All-Pairs Shortest Paths

Introduction

In this Lecture, we consider the problem of finding shortest paths between all pairs of vertices in a graph.

This problem might arise in making a table of distances between all pairs of cities for a road atlas.

As previously, we are given a weighted, directed graph

$G = (V, E)$ with a weight function $w : E \rightarrow \mathbb{R}$ that maps edges to real-valued weights.

We wish to find, for every pair of vertices $u, v \in V$, a shortest (least-weight) path from u to v, where the weight of a path is the sum of the weights of its constituent edges.

We typically want the output in tabular form: the entry in u's row and v's column should be the weight of a shortest path from u to v.

We can solve an all-pairs shortest-paths problem by running a single-source shortest-paths algorithm $|V|$ times, once for each vertex as the source.

If all edge weights are nonnegative, we can use Dijkstra's algorithm.

The rough estimation of the running time is than $O(V^3)$.

If the graph has negative-weight edges, we cannot use Dijkstra's algorithm.

Instead, we must run the slower Bellman-Ford algorithm once from each vertex.

The resulting running time is $O(V^2E)$, which on a dense graph is $O(V^4)$.

We shall see how to do better.

Unlike the single-source algorithms, which assume an adjacency-list representation of the graph, most of the algorithms in this Lecturer use an adjacency-matrix representation.

(Johnson's algorithm for sparse graphs uses adjacency lists.)

We assume that the vertices are numbered

$1, 2, \ldots, |V|$.

Input: an n x n matrix W representing the edge weights of an n-vertex directed graph $G = (V, E)$.

That is, $W = \{w_{ij}\}$, where

$$w_{ij} = \begin{cases} 0 & \text{if } i = j\,; \\ \text{the weight of directed edge } (i, j) & \text{if } i \neq j \text{ and } (i, j) \in E\,; \\ \propto & \text{if } i \neq j \text{ and } (i, j) \notin E\,: \end{cases} \qquad (1)$$

We allow negative-weight edges, but we assume that the input graph contains no negative-weight cycles.

Output: an n x n matrix $D = (d_{ij})$, where entry d_{ij} contains the weight of a shortest path from vertex i to vertex j .

That is, if δ(i, j) denote the shortest- path weight from vertex i to vertex j, then $d_{ij} = \delta(i, j)$ at termination.

Predecessor matrix

To solve the all-pairs shortest-paths problem on an input adjacency matrix, we need to compute not only the shortest-path weights but also a predecessor matrix Π, *where* $\pi_{ij} = NIL$ if

either $i = j$

or there is no path from i to j, and

otherwise π_{ij} is the predecessor of j on some shortest path from i .

Just as the predecessor subgraph G_π is a shortest-paths tree for a given source vertex, the subgraph induced by the i-th row of the Π matrix should be a shortest-paths tree with root i.

Predecessor subgraph

For each vertex $i \in V$, we define the predecessor subgraph of G for i as

$G_{\pi,i} = (V_{\pi,i}, E_{\pi,i})$,

where

$V_{\pi,i} = \{j \in V\colon \pi_{ij} \neq \text{NIL}\} \cup \{i\}$

and

$E_{\pi,i} = \{(\pi_{ij}, j)\colon j \in V_{\pi,i} - \{i\}\}$.

Some conventions:

1) The input graph $G = (V, E)$ has n vertices, so that $n = |V|$.
2) We shall use the convention of denoting matrices by uppercase letters, such as W, L, or D, and their individual elements by subscripted lowercase letters, such as w_{ij}, l_{ij} , or d_{ij} .
3)Some matrices will have parenthesized superscripts, as in $L^{(m)} = l_{ij}^{(m)}$ or $D^{(m)} = d_{ij}^{(m)}$, to indicate iterates.
4) For a given n x n matrix A, we shall assume that the value of n is stored in the attribute *A.rows.*

Shortest paths and matrix multiplication

First, we consider a dynamic-programming algorithm for the all-pairs shortest-paths problem on a directed graph $G = (V, E)$.

Each major loop of the dynamic program will invoke an operation that is very similar to matrix multiplication, so that the algorithm will look like repeated matrix multiplication.

We shall start by developing a $\theta(V^4)$-time algorithm for the all-pairs shortest-paths problem and then improve its running time to $(V^3 \lg V)$.

The structure of a shortest path

We start by characterizing the structure of an optimal solution.

For the single-source shortest-paths problem on a graph $G = (V, E)$, we have proven (Lemma 1 from previous Lecture) that all subpaths of a shortest path are shortest paths.

Suppose that we represent the graph by an adjacency matrix

$W = (w_{ij})$.

Consider a shortest path p from vertex i to vertex j, and suppose that p contains $\leq m$ edges.

Assuming that there are no negative-weight cycles, m is finite.

If $i = j$, then p has weight 0 and no edges.

If vertices i and j are distinct, then we decompose path p into

$i \overset{p'}{\sim}> k \to j$, where path p' now contains at most m-1 edges.

By Lemma 1, p' is a shortest path from i to k, and so

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

A recursive solution to the all-pairs shortest-paths problem

Now, let $l_{ij}^{(m)}$ be the minimum weight of any path from vertex i to vertex j that contains $\leq m$ edges.

When $m = 0$, there is a shortest path from i to j with no edges $\Leftrightarrow$ $i = j$.

Thus,

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

For $m \geq 1$, we compute $l_{ij}^{(m)}$ as the minimum of $l_{ij}^{(m\text{-}1)}$ (the weight of a shortest path from i to j consisting of at most m-1 edges) and the minimum weight of any path from i to j consisting of at most m edges, obtained by looking at all possible predecessors k of j .

Thus, we recursively define

$$\begin{aligned} l_{ij}^{(m)} &= \min\left(l_{ij}^{(m-1)}, \min_{1 \leq k \leq n}\left\{l_{ik}^{(m-1)} + w_{kj}\right\}\right) \\ &= \min_{1 \leq k \leq n}\left\{l_{ik}^{(m-1)} + w_{kj}\right\} . \end{aligned} \tag{2}$$

The latter equality follows since $w_{jj} = 0$ for all j .

If the graph contains no negative-weight cycles, then for every pair of vertices i and j for which $\delta(i, j) < \propto$, there is a shortest path from i to j that is simple and thus contains at most n-1 edges.

A path from vertex i to vertex j with > n -1 edges cannot have lower weight than a shortest path from i to j .

The actual shortest-path weights are therefore given by

$$\delta(i, j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n+1)} = \cdots . \qquad (3)$$

Computing the shortest-path weights bottom up

Taking as our input the matrix $W = (w_{ij})$, we now compute a series of matrices

$L^{(1)}, L^{(2)}, \ldots, L^{(n-1)}$, where for $m = 1, 2, \ldots, n - 1$, we have

$L^{(m)} = (l_{ij}^{(m)})$.

The final matrix $L^{(n-1)}$ contains the actual shortest-path weights.

Observe that $l_{ij}^{(1)} = w_{ij}$ for all vertices $i, j \in V$, and so $L^{(1)} = W$.

The heart of the algorithm is the following procedure, which, given matrices $L^{(m-1)}$ *and* W, returns the matrix $L^{(m)}$

That is, it extends the shortest paths computed so far by one more edge.

```
EXTEND-SHORTEST-PATHS(L, W)
n = L.rows
let L' = (l_ij') be a new n x 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'
```

The procedure computes a matrix $L' = (l_{ij}')$, which it returns at the end.

It does so by computing equation (2) for all i and j , using L for $L^{(m-1)}$ and L' for $L^{(m)}$.

(It is written without the superscripts to make its input and output matrices independent of m.)

Its running time is $\Theta(n^3)$ due to the 3 nested **for** loops.

Now we can see the relation to matrix multiplication.

Suppose we wish to compute the matrix product $C = A \cdot B$ of two n x n matrices A and B.

Then, for $i, j = 1, 2, \ldots, n$, we compute

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj} \ . \qquad (4)$$

SQUARE-MATRIX-MULTIPLY(A, B)

$n = A.rows$
let C be a new n x n matrix
for $i = 1$ **to** n
    for $j = 1$ **to** n
        $c_{ij} = 0$
        for $k = 1$ **to** n
            $c_{ij} = c_{ij} + a_{ik} \times b_{kj}$
return C

Observe that if we make the substitutions

$l^{(m-1)}$ -> a

w -> b ,

$l^{(m)}$ -> c

min -> + ,

+ -> $\cdot$

in equation (2), we obtain equation (4).

Thus, if we make these changes to EXTEND-SHORTEST-PATHS and also replace $\propto$ (the identity for min) by 0 (the identity for +), we obtain the same $\Theta(n^3)$-time procedure for multiplying square matrices.

Returning to the all-pairs shortest-paths problem, we compute the shortest-path weights by extending shortest paths edge by edge.

Letting $A \cdot B$ denote the matrix "product" returned by EXTEND-SHORTEST-PATHS(A, B), we compute the sequence of n -1 matrices

$L^{(1)} = L^{(0)} \cdot W = W,$

$L^{(2)} = L^{(1)} \cdot W = W^2$

$L^{(3)} = L^{(2)} \cdot W = W^3$;

:

:

:

$L^{(n-1)} = L^{(n-2)} \cdot W = W^{n-1}$:

The matrix $L^{(n-1)} = W^{n-1}$ contains the shortest-path weights.

The following procedure computes this sequence in $\Theta(n^4)$ time.

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

$n = W$.rows

$L^{(1)} = W$

for $m = 2$ **to** $n - 1$

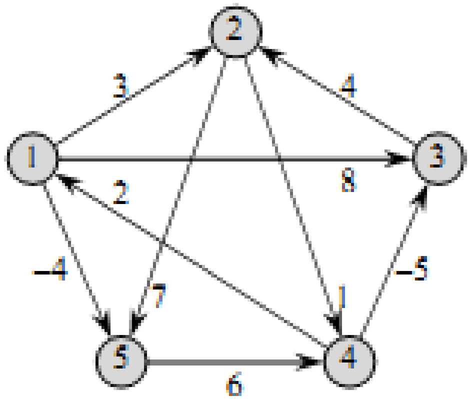
    let $L^{(m)}$ be a new $n \times n$ matrix

    $L^{(m)}$ = EXTEND-SHORTEST-PATHS($L^{(m-1)}$, W)

return $L^{(n-1)}$

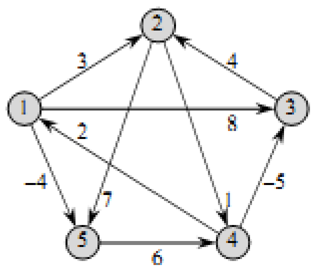
Slides 21-24 show a graph and the matrices $L^{(m)}$ computed by the procedure SLOW-ALL-PAIRS-SHORTEST-PATHS.

$$L^{(1)} = L^{(0)} \cdot W = W$$



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

$$L^{(2)} = L^{(1)} \cdot W = W^2$$



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

$l_{11}^{(2)} = \min(0+0, 3+\propto, 8+\propto, \propto+2, -4+\propto) = 0$

$l_{12}^{(2)} = \min(0+3, 3+0, 8+4, \propto+\propto, -4+\propto) = 3$

$l_{13}^{(2)} = \min(0+8, 3+\propto, 8+0, \propto -5, -4+\propto) = 8$

$l_{14}^{(2)} = \min(0+2, 3+1, 8+5, \propto+0, -4+6) = 2$

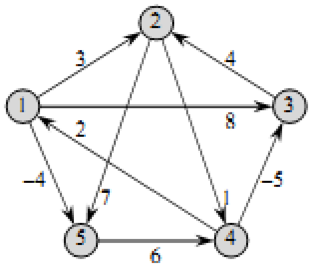
$l_{15}^{(2)} = \min(0-4, 3+7, 8+11, \propto-2, -4+0) = -4$

…

…

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

$L^{(3)} = L^{(2)} \cdot W = W^3$



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

$l_{11}^{(3)}$ = min (0+0, 3+$\propto$, 8+$\propto$, 2+2, -4+$\propto$) = 0

$l_{12}^{(3)}$ = min (0+3, 3+0, 8+4, 2+$\propto$, -4+$\propto$) = 3

$l_{13}^{(3)}$ = min(0+8, 3+$\propto$, 8+0, 2 -5, -4+$\propto$) = -3

$l_{14}^{(3)}$ = min (0+$\propto$, 3+1, 8+$\propto$, 2+0, -4+6) = 2

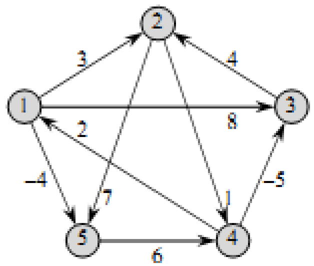
$l_{15}^{(3)}$ = min (0-4, 3+7, 8+$\propto$, 2+$\propto$, -4+0) = -4

…

…

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

$$L^{(4)} = L^{(3)} \cdot W = W^4$$



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

$l_{11}^{(4)}$ = min (0+0, 3+$\propto$, -3+$\propto$, 2+2, -4+$\propto$) = 0
$l_{12}^{(4)}$ = min (0+3, 3+0, -3+4, 2+$\propto$, -4+$\propto$) = 1
$l_{13}^{(4)}$ = min(0+8, 3+$\propto$, -3+0, 2 -5, -4+$\propto$) = -3
$l_{14}^{(4)}$ = min (0+$\propto$, 3+1, -3+$\propto$, 2+0, -4+6) = 2
$l_{15}^{(4)}$ = min (0-4, 3+7, -3+$\propto$, 2+$\propto$, -4+0) = -4
…

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

Improving the running time

Our goal, however, is not to compute *all* the $L^{(m)}$ matrices: we are interested only in matrix $L^{(n-1)}$.

Recall that in the absence of negative-weight cycles, equation (3) implies

$L^{(m)} = L^{(n-1)}$ for all integers $m \geq n - 1$.

Just as traditional matrix multiplication is associative, so is matrix multiplication defined by the EXTEND-SHORTEST-PATHS procedure.

Therefore, we can compute $L^{(n-1)}$ with only $\lceil \lg(n-1) \rceil$ matrix products by computing the sequence

$L(^{1)} = W;$

$L^{(2)} = W^2 = W \cdot W;$

$L^{(4)} = W^4 = W^2 \cdot W^2;$

$L^{(8)} = W^8 = W^4 \cdot W^4 ;$

:

:

:

$L^{(2\lceil \lg(n-1 \rceil)} = \mathrm{W}^{2\lceil \lg(n-1) \rceil} = W^{2\lceil \lg(n-1) \rceil -1} \cdot W^{2\lceil \lg(n-1) \rceil -1}.$

Since $2^{\lceil \lg(n-1) \rceil} \geq n -1$, the final product $L^{(2\lceil \lg(n-1) \rceil)}$ is equal to $L^{(n-1)}$.

The following procedure computes the above sequence of matrices by using this technique of repeated squaring

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 x n matrix

    $L^{(2m)}$ = EXTEND-SHORTEST-PATHS($L^{(m)}$, $L^{(m)}$)

    $m = 2m$

return $L^{(m)}$

In each iteration of the **while** loop of lines 4–7, we compute $L^{(2m)} = (L^{(m)})^{2,}$ starting with $m = 1$.

At the end of each iteration, we double the value of m.

The final iteration computes $L^{(n-1)}$ by actually computing $L^{(2m)}$ for some $n - 1 \leq 2m < 2n - 2$.

By equation (3), $L^{(2m)} = L^{(n-1).}$

The next time the test in line 4 is performed, m has been doubled,

so now $m \geq n - 1$, the test fails, and the procedure returns the last matrix it computed.

Because each of the $\lceil \lg(n - 1) \rceil$ matrix products takes $\Theta(n^3)$ time,

FASTER-ALL-PAIRS-SHORTEST-PATHS runs in $\Theta(n^3 \lg n)$ time.

Observe that the code is tight, containing no elaborate data structures, and the constant hidden in the Θ-notation is therefore small.

The Floyd-Warshall algorithm

Next, we shall use a different dynamic-programming formulation to solve the all-pairs shortest-paths problem on a directed graph $G = (V, E)$.

The resulting algorithm, known as the Floyd-Warshall algorithm, runs in $\Theta(V^3)$ time.

As before, negative-weight edges may be present, but we assume that there are no negative-weight cycles.

We follow the dynamic-programming process to develop the algorithm.

The structure of a shortest path

The Floyd-Warshall algorithm considers the intermediate vertices of a shortest path, where an intermediate vertex of a simple path $p = \langle v_1, v_2, \ldots, v_l \rangle$ is any vertex of p other than v_1 or v_l, that is, any vertex in the set $\{v_2, v_3, \ldots, v_{l-1}\}$.

The Floyd-Warshall algorithm relies on the following observation.

Under our assumption that the vertices of G are $V = \{1, 2, \ldots, n\}$, let us consider a subset $\{1, 2, \ldots, k\}$ of vertices for some k.

For any pair of vertices $i, j \in V$, consider all paths from i to j whose intermediate vertices are all drawn from $\{1, 2, \ldots, k\}$, and let p be a minimum-weight path from among them. (Path p is simple.)

The Floyd-Warshall algorithm exploits a relationship between path p and shortest paths from i to j with all intermediate vertices in the set $\{1, 2, \ldots, k - 1\}$.

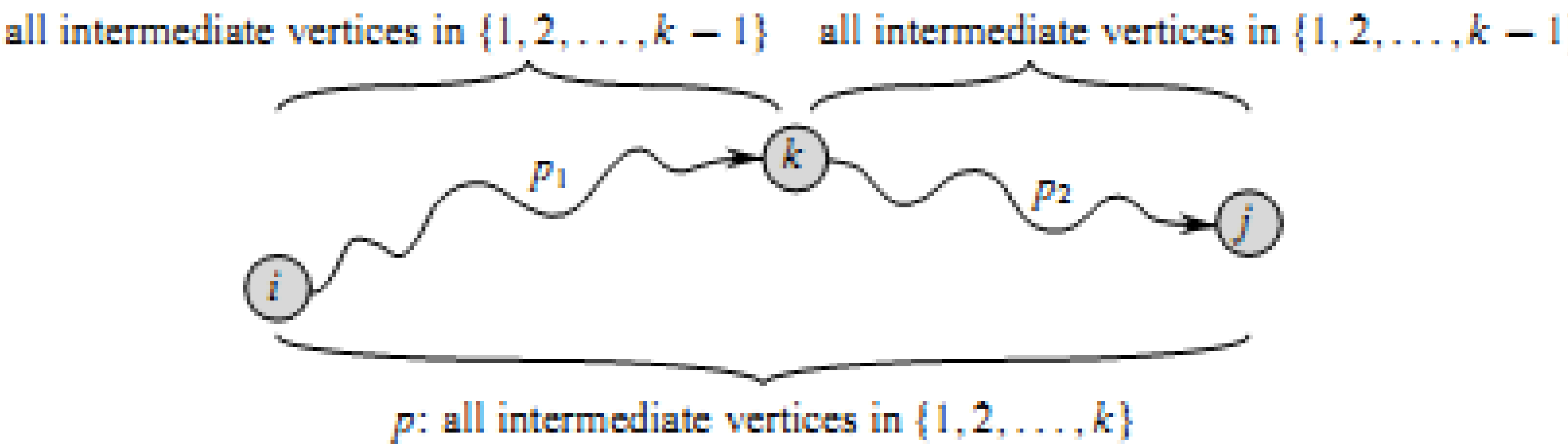
The relationship depends on whether or not k is an intermediate vertex of path p.

If k is not an intermediate vertex of path p, then all intermediate vertices of path p are in the set

$\{1, 2, ..., k - 1\}$.

Thus, a shortest path from vertex i to vertex j with all intermediate vertices in the set $\{1, 2, ..., k - 1\}$ is also a shortest path from i to j with all intermediate vertices in the set $\{1, 2, ..., k\}$.

If k is an intermediate vertex of path p, then we decompose p into $i^{(p1)}\sim> k^{(p2)}\sim> j$ as next slide demonstrates .



Path p is a shortest path from vertex i to vertex j , and k is the highest-numbered intermediate vertex of p.
Path p_1, the portion of path p from vertex i to vertex k, has all intermediate vertices in the set {1, 2, …, k - 1}.
The same holds for path p_2 from vertex k to vertex j .

By Lemma 1 from previous Lecture, p_1 is a shortest path from *i* to *k* with all intermediate vertices in the set

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

In fact, we can make a slightly stronger statement.

Because vertex *k* is not an intermediate vertex of path p_1, all intermediate vertices of p_1 are in the set $\{1, 2, \ldots, k - 1\}$.

Therefore, p_1 is a shortest path from *i* to *k* with all intermediate vertices in the set

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

Similarly, p_2 is a shortest path from vertex *k* to vertex *j* with all intermediate vertices in the set $\{1, 2 \ldots, k - 1\}$.

A recursive solution to the all-pairs shortest-paths problem

Based on the above observations, we define one more recursive formulation of shortest-path estimates.

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*}.

When *k* = 0, a path from vertex *i* to vertex *j* with no intermediate vertex numbered higher than 0 has no intermediate vertices at all.

Such a path has at most one edge, and hence $d_{ij}^{(0)} = w_{ij}$.

Following the above discussion, we define $d_{ij}^{(k)}$ recursively by

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0\,, \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 1\,. \end{cases} \qquad (5)$$

Because for any path, all intermediate vertices are in the set $\{1, 2, \ldots, n\}$, the matrix $D^{(n)} = d_{ij}^{(n)}$ gives the final answer: $d_{ij}^{(n)} = \delta(i, j)$ for all $i, j \in V$.

Computing the shortest-path weights bottom up

Based on recurrence (5), we can use the following bottom-up procedure to compute the values $d_{ij}^{(k)}$ in order of increasing values of k.

Input: an n x n matrix W defined as in equation (1).

Output: the matrix $D^{(n)}$ of shortest-path weights.

FLOYD-WARSHALL(W)

$n = W.rows$

$D^{(0)} = W$

for $k = 1$ **to** n

    let $D^{(k)} = d_{ij}^{(k)}$ be a new n x n matrix

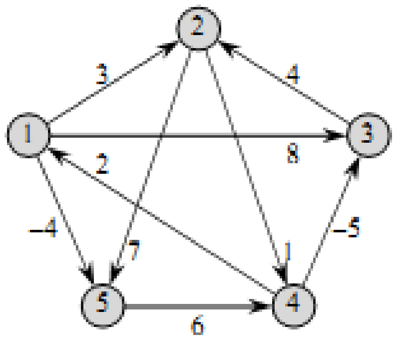
    for $i = 1$ **to** n

        for $j = 1$ **to** n

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

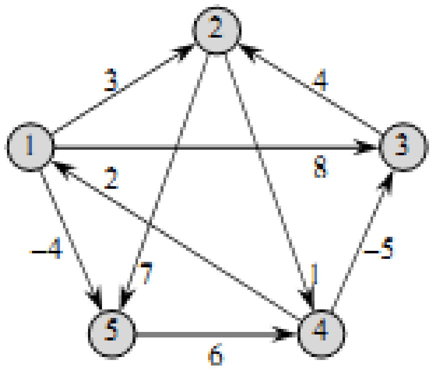
return $D^{(n)}$

The sequence of matrices $D^{(k)}$ computed by the Floyd-Warshall algorithm for the graph shown bellow.



$$D^{(0)} = W$$

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



Intermediate vertices = {1}

All the nodes to enter node 1 are shown in column 1.

All the nodes to leave node 1 are shown in row 1.

$d_{41}^{(1)} = \min(d_{41}^{(0)} + d_{11}^{(0)}, d_{41}^{(0)})$ = min(0+2, 2) = 2

$d_{42}^{(1)} = \min(d_{41}^{(0)} + d_{12}^{(0)}, d_{42}^{(0)})$ = min(3+2, ∞) => d_{42}= 5, π_{42} = 1

$d_{43}^{(1)} = \min(d_{41}^{(0)} + d_{13}^{(0)}, d_{43}^{(0)})$ = min(8+2, -5) = -5

$d_{44}^{(1)} = \min(d_{41}^{(0)} + d_{14}^{(0)}, d_{44}^{(0)})$ = min(∞+2, 0) = 0

$d_{45}^{(1)} = \min(d_{41}^{(0)} + d_{15}^{(0)}, d_{45}^{(0)})$ = min(-4+2 =-2, ∞) => d_{45}= -2, π_{45} =1

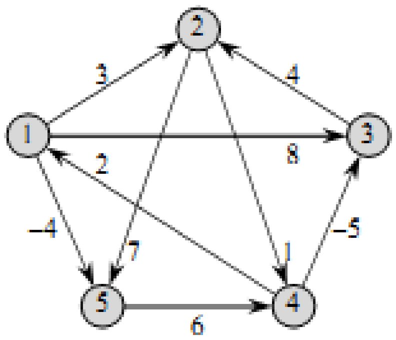
$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

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

Intermediate vertices = {1, 2}

All the nodes to enter node 2 are shown in column 2.

All the nodes to leave node 2 are shown in row 2.



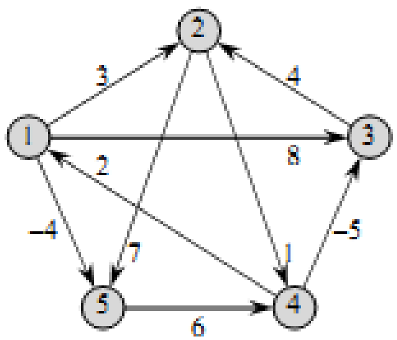
$d_{14}^{(2)} = d_{12}^{(1)} + d_{24}^{(1)} = 1 + 3 = 4,$

$d_{34}^{(2)} = d_{32}^{(1)} + d_{24}^{(1)} = 1 + 4 = 5,$

$d_{35}^{(2)} = d_{32}^{(1)} + d_{25}^{(1)} = 4 + 7 = 11.$

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

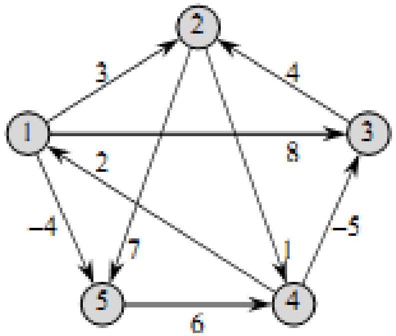
$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$



Intermediate vertices = {1, 2, 3}

$d_{42}^{(3)} = d_{43}^{(2)} + d_{32}^{(2)} = 4 -5 = -1.$

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



Intermediate vertices = {1, 2, 3, 4}

$d_{13}^{(4)} = d_{43}^{(3)}\ d_{43}^{(3)} = 4 - 5 = -1,$

$d_{21}^{(4)} = d_{24}^{(3)} + d_{41}^{(3)} = 1+2 = 3$

$d_{23}^{(4)} = d_{24}^{(3)} + d_{43}^{(3)} = 1 - 5 = -4,$

$d_{25}^{(4)} = d_{24}^{(3)} + d_{45}^{(3)} = 1 - 2 = -1,$

$d_{31}^{(4)} = d_{34}^{(3)} + d_{41}^{(3)} = 5 + 2 = 7,$

$d_{35}^{(4)} = d_{34}^{(3)} + d_{45}^{(3)} = 5 - 2 = 3,$

$d_{51}^{(4)} = d_{54}^{(3)} + d_{41}^{(3)} = 6 + 2 = 8$

$d_{52}^{(4)} = d_{54}^{(3)} + d_{42}^{(3)} = 6 - 1 = 5$

$d_{53}^{(4)} = d_{54}^{(3)} + d_{43}^{(3)} = 6 - 5 = 1$

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

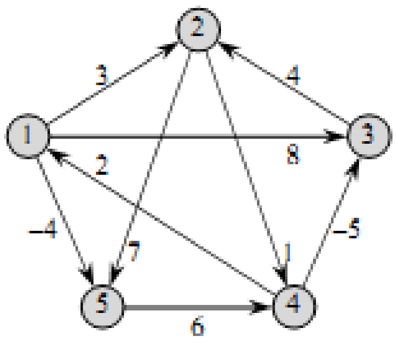
$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

Intermediate vertices = {1, 2, 3, 4, 5}

$d_{12}^{(5)} = d_{15}^{(4)} + d_{52}^{(4)} = -4 + 5 = 1,$

$d_{13}^{(5)} = d_{15}^{(4)} + d_{53}^{(4)} = -4 + 1 = -3$

$d_{14}^{(5)} = d_{15}^{(4)} + d_{54}^{(4)} = -4 + 6 = 2.$



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

The running time of the Floyd-Warshall algorithm is determined by the triply nested for loops of lines 3–7. Because each execution of line 7 takes $O(1)$ time, the algorithm runs in time $\Theta(n^3)$.

The code is tight, with no elaborate data structures, and so the constant hidden in the Θ-notation is small.

Thus, the Floyd-Warshall algorithm is quite practical for even moderate-sized input graphs.

Constructing a shortest path

There are a variety of different methods for constructing shortest paths in the Floyd-Warshall algorithm.

One way is to compute the matrix *D* of shortest-path weights and then construct the predecessor matrix Π from the *D* matrix.

Alternatively, we can compute the predecessor matrix Π while the algorithm computes the matrices $D^{(k)}$

Specifically, we compute a sequence of matrices $\Pi^{(0)}, \Pi^{(1)}, \Pi^{(n)}$, *where* $\Pi = \Pi^{(n)}$ and we define $\pi_{ij}^{(k)}$ as the predecessor of vertex *j* on a shortest path from vertex *i* with all intermediate vertices in the set {1, 2, ..., *k*}.

We can give a recursive formulation of $\pi_{ij}^{(k)}$.

When $k = 0$, a shortest path from *i* to *j* has no intermediate vertices at all.

Thus,

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

For $k \geq 1$, if we take the path $i \sim> k \sim> j$, where $k \neq j$, then the predecessor of j we choose is the same as the predecessor of j we chose on a shortest path from k with all intermediate vertices in the set {1, 2, ..., k- 1}.

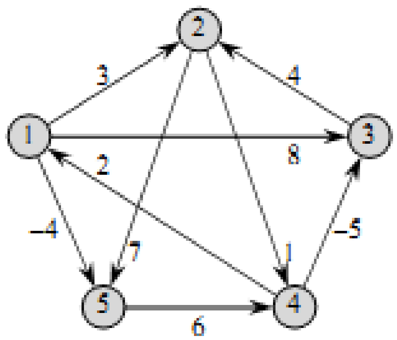
Otherwise, we choose the same predecessor of j that we chose on a shortest path from i with all intermediate vertices in the set {1, 2, ..., k- 1}.

Formally, for $k \geq 1$,

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

Slides 49 - 54 show the sequence of $\Pi^{(k)}$ matrices that the resulting algorithm computes for the graph of slides 38-43.

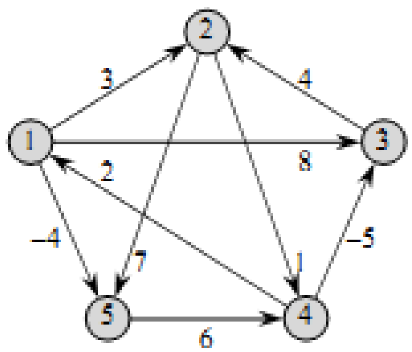
The sequence of matrices $D^{(k)}$ and $\Pi^{(k)}$ computed by the Floyd-Warshall algorithm for the graph shown bellow.



$$D^{(0)} = W$$

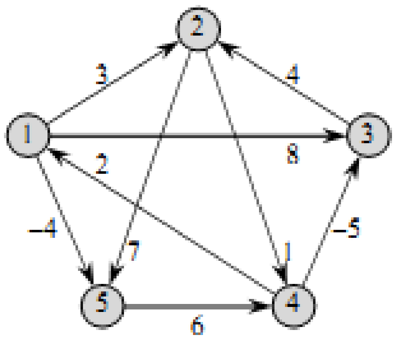
$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(0)} = \begin{pmatrix} \mathrm{NIL} & 1 & 1 & \mathrm{NIL} & 1 \\ \mathrm{NIL} & \mathrm{NIL}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

$$d_{42}^{(1)} = d_{41}^{(0)} + d_{12}^{(0)} \Rightarrow \pi_{42}^{(1)} = \pi_{12}^{(0)} = 1$$
$$d_{45}^{(1)} = (d_{41}^{(0)} + d_{15}^{(0)} \Rightarrow \pi_{45}^{(1)} = \pi_{15}^{(0)} = 1$$



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

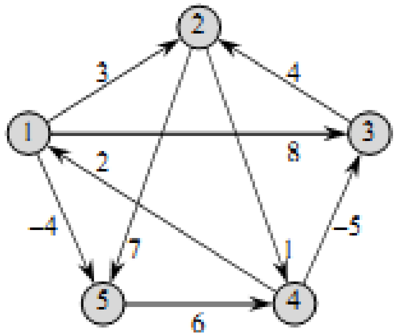
$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL}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



$d_{14}^{(2)} = d_{12}^{(1)} + d_{24}^{(1)} \Rightarrow \pi_{14}^{(2)} = \pi_{24}^{(1)} = 2$
$d_{34}^{(2)} = d_{32}^{(1)} + d_{24}^{(1)} \Rightarrow \pi_{34}^{(2)} = \pi_{24}^{(1)} = 2$
$d_{35}^{(2)} = d_{32}^{(1)} + d_{25}^{(1)} \Rightarrow \pi_{42}^{(1)} = \pi_{12}^{(0)} = 2$

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

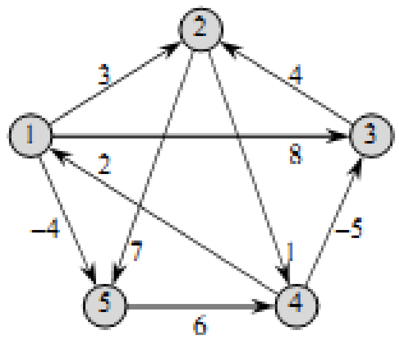
$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL}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



$$d_{42}^{(3)} = d_{43}^{(2)} + d_{32}^{(2)} \Rightarrow \pi_{42}^{(3)} = \pi_{32}^{(2)} = 3$$

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

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



$d_{13}^{(4)} = d_{43}^{(3)} + d_{43}^{(3)} \Rightarrow \pi_{13}^{(4)} = \pi_{43}^{(3)} = 4$
$d_{21}^{(4)} = d_{24}^{(3)} + d_{41}^{(3)} \Rightarrow \pi_{21}^{(4)} = \pi_{41}^{(3)} = 4$
$d_{23}^{(4)} = d_{24}^{(3)} + d_{43}^{(3)} \Rightarrow \pi_{23}^{(4)} = \pi_{43}^{(3)} = 4$
$d_{25}^{(4)} = d_{24}^{(3)} + d_{45}^{(3)} \Rightarrow \pi_{25}^{(4)} = \pi_{45}^{(3)} = 1$
$d_{31}^{(4)} = d_{34}^{(3)} + d_{41}^{(3)} \Rightarrow \pi_{31}^{(4)} = \pi_{41}^{(3)} = 4$
$d_{35}^{(4)} = d_{34}^{(3)} + d_{45}^{(3)} \Rightarrow \pi_{35}^{(4)} = \pi_{45}^{(3)} = 1$
$d_{51}^{(4)} = d_{54}^{(3)} + d_{41}^{(3)} \Rightarrow \pi_{51}^{(4)} = \pi_{41}^{(3)} = 4$
$d_{52}^{(4)} = d_{54}^{(3)} + d_{42}^{(3)} \Rightarrow \pi_{52}^{(4)} = \pi_{42}^{(3)} = 3$
$d_{53}^{(4)} = d_{54}^{(3)} + d_{43}^{(3)} \Rightarrow \pi_{53}^{(4)} = \pi_{43}^{(3)} = 4$

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

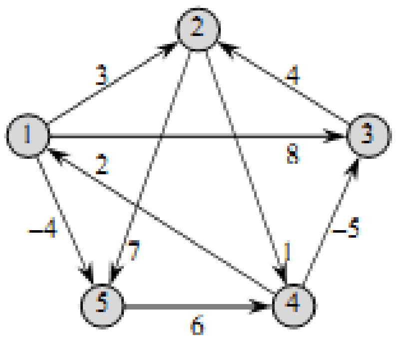
$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \qquad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} &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

Intermediate vertices = {1, 2, 3, 4, 5}

$d_{12}^{(5)} = d_{15}^{(4)} + d_{52}^{(4)} \Rightarrow \pi_{12}^{(5)} = \pi_{52}^{(4)} = 3$

$d_{13}^{(5)} = d_{15}^{(4)} + d_{53}^{(4)} \Rightarrow \pi_{13}^{(5)} = \pi_{53}^{(4)} = 4$

$d_{14}^{(5)} = d_{15}^{(4)} + d_{54}^{(4)} \Rightarrow \pi_{14}^{(5)} = \pi_{54}^{(4)} = 5$.



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

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

Johnson's algorithm for sparse graphs

Johnson's algorithm finds shortest paths between all pairs in $O(V^2 \lg V + VE)$ time.

For sparse graphs, it is asymptotically faster than either repeated squaring of matrices or the Floyd-Warshall algorithm.

The algorithm either returns a matrix of shortest-path weights for all pairs of vertices or reports that the input graph contains a negative-weight cycle.

Johnson's algorithm uses as subroutines both Dijkstra's algorithm and the Bellman-Ford algorithm.

Johnson's algorithm uses the technique of reweighting, which works as follows.

If all edge weights w in a graph $G = (V, E)$ are nonnegative, we can find shortest paths between all pairs of vertices by running Dijkstra's algorithm once from each vertex.

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.

The new set of edge weights *w*' must satisfy 2 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 ⇔ p is also a shortest path from u to v using weight function w'.
2. For all edges (u, v) , the new weight $w'(u, v)$ is nonnegative.

We can preprocess G to determine the new weight function w' in $O(VE)$ time.

Preserving shortest paths by reweighting

The following lemma shows how easily we can reweight the edges to satisfy 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'*.

Lemma 1 (Reweighting does not change shortest paths)

Given a weighted, directed graph $G = (V, E)$ with weight function

w: $E \to R$,

let h: $V \to R$ be any function mapping vertices to real numbers.

For each edge $(u, v) \in E$, define

$$w'(u, v) = w(u, v) + h(u) - h(v) \qquad (9)$$

Let $p = \langle 0, 1, \ldots, k \rangle$ be any path from vertex 0 to vertex k.

Then p is a shortest path from 0 to k with weight function w ⇔ it is a shortest path with weight function w'.

That is,

$w(p) = \delta(0, k) \Leftrightarrow w'(p) = \delta'(0, k)$.

Furthermore, G has a negative-weight cycle using weight function w ⇔ G has a negative-weight cycle using weight function w'.

Proof We start by showing that

$w'(p) = w(p) + h(v_0) - h(v_k)$ (10)

We have

$$
\begin{aligned}
\hat{w}(p) &= \sum_{i=1}^{k} \hat{w}(v_{i-1}, v_i) \\
&= \sum_{i=1}^{k} (w(v_{i-1}, v_i) + h(v_{i-1}) - h(v_i)) \\
&= \sum_{i=1}^{k} w(v_{i-1}, v_i) + h(v_0) - h(v_k) \qquad \text{(because the sum telescopes)} \\
&= w(p) + h(v_0) - h(v_k) .
\end{aligned}
$$

Therefore, any path p from v_0 to v_k has

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

Because $h(v_0)$ and $h(v_k)$ do not depend on the path, 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, $w(p) = \delta(v_0, v_k) \Leftrightarrow w'(p) = \delta'(v_0, v_k)$.

Finally, we show that G has a negative-weight cycle using weight function $w \Leftrightarrow G$ has a negative-weight cycle using weight function w'.

Consider any cycle $c = \langle v_0, v_1, \ldots, v_k \rangle$, where $v_0 = v_k$.

By equation (10),

$w'(c) = w(c) + h(v_0) - h(v_k) = w(c)$;

and thus c has negative weight using $w \Leftrightarrow$ it has negative weight using w'.

Producing nonnegative weights by reweighting

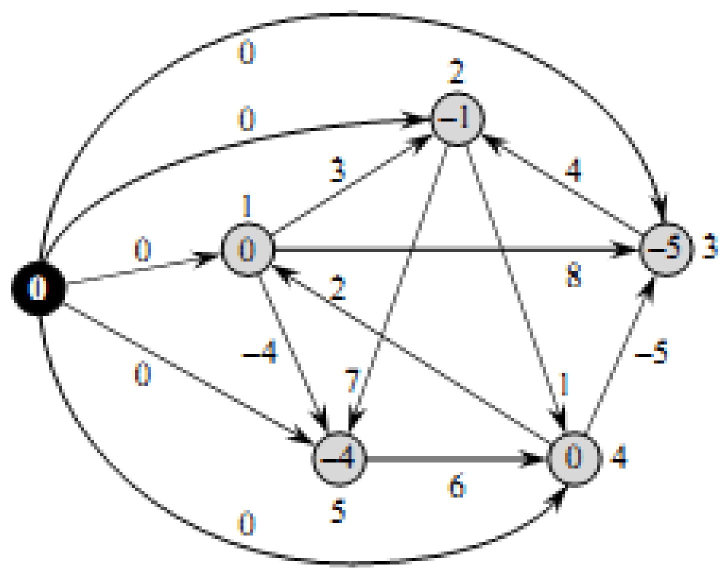
Our next goal is to ensure that the second property holds: we want $w'(u, v)$ to be nonnegative for all edges $(u, v) \in E$.

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 $s \in V$ and $E' = E \cup \{(s, v): v \in V\}$.

We extend the 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 $\Leftrightarrow$ G has no negative-weight cycles.



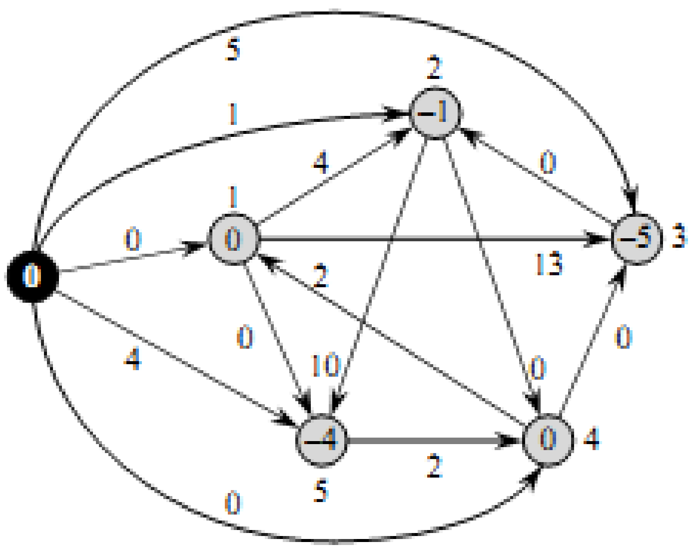
(a)

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 the triangle inequality (Lemma 24.10), we have $h(v) \leq h(u) + w(u, v)$ for all edges $(u, v) \in E'$.

Thus, if we define the new weights w' by reweighting according to equation (9), we have $w'(u, v) = w(u, v) + h(u) - h(v) \geq 0$, and we have satisfied the second property.



(b)

$w'(s, 1) = 0 + 0 - 0 = 0$

$w'(s, 2) = 0 + 0 - (-1) = 1$

$w'(s, 3) = 0 + 0 - (-5) = 5$

$w'(s, 4) = 0 + 0 - 0 = 0$

$w'(s, 5) = 0 + 0 - (-4) = 4$

$w'(1, 2) = 3 + 0 - (-1) = 4$

$w'(1, 3) = 8 + 0 - (-5) = 13$

$w'(1, 5) = -4 + 0 - (-4) = 0$

$w'(2, 4) = 1 + (-1) - 0 = 0$

$w'(2, 5) = 7 + (-1) - (-4) = 10$

$w'(3, 2) = 4 + (-5) - (-1) = 0$

$w'(4, 3) = -5 + 0 - (-5) = 0$

$w'((4, 1) : 2 + 0 - 0 = 2$

$w'((5, 4): 6 - 4 + 0 = 2$

Computing all-pairs shortest paths

Johnson's algorithm to compute all-pairs shortest paths uses the Bellman-Ford algorithm and Dijkstra's algorithm as subroutines.

It assumes implicitly that the edges are stored in *adjacency lists*.

The algorithm returns the usual $|V| \times |V|$ matrix $D = d_{ij}$, where $d_{ij} = \delta(i, j)$, or it reports that the input graph contains a negative-weight cycle.

As is typical for an all-pairs shortest-paths algorithm, we assume that the vertices are numbered from 1 to $|V|$.

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

This code simply performs the actions we specified earlier.

Line 1 produces *G*'.

Line 2 runs the Bellman-Ford algorithm on *G*' with weight function *w* and source vertex *s*.

If *G*', and hence *G*, contains a negative-weight cycle, line 3 reports the problem.

Lines 4–12 assume that *G*' contains no negative-weight cycles.

Lines 4–5 set *h*(*v*) to the shortest-path weight (*s, v*) computed by the Bellman-Ford algorithm for all $v \in V'$.

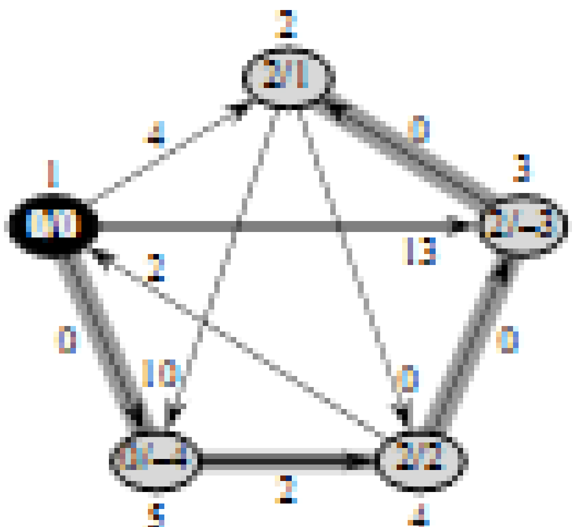
Lines 6–7 compute the new weights *w*'.

For each pair of vertices $u, v \in V$, the **for** loop of lines 9–12 computes the shortest-path weight $\delta'(u, v)$ by calling Dijkstra's algorithm once from each vertex in *V* .

Line 12 stores in matrix entry d_{uv} the correct shortest-path weight $\delta(u, v)$, calculated using equation(10).

Finally, line 13 returns the completed *D* matrix.

Slide 69 depicts the execution of Johnson's algorithm.



(c)

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

$$\delta(1, 1) = 0 + 0 - 0 = 0$$

$$\delta(1, 2) = 2 + (-1) - 0 = 1$$

$$\delta(1, 3) = 2 + (-5) - 0 = -3$$

$$\delta(1, 4) = 2 + 0 - 0 = 2$$

$$\delta(1, 5) = 0 + (-4) - 0 = -4$$

The result of running Dijkstra's algorithm on each vertex of *G* using weight function *w*′.