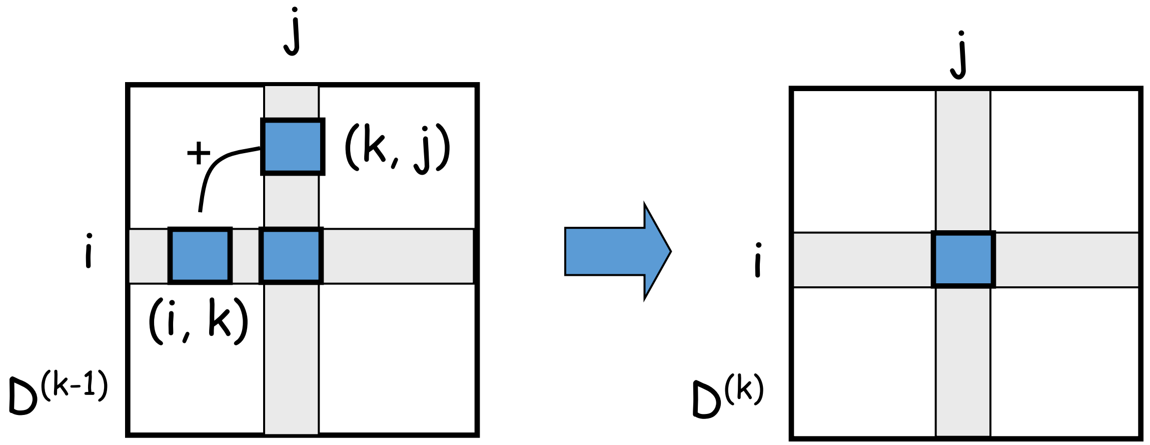


Computing the Shortest Path Weights

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

The final solution: $D^{(n)} = (d_{ij}^{(n)})$:

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



The Floyd-Warshall algorithm

```
Floyd-Warshall(W[1..n][1..n])
D ← W    // $D^{(0)}$
for k ← 1 to n do // compute $D^{(k)}$
    for i ←1 to n do
        for j ←1 to n do
           if D[i][k] + D[k][j] < D[i][j] then
               D[i][j] ← D[i][k] + D[k][j]
return D
```

Running Time: $O(n^3)$

Computing predecessor matrix

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

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

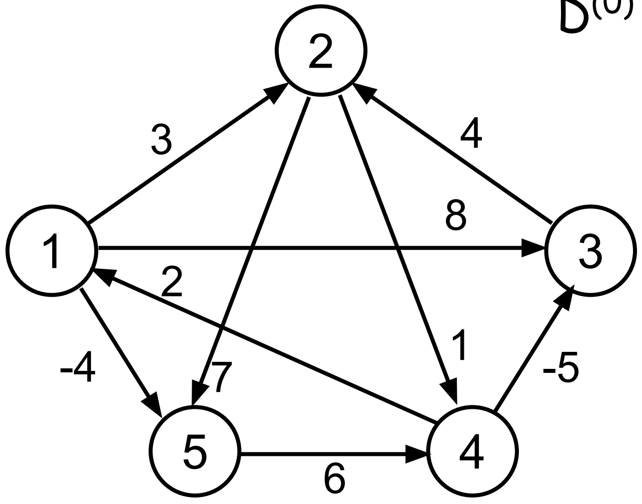
Updating: $p^{(k)}(i,j) = \mathbf{p^{(k-1)}(i,j)}$ *if*$(d^{(k-1)}(i,j) \leq d^{(k-1)}(i,k)+(d^{(k-1)}(k,j)$

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

```
Floyd-Warshall(W[1..n][1..n])
…
for k ← 1 to n do // compute D^(k)
    for i ←1 to n do
        for j ←1 to n do
            if D[i][k] + D[k][j] < D[i][j] then
                D[i][j] ← D[i][k] + D[k][j]
                P[i][j] ← P[k][j]
return D
```

Example

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



$D^{(0)} = W$

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 3 & 8 & \infty & -4 \\ 2 & \infty & 0 & \infty & 1 & 7 \\ 3 & \infty & 4 & 0 & \infty & \infty \\ 4 & 2 & \infty & -5 & 0 & \infty \\ 5 & \infty & \infty & \infty & 6 & 0 \end{array}$$

$D^{(1)}$

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 3 & 8 & \infty & -4 \\ 2 & \infty & & & & \\ 3 & \infty & & & & \\ 4 & 2 & & & & \\ 5 & \infty & & & & \end{array}$$

$i=1,\ k=1 \Rightarrow d_{1j}^{(1)} = \min\{d_{1j}^{(0)},\ d_{11}^{(0)} + d_{1j}^{(0)}\}$

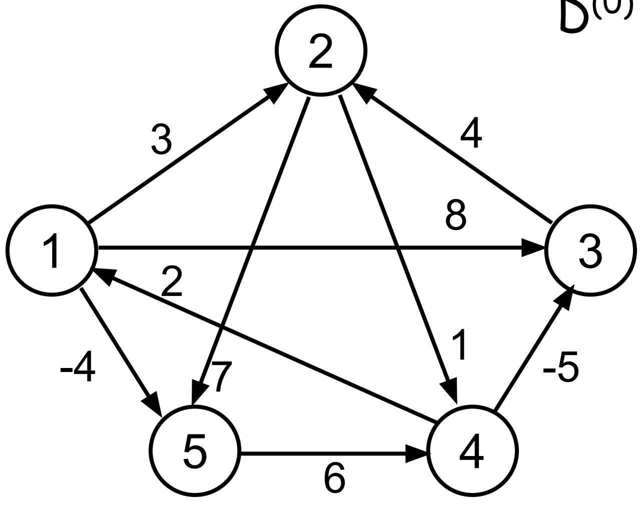
But $d_{11}^{(0)} = 0$; **so** $d_{1j}^{(1)} = d_{1j}^{(0)}$ for all j

Similarly, $d_{i1}^{(1)} = \min\{d_{i1}^{(0)},\ d_{i1}^{(0)} + d_{11}^{(0)}\}$

$= \min\{d_{i1}^{(0)},\ d_{i1}^{(0)} + 0\} = d_{i1}^{(0)}$

Example

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



$D^{(0)} = W$

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 3 & 8 & \infty & -4 \\ 2 & \infty & 0 & \infty & 1 & 7 \\ 3 & \infty & 4 & 0 & \infty & \infty \\ 4 & 2 & \infty & -5 & 0 & \infty \\ 5 & \infty & \infty & \infty & 6 & 0 \end{array}$$

$D^{(1)}$

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 3 & 8 & \infty & -4 \\ 2 & \infty & 0 & \infty & 1 & 7 \\ 3 & \infty & & & & \\ 4 & 2 & & & & \\ 5 & \infty & & & & \end{array}$$

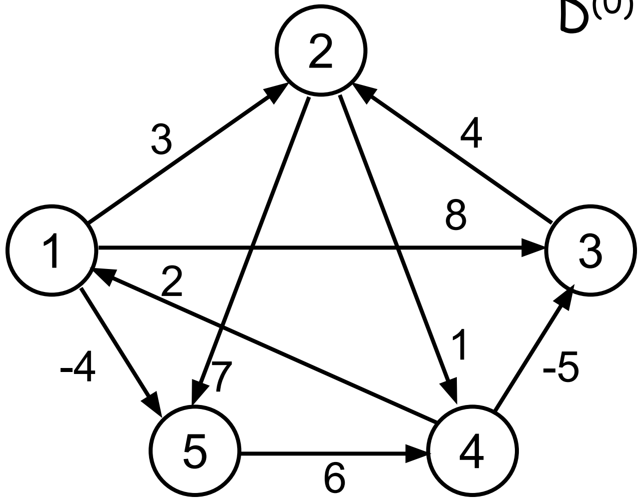
Setting, i=2, k=1:

$$d_{2j}^{(1)} = \min\{d_{2j}^{(0)}, d_{21}^{(0)} + d_{1j}^{(0)}\}$$

But $d_{21}^{(0)} = \infty$; **so** $d_{2j}^{(1)} = d_{2j}^{(0)}$ for all j

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	∞				

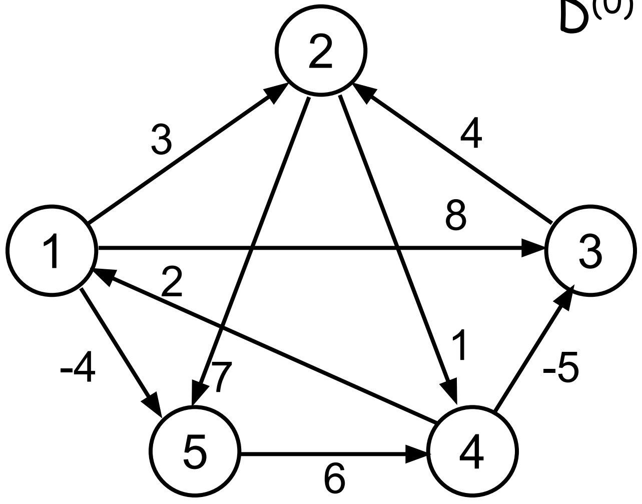
Setting, i=3, k=1:

$$d_{3j}^{(1)} = \min\{d_{3j}^{(0)}, d_{31}^{(0)} + d_{1j}^{(0)}\}$$

But $d_{31}^{(0)} = \infty$; **so** $d_{3j}^{(1)} = d_{3j}^{(0)}$ for all j

Example

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



$D^{(0)} = W$

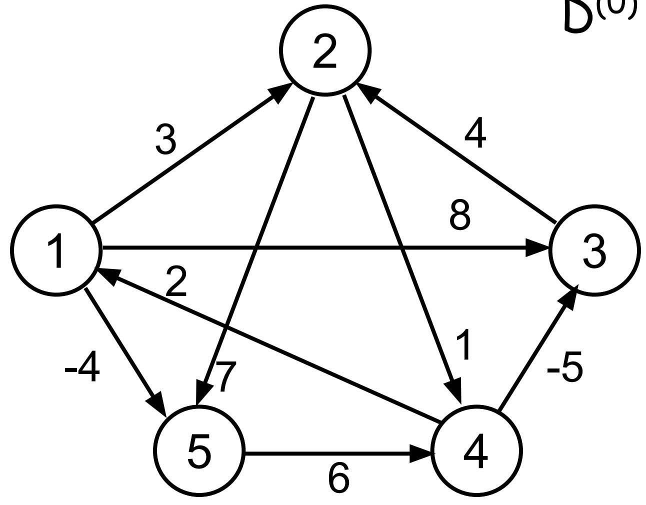
$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 3 & 8 & \infty & -4 \\ 2 & \infty & 0 & \infty & 1 & 7 \\ 3 & \infty & 4 & 0 & \infty & \infty \\ 4 & 2 & \infty & -5 & 0 & \infty \\ 5 & \infty & \infty & \infty & 6 & 0 \end{array}$$

$D^{(1)}$

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 3 & 8 & \infty & -4 \\ 2 & \infty & 0 & \infty & 1 & 7 \\ 3 & \infty & 4 & 0 & \infty & \infty \\ 4 & 2 & & & & \\ 5 & \infty & \infty & \infty & 6 & 0 \end{array}$$

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			0	
5	∞	∞	∞	6	0

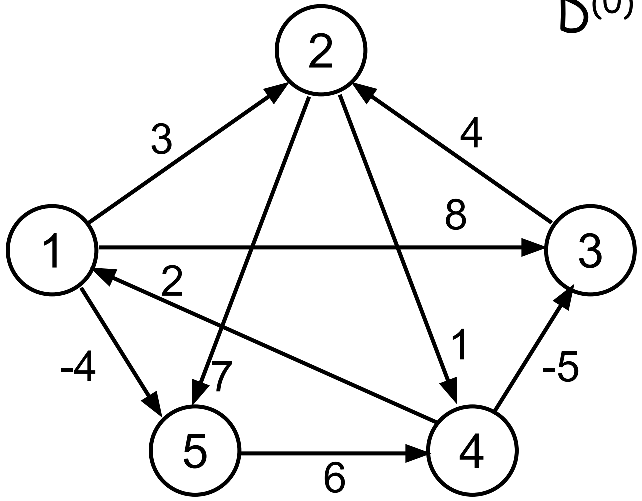
Setting, j=4, k=1:

$d_{i4}^{(1)} = \min\{d_{i4}^{(0)}, d_{i1}^{(0)} + d_{14}^{(0)}\}$

But $d_{14}^{(0)} = \infty$; **so** $d_{i4}^{(1)} = d_{i4}^{(0)}$ for all i

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

$D^{(1)}$

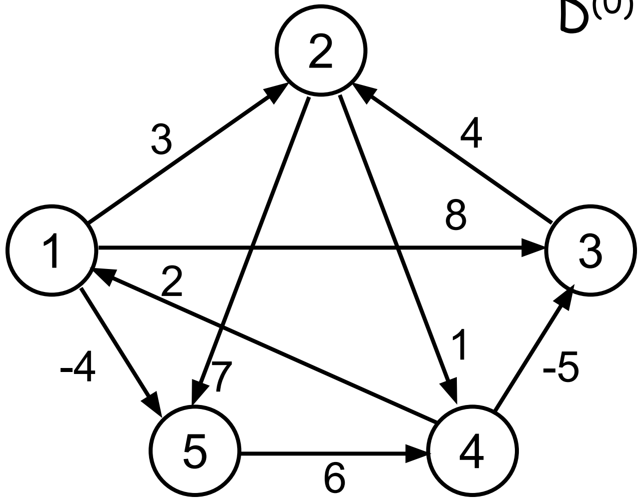
	1	2	3	4	5
1	0	3	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	5	-5	0	-2
5	∞	∞	∞	6	0

$D^{(2)}$

	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

Example

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



$D^{(0)} = W$

$$\begin{array}{c|ccccc}
 & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & 3 & 8 & \infty & -4 \\
2 & \infty & 0 & \infty & 1 & 7 \\
3 & \infty & 4 & 0 & \infty & \infty \\
4 & 2 & \infty & -5 & 0 & \infty \\
5 & \infty & \infty & \infty & 6 & 0
\end{array}$$

$D^{(1)}$

$$\begin{array}{c|ccccc}
 & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & 3 & 8 & \infty & -4 \\
2 & \infty & 0 & \infty & 1 & 7 \\
3 & \infty & 4 & 0 & \infty & \infty \\
4 & 2 & 5 & -5 & 0 & -2 \\
5 & \infty & \infty & \infty & 6 & 0
\end{array}$$

$D^{(2)}$

$$\begin{array}{c|ccccc}
 & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & 3 & 8 & 4 & -4 \\
2 & \infty & 0 & \infty & 1 & 7 \\
3 & \infty & 4 & 0 & 5 & 11 \\
4 & 2 & 5 & -5 & 0 & -2 \\
5 & \infty & \infty & \infty & 6 & 0
\end{array}$$

$D^{(3)}$

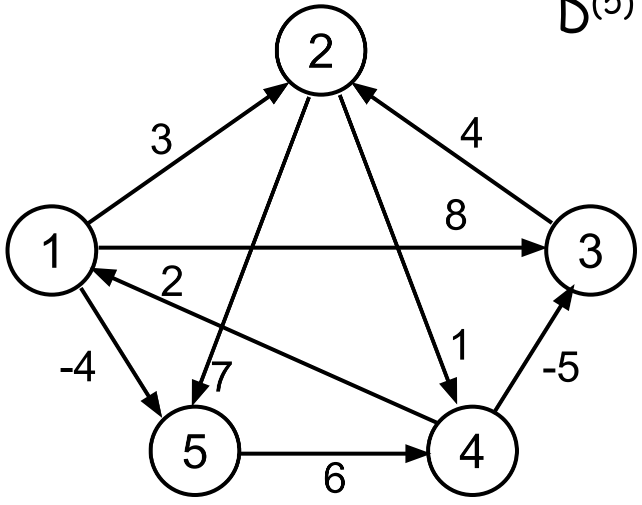
$$\begin{array}{ccccc}
0 & 3 & 8 & 4 & -4 \\
\infty & 0 & \infty & 1 & 7 \\
\infty & 4 & 0 & 5 & 11 \\
2 & -1 & -5 & 0 & -2 \\
\infty & \infty & \infty & 6 & 0
\end{array}$$

$D^{(4)}$

$$\begin{array}{ccccc}
0 & 3 & -1 & 4 & -4 \\
3 & 0 & -4 & 1 & -1 \\
7 & 4 & 0 & 5 & 3 \\
2 & -1 & -5 & 0 & -2 \\
8 & 5 & 1 & 6 & 0
\end{array}$$

Example

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



$D^{(5)}$

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & -3 & 2 & -4 \\ 2 & 3 & 0 & -4 & 1 & -1 \\ 3 & 7 & 4 & 0 & 5 & 3 \\ 4 & 2 & -1 & -5 & 0 & -2 \\ 5 & 8 & 5 & 1 & 6 & 0 \end{array}$$

$P^{(5)}$

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & - & 3 & 4 & 5 & 1 \\ 2 & 4 & - & 4 & 2 & 1 \\ 3 & 4 & 3 & - & 2 & 1 \\ 4 & 4 & 3 & 4 & - & 1 \\ 5 & 4 & 3 & 4 & 5 & - \end{array}$$

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 of Directed Un-weighted 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 j ←1 to n do
            T[i][j] ← T[i][j] ∨ (T[i][k] ∧ T[k][j])
return T
```

Johnson's Algorithm for computing APSP

Makes clever use of Bellman-Ford and Dijkstra to do All-Pairs-Shortest-Paths efficiently on sparse graphs.

Motivation: By running Dijkstra $|V|$ times, we could do APSP in time $\Theta(VE\lg V)$ (Modified Dijkstra), or $\Theta(V^2\lg V + VE)$ (Fibonacci Dijkstra). This beats $\Theta(V^3)$ (Floyd-Warshall) when the graph is sparse.

Problem: Dijkstra doesn't work when negative edge weights are present

The Basic Idea

Reweight the edges so that:

1. No edge weight is negative.
2. Shortest paths are preserved. (A shortest path in the original graph is still one in the new, reweighted graph.)

An obvious attempt: subtract the minimum weight from all the edge weights. E.g. if the minimum weight is -2:

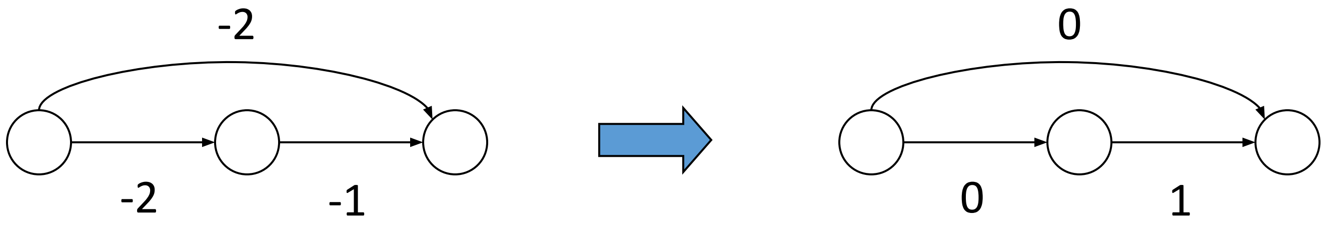
-2 - -2 = 0
3 - -2 = 5

etc.

Counterexample

Subtracting the minimum weight from every weight doesn't work.

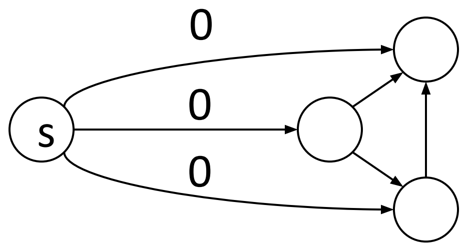
Consider:



Paths with more edges are unfairly penalized.

Johnson's Insight

Add a vertex *s* to the original graph G, with edges of weight 0 to each vertex in G:



Assign new weights ŵ to each edge as follows:

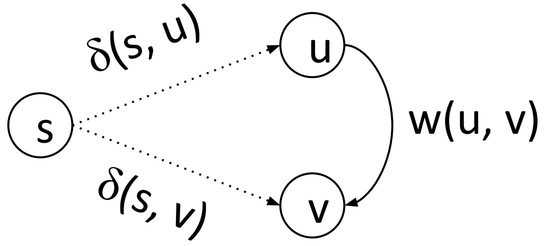
$$\hat{w}(u, v) = w(u, v) + \delta(s, u) - \delta(s, v)$$

Question 1

Are all the ŵ's non-negative? Yes:

$\delta(s,u) + w(u,v)$ must be $\geq \delta(s,v)$

Otherwise, s $\Rightarrow$ u $\rightarrow$ v would be shorter than the shortest path from s to v. So,



$$\delta(s,u) + w(u,v) \geq \delta(s,v)$$

Rewriting :

$$\underbrace{w(u,v) + \delta(s,u) - \delta(s,v)}_{\hat{w}(u,v)} \geq 0$$

Question 2

Does the reweighting preserve shortest paths? Yes: Consider any path $p = v_1, v_2, \ldots, v_k$

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

The sum telescopes

$$\left\{\begin{aligned} &= w(v_1, v_2) + \delta(s, v_1) - \cancel{\delta}(s, v_2) \\ &+ w(v_2, v_3) \qquad + \cancel{\delta}(s, v_2) - \delta(s, v_3) \\ &\qquad \cdots\cdots\cdots\cdots \\ &+ w(v_{k-1}, v_k) \qquad + \cancel{\delta}(s, v_{k-1}) - \delta(s, v_k) \end{aligned}\right.$$

$$= w(p) + \underbrace{\delta(s, v_1) - \delta(s, v_k)}_{\text{A value that depends only on the endpoints, not on the path.}}$$

In other words, we have adjusted the lengths of all paths by the same amount. So this will not affect the relative ordering of the paths— i.e., shortest paths will be preserved.

Question 3

How do we compute the $\delta(s, v)$'s?

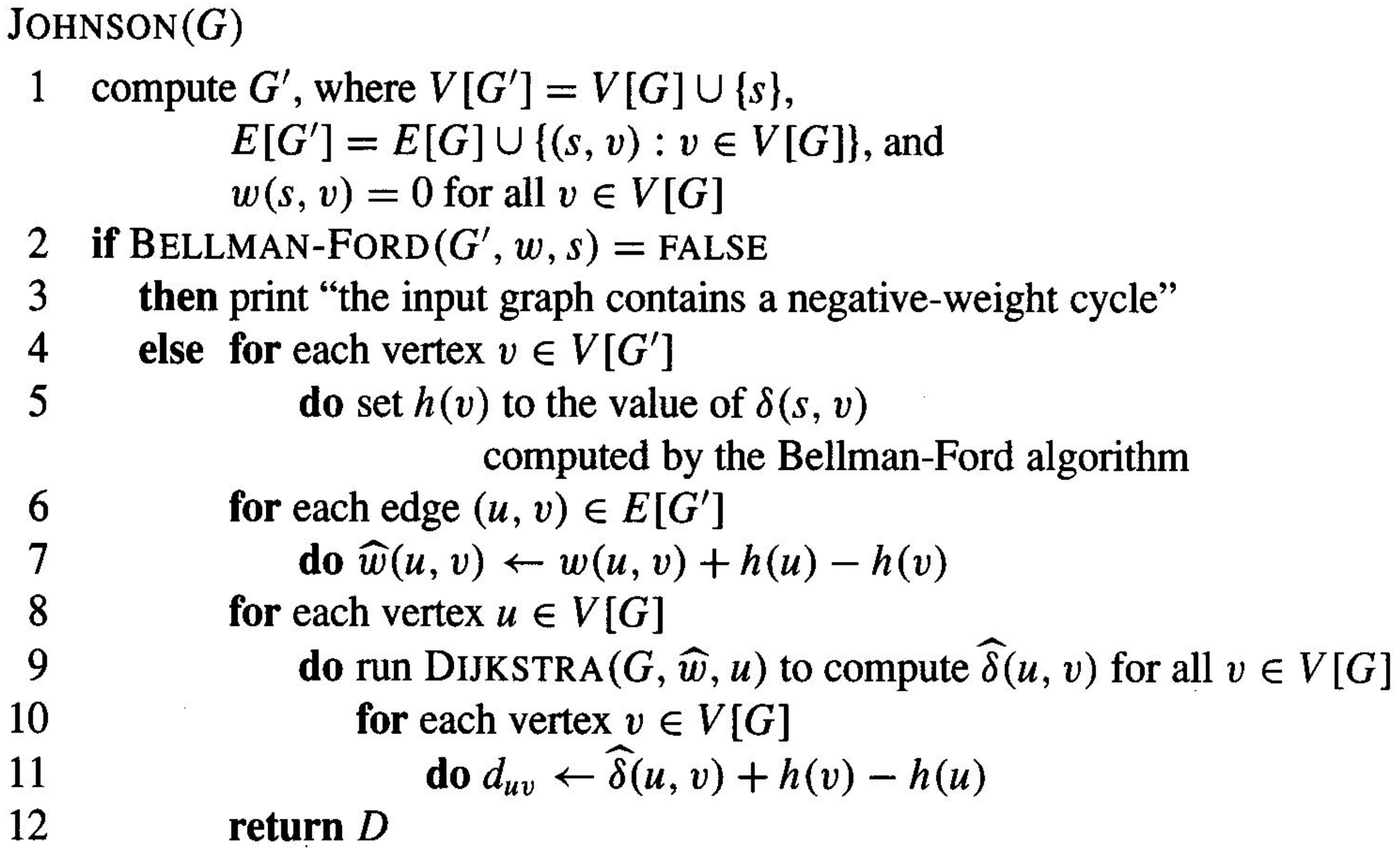
Use Bellman-Ford.

This also tells us if we have a negative-weight cycle.

Johnson’s: Algorithm

1. Compute G’, which consists of G augmented with s and a zero-weight edge from s to every vertex in G.
2. Run Bellman-Ford(G’, w, s) to obtain the $\delta(s,v)$’s
3. Reweight by computing $\hat{w}$ for each edge
4. Run Dijkstra on each vertex to compute $\hat{\delta}$
5. Undo reweighting factors to compute δ

Johnson's: CLRS



```
JOHNSON($G$)
1  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 $v \in V[G]$
2  if BELLMAN-FORD($G', w, s$) = FALSE
3     then print "the input graph contains a negative-weight cycle"
4     else for each vertex $v \in V[G']$
5              do set $h(v)$ to the value of $\delta(s, v)$
                      computed by the Bellman-Ford algorithm
6          for each edge $(u, v) \in E[G']$
7              do $\widehat{w}(u, v) \leftarrow w(u, v) + h(u) - h(v)$
8          for each vertex $u \in V[G]$
9              do run DIJKSTRA($G, \widehat{w}, u$) to compute $\widehat{\delta}(u, v)$ for all $v \in V[G]$
10                for each vertex $v \in V[G]$
11                    do $d_{uv} \leftarrow \widehat{\delta}(u, v) + h(v) - h(u)$
12         return $D$
```

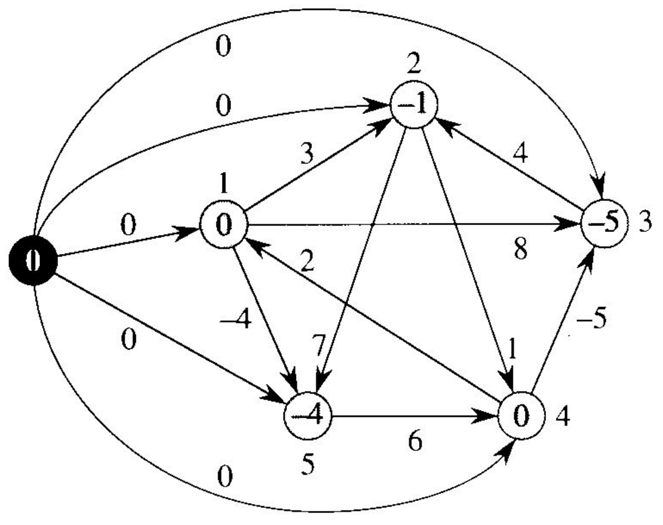
Johnson's: Running Time

1. Computing G': $\Theta(V)$
2. Bellman-Ford: $\Theta(VE)$
3. Reweighting: $\Theta(E)$
4. Running (Modified) Dijkstra: $\Theta(VE\lg V)$
5. Adjusting distances: $\Theta(V^2)$

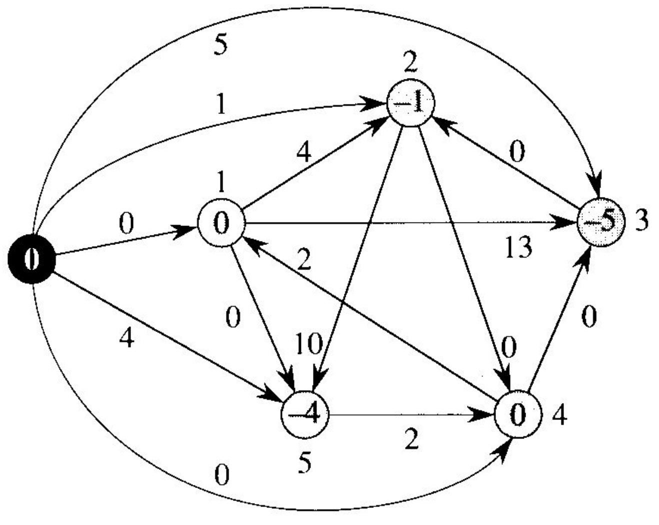
———————————————————————

Total Time: $\Theta(V^2 + VE\lg V)$

Johnson: reweighting



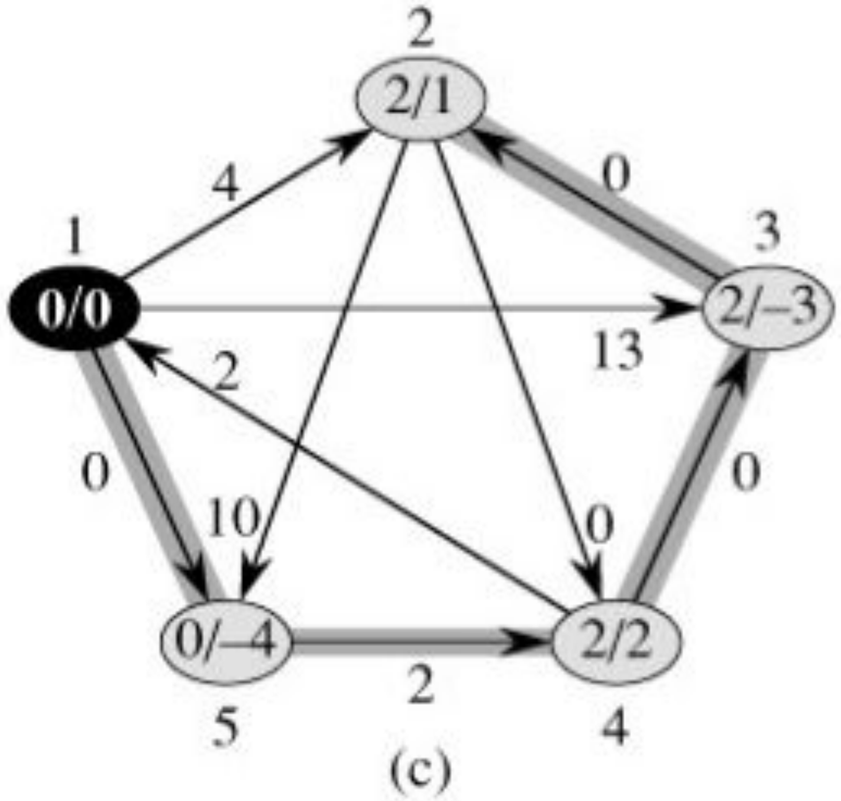
(a)



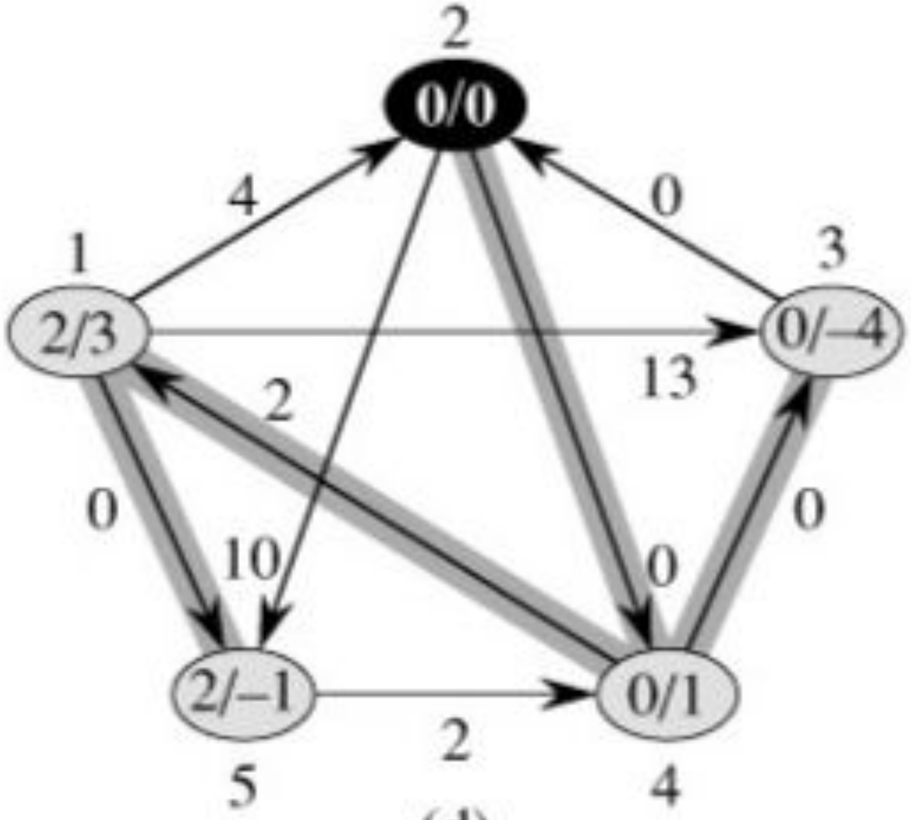
(b)

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

Johnson using Dijkstra

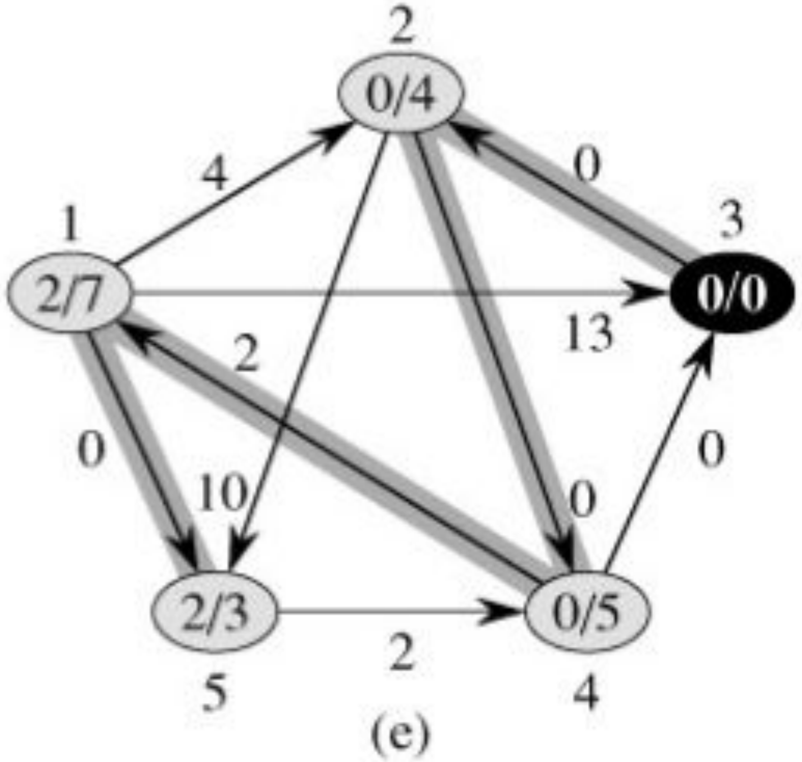


(c)

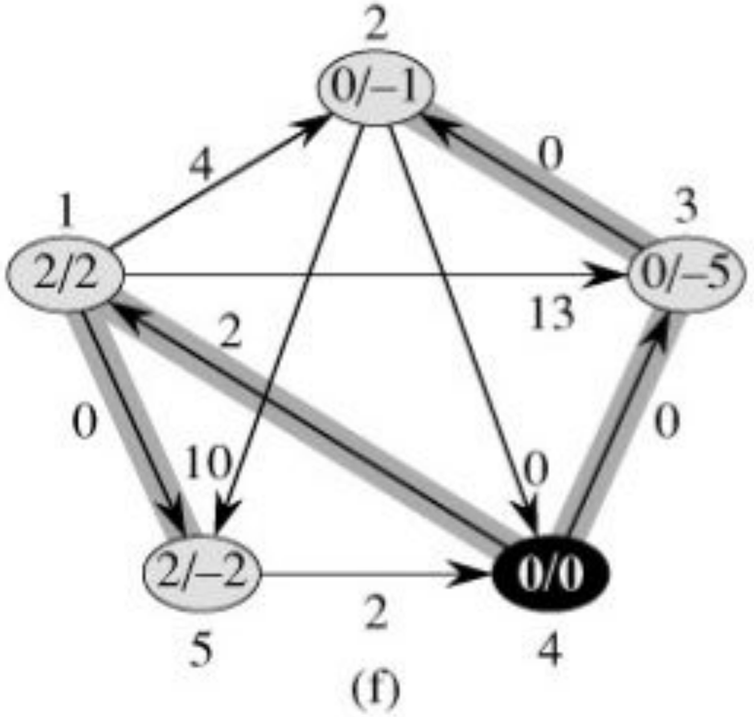


(d)

Johnson using Dijkstra

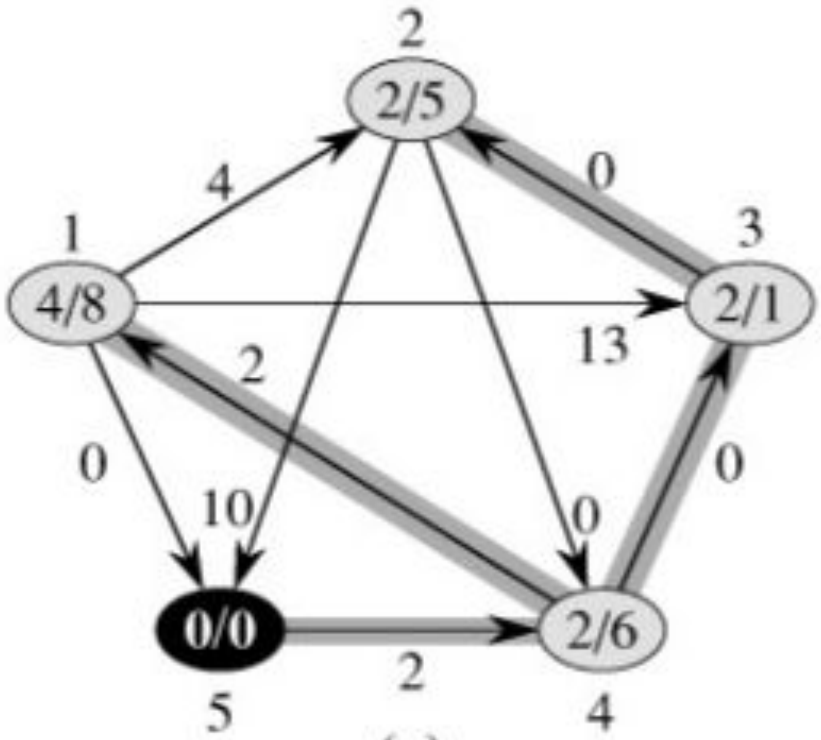


(e)



(f)

Johnson using Dijkstra



(g)