Chapter 25: All Pairs Shortest Paths.

Context.

1. Throughout the chapter, we assume that our input graphs have no negative cycles.

2. $[G = (V, E), w : E \rightarrow \mathbb{R}]$ is a weighted, directed graph with adjacency matrix $[w_{ij}]$. Wolog, vertices are $1, 2, \ldots, V$.

   $$ w_{ij} = \begin{cases} 
   0, & i = j \\
   \infty, & i \neq j \text{ and } (i, j) \notin E \\
   \text{finite}, & i \neq j \text{ and } (i, j) \in E.
   \end{cases} $$

Note: Output is already $O(V^2)$, because for every vertex pair $(u, v)$, the algorithm must report the weight of the shortest (weight) path from $u$ to $v$. Since the output requires $O(V^2)$ work, there is no significant increase in using an $O(V^2)$ input scheme.

3. The output consists of two matrices:

   $$ \Delta = \delta_{ij} = \text{the weight of the shortest path } i \leadsto j $$

   $$ \Pi = \pi_{ij} = \begin{cases} 
   \text{null}, & \text{if } i = j \text{ or there is no path } i \leadsto j \\
   \text{predecessor of } j \text{ on a shortest } i \leadsto j, & \text{otherwise.}
   \end{cases} $$

4. Row $i$ of $[\pi_{ij}]$ defines a shortest path tree rooted at $i$. That is,

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

   $$ V_{\pi,i} = \{ j : \pi_{ij} \neq \text{null} \} \cup \{i\} $$

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

![Diagram of shortest path tree](image)
5. Can recover a shortest path $i \sim j$ with

\[
\text{Print-Shortest-Path}(\Pi, i, j) \{
\text{if } (i = j) \\
\text{print } i; \\
\text{else if } (\pi_{ij} = \text{null}) \\
\text{print("no shortest path from } i \text{ to } j"); \\
\text{else } \\
\text{Print-Shortest-Path}(\Pi, i, \pi_{ij}); \\
\text{print } j
\}
\]

Obvious approaches.

1. If negative edges exist (but no negative cycles), we can run multiple Bellman-Ford Single-Source-Shortest-Paths, once for each $v \in V$ as the Bellman-Ford source node. Since Bellman-Ford is $O(VE)$, this approach is $O(V^2E)$, which is $O(V^4)$ for dense graphs ($E = \Omega(V^2)$).

2. If there are no negative edges, we can run multiple Dijkstra’s, one for each $v \in V$ as the Dijkstra source node. Since Dijkstra is $O(E \lg V)$, this approach is $O(VE \lg V)$, which is $O(V^3 \lg V)$ for dense graphs ($E = \Omega(V^2)$).

Improvements in this chapter, all of which work in the presence of negative edges, provided there are no negative cycles.

1. Via DP (Dynamic Programming), we can obtain $\Theta(V^3 \lg V)$.

2. Via Warshall’s Algorithm, we can obtain $\Theta(V^3)$.

3. Johnson’s Algorithm delivers $O(V^2 \lg V + VE)$. (Donald B. Johnson, 1977) This approach is better for sparse graphs, specifically for $E = O(V \lg V)$, we break the $V^3$ level with $O(V^2 \lg V)$. 


The DP Algorithm.

Let $V = \{1, 2, \ldots, n\}$, and let $\delta_{ij}$ be the weight of the shortest-weight path from $i$ to $j$.

Suppose $p : i \leadsto j$ is a shortest path from $i$ to $j$ containing at most $m$ edges. As there are no negative cycles, $m$ is finite.

Let $k$ be the predecessor of $j$ on this path. That is, $p : i \leadsto k \rightarrow j$. Then $p' : i \leadsto k$ is a shortest path from $i$ to $k$ by the optimal substructure property. Moreover, the length of $p'$ is at most $m - 1$ edges.

Let $\delta_{ij}^{(m)}$ denote the minimum weight of any path from $i$ to $j$ with at most $m$ edges. We have

$$\delta_{ij}^{(0)} = \begin{cases} 0, & i = j \\ \infty, & i \neq j, \end{cases}$$

the lower result because, if $i \neq j$, the set of paths from $i$ to $j$ containing at most zero edges is the empty set. Recall that the minimum over a set of numbers is the largest number that is less than or equal to every member of the set.

For $m \geq 1$,

$$\delta_{ij}^{(m)} = \min \left\{ \delta_{ij}^{(m-1)}, \min_{1 \leq k \leq n} \left\{ \delta_{ik}^{(m-1)} + w_{kj} \right\} \right\} = \min_{1 \leq k \leq n} \left\{ \delta_{ik}^{(m-1)} + w_{kj} \right\}. \quad (*)$$

The last reduction follows because $w_{jj} = 0$ for all $j$:

$$\left[ \delta_{ik}^{(m-1)} + w_{kj} \right]_{k=j} = \delta_{ij}^{(m-1)} + w_{jj} = \delta_{ij}^{(m-1)}.$$

Now, in the absence of negative cycles, a shortest-weight path from $i$ to $j$ can contains at most $n - 1$ edges. That is

$$\delta_{ij} = \delta_{ij}^{(n-1)} = \delta_{ij}^{(n)} = \delta_{ij}^{(n+1)} = \ldots.$$
We envision a three-dimensional storage matrix, consisting of planes 0, 1, \ldots, n - 1. In each plane, say $k$, is an $n \times n$ matrix of $\delta_{ij}^{(k)}$ values. Envision $i = 1, 2, \ldots, n$ as the row index and $j = 1, 2, \ldots, n$ as the column index in each plane. Recall that we are computing $\delta_{ij}^{(m)}$ in plane $m$ from the row $\delta_{i,*}^{(m-1)}$ in plane $m - 1$.

![Diagram showing the process of updating the matrix](image_url)

Plane 0 is particularly easy to establish since

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

Each subsequent plane fills each of $n^2$ cells by choosing the minimum over $n$ values derived from the previous plane. This procedure then requires $O(n^3)$ time per plane. As we need $n$ planes to reach the desired minimal paths lengths in $\delta_{ij}^{(n-1)}$, we can accomplish the entire computation in $O(n^4)$ time.

But, Bellman-Ford repeated $V$ times is more straightforward, and it also runs in $O(n^4)$ time, even for dense graphs ($E = \Omega(V^2)$).
Extend-Shortest-Paths($\Delta = [\delta_{ij}], W = [w_{ij}]$) {  
  \( n = \) row count of $\Delta$;
  $\Delta' = [\delta'_{ij}] = \) new $n \times n$ matrix;
  for $i = 1$ to $n$
    for $j = 1$ to $n$
      $\delta'_{ij} = \infty$;
      for $k = 1$ to $n$
        $\delta'_{ij} = \min\{\delta'_{ij}, \delta'_{ik} + w_{kj}\}$;
  return $\Delta'$;
}

Slow-All-Pairs-Shortest-Paths($W = [w_{ij}]$) {  
  \( n = \) row count of $W$;
  $\Delta = [\delta_{ij}] = \) new $n \times n$ matrix;
  for $i = 1$ to $n$
    for $j = 1$ to $n$
      $\delta_{ij} = \infty$;
      $\delta_{ii} = 0$;
  for $k = 1$ to $n - 1$  \( // \Theta(n^4)\)
    $\Delta_{out} = \) Extend-Shortest-Paths($\Delta, W$);
    $\Delta = \Delta_{out}$;
  return $\Delta_{out}$;
}
Some helpful observations:

(a) The passage from plane 0 to plane 1 is particularly easy.

\[
\delta_{ik}^{(0)} = \begin{cases} 
\infty, & k \neq i \\
0, & k = i 
\end{cases}
\]

\[
\delta_{ik}^{(0)} + w_{kj} = \begin{cases} 
\infty, & k \neq i \\
w_{ij}, & k = i.
\end{cases}
\]

\[
\delta_{ij}^{(1)} = \min_{1 \leq k \leq n} \{ \delta_{ik}^{(0)} + w_{kj} \} = \min\{\infty, \infty, \ldots, \infty, w_{ij}, \infty, \ldots, \infty\} = w_{ij}.
\]

**Plane 1 is the input weight matrix.**

(b) Moreover, we can modify the initial recursion argument as follows. If, for **even** \(m\), \(p : i \leadsto j\) is a shortest path of at most \(m\) edges, then we can express the path as \(p : i \leadsto k \leadsto j\), for some intermediate vertex \(k\), such that each of \(i \leadsto k\) and \(k \leadsto j\) contains at most \(m/2\) edges. Then

\[
\delta_{ij}^{(m)} = \min \left\{ \delta_{ij}^{(m/2)}, \min_{1 \leq k \leq n} \left\{ \delta_{ik}^{(m/2)} + \delta_{kj}^{(m/2)} \right\} \right\} = \min_{1 \leq k \leq n} \left\{ \delta_{ik}^{(m/2)} + \delta_{kj}^{(m/2)} \right\},
\]

again since \(\delta_{ik}^{(m/2)} + \delta_{kj}^{(m/2)} = \delta_{ij}^{(m/2)}\) when \(k = j\).

Recall that \(\delta_{jj}^{\text{whatever}} = 0\) can not be improved in the absence of negative cycles.

(c) Note similarity to previous recursion:

\[
\delta_{ij}^{(m)} = \min_{1 \leq k \leq n} \left\{ \delta_{ik}^{(m-1)} + w_{kj} \right\} = \min_{1 \leq k \leq n} \left\{ \delta_{ik}^{(m-1)} + \delta_{kj}^{(1)} \right\}.
\]

So, Extend-Shortest-Paths(\(\Delta = [\delta_{ij}], W = [w_{ij}]\)) can be used as Extend-Shortest-Paths(\(\Delta_{ij}^{(m/2)}, \Delta_{ij}^{(m/2)}\)).
We can now compute as follows,

\[
\Delta^{(1)} = [\delta^{(1)}_{ij}] = [w_{ij}] = W
\]
\[
\Delta^{(2)} = [\delta^{(2)}_{ij}] = \text{Extend-Shortest-Paths}(\Delta^{(1)}, \Delta^{(1)})
\]
\[
\Delta^{(4)} = [\delta^{(4)}_{ij}] = \text{Extend-Shortest-Paths}(\Delta^{(2)}, \Delta^{(2)})
\]

\[\vdots\]

Since \([\delta^{(k)}_{ij}] = [\delta^{(n-1)}_{ij}]\) for all \(k \geq n - 1\), this computation will stabilize after at most \([\lg n]\) iterations.

Fast-All-Pairs-Shortest-Paths\((W = [w_{ij}])\) { 
  \n  \ n = \text{row count of } W;
  \n  \Delta = W;
  \n  m = 1;
  \n  \text{while } m < n - 1 \{ 
  \n  \ \Delta = \text{Extend-Shortest-Paths}(\Delta, \Delta);
  \n  \ \ m = 2m;
  \n  \}\n  \n  \text{return } \Delta;
  
}

This DP computation uses \(\lg n\) passes through \text{Extend-Shortest-Paths}, which is a \(\Theta(n^3)\) algorithm. Therefore the Fast-All-Pairs-Shortest-Paths algorithm is \(\Theta(n^3 \lg n)\). Consequently, this algorithm

1. beats the \(\Theta(n^4)\) required by multiple passes through the Bellman-Ford algorithm,

2. ties the \(\Theta(n^3 \lg n)\) required by multiple passes through the Dijkstra algorithm (which is not available is the graph contains negative edges),

3. and accommodates negative edges (although not negative cycles).
The **Floyd-Warshall Algorithm** also accommodates negative edges, although the input graph must be free of negative cycles. Let $G = (V, E)$, where $V = \{1, 2, \ldots, n\}$.

Define $\delta^{(k)}_{ij}$ to be the weight of the shortest path $p : i \leadsto j$ with all *intermediate* vertices constrained to lie in the set $\{1, 2, \ldots, k\}$.

We proceed via induction from the basis $\delta^{(0)}_{ij} = w_{ij}$, which reflects the constraint that there be no intermediate vertices. The induction step is then

$$
\delta^{(k)}_{ij} = \min \left\{ \delta^{(k-1)}_{ij}, \delta^{(k-1)}_{ik} + d^{(k-1)}_{kj} \right\},
$$

reflecting the fact that the shortest path with all of $\{1, 2, \ldots, k\}$ available as intermediates is either (a) a path that does not use vertex $k$ as an intermediary, or (b) a path that does use $k$.

**Preliminary-Floyd-Warshall**$(W = [w_{ij}])$ { // $\Theta(n^3)$

$n =$ row count of $W$;

$\Delta = [\delta_{ij}] = W$;

$\Delta' = [\delta'_{ij}] =$ new $n \times n$ matrix;

for $k = 1$ to $n$ {

for $i = 1$ to $n$

for $j = 1$ to $n$

$\delta'_{ij} = \min \{\delta_{ij}, \delta_{ik} + \delta_{kj}\}$;

$\Delta = \Delta'$;

}$

return $\Delta'$;

}

Thus Floyd-Warshall, a simple data-structures algorithm, beats our best dynamic-programming algorithm: $\Theta(n^3)$ against $\Theta(n^3 \lg n)$. We now further develop Floyd-Warshall to capture the shortest paths themselves.
Recall that $\pi_{ij}$ is the predecessor of $j$ on a shortest path $p : i \leadsto j$, null if no such path exists. We define $\pi_{ij}^{(k)}$ to be the predecessor of $j$ on a shortest path $p : i \leadsto j$ constrained such that all intermediate vertices lie in the set $\{1, 2, \ldots, k\}$. We have

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

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)}, \quad \delta_{ij}^{(k-1)} \leq \delta_{ik}^{(k-1)} + \delta_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)}, \quad \delta_{ij}^{(k-1)} > \delta_{ik}^{(k-1)} + \delta_{kj}^{(k-1)} \end{cases}$$

In the context of

Preliminary-Floyd-Warshall($W = [w_{ij}]$) \{ \hspace{1cm} // $\Theta(n^3)$

\begin{align*}
n &= \text{row count of } W; \\
\Delta &= [\delta_{ij}] = W; \\
\Delta' &= [\delta'_{ij}] = \text{new } n \times n \text{ matrix;}
\end{align*}

for $k = 1$ to $n$ \{ 

\begin{align*}
&\text{for } i = 1 \text{ to } n \\
&\text{for } j = 1 \text{ to } n \\
&\delta'_{ij} = \min\{\delta_{ij}, \delta_{ik} + \delta_{kj}\}; \\
&\Delta = \Delta'; \\
&\text{return } \Delta';
\end{align*}

\}

the computation for $\pi_{ij}^{(k)}$, the upper option is used when $\delta_{ij}^{(k)}$ is achieved via a path that uses only intermediate vertices in $\{1, 2, \ldots, k-1\}$. That is, the predecessor of $j$ is unchanged from $\pi_{ij}^{(k-1)}$.

The lower option is used when the new intermediate vertex $k$ produces a shorter path via $k$. That is, the new shortest path has the form $p : i \leadsto k \leadsto j$, in which case the new predecessor of $j$ on this path is the predecessor of $j$ on best path $q : k \leadsto j$ constrained to intermediaries in $\{1, 2, \ldots, k-1\}$.
Floyd-Warshall($W = [w_{ij}]$) \{ // $\Theta(n^3)$
    $n =$ row count of $W$;
    $\Delta = [\delta_{ij}] = W$;
    $\Delta' = [\delta'_{ij}] =$ new $n \times n$ matrix;
    $\Pi = [\pi_{ij}] =$ new $n \times n$ matrix;
    $\Pi' = [\pi'_{ij}] =$ new $n \times n$ matrix;
    for $i = 1$ to $n$
        for $j = 1$ to $n$
            if ($i = j$) or ($w_{ij} = \infty$)
                $\pi_{ij} =$ null;
            else
                $\pi_{ij} = i$;
    for $k = 1$ to $n$ {
        for $i = 1$ to $n$
            for $j = 1$ to $n$
                if $\delta_{ij} \leq \delta_{ik} + \delta_{kj}$ {
                    $\delta'_{ij} = \delta_{ij}$;
                    $\pi'_{ij} = \pi_{ij}$;
                } 
                else {
                    $\delta'_{ij} = \delta_{ik} + \delta_{kj}$;
                    $\pi'_{ij} = \pi_{kj}$;
                } 
            $\Delta =$ $\Delta'$;
            $\Pi =$ $\Pi'$;
        $\Delta = \Delta'$;
        $\Pi = \Pi'$;
    } 
    return ($\Delta'$, $\Pi'$);
}
Transitive closure via Warshall’s Algorithm.

Definition. Given a directed graph $G = (V,E)$, the transitive closure of $G$ is $G^* = (V,E^*)$, where

$$E^* = \{(i,j) : \text{there exists a path } i \rightsquigarrow j \text{ in } G\}.$$ 

First Approach.

1. Redefine the edge set:

   $$w_{ij} = \begin{cases} 
   0, & i = j \\
   \infty, & i \neq j \text{ and } (i,j) \notin E \\
   1, & i \neq j \text{ and } (i,j) \in E.
   \end{cases}$$

2. Run the Floyd-Warshall Algorithm $\Rightarrow \Delta = [\delta_{ij}]$, a matrix of shortest paths between each $i,j$ pair.

3. Establish $E^* = \{(i,j) : \delta_{ij} < \infty\}$. 
Second Approach: Rework Floyd-Warshall to use logical operations.

1. Redefine the edge set:

\[
\begin{align*}
    w_{ij} &= \begin{cases} 
        \text{false}, & i \neq j \text{ and } (i, j) \notin E \\
        \text{true}, & i = j \text{ or } (i, j) \in E.
    \end{cases}
\end{align*}
\]

2. Run the following revised Floyd-Warshall algorithm, which returns \( \Delta' = [\delta'_{ij}] \), a matrix of true-false values.

\[
\text{Floyd-Warshall}(W = [w_{ij}]) \{ \quad \text{// } \Theta(n^3) \}
\]

\[
\begin{align*}
    n &= \text{row count of } W; \\
    \Delta &= [\delta_{ij}] = W; \\
    \Delta' &= [\delta'_ij] = \text{new } n \times n \text{ matrix}; \\
    \text{for } k = 1 \text{ to } n \{ \\
    &\quad \text{for } i = 1 \text{ to } n \\
    &\quad \quad \text{for } j = 1 \text{ to } n \\
    &\quad \quad \quad \delta'_{ij} = \delta_{ij} \lor (\delta_{ik} \land \delta_{kj}); \\
    &\quad \quad \Delta = \Delta'; \\
    &\quad \text{return } \Delta'; \\
\}
\]

On conclusion, \( E^* = \{(i, j) : \delta'_{ij} = \text{true}\} \).
**Johnson’s Algorithm.**

We want to exploit Dijkstra’s single-source-shortest paths algorithm, but we need to circumvent the requirement that the graph cannot contain negative edge weights.

We continue to comply with the requirement that there be no negative cycles.

So, given a directed $G = (V, E)$ with no negative weight cycles, we need a new weighting function, say $\hat{w}$, with the following properties.

1. For all $u, v \in V$, $p : u \rightsquigarrow v$ is a shortest $w$-weighted path from $u$ to $v$ if and only if $p$ is a shortest $\hat{w}$-weighted path from $u$ to $v$.

2. $\hat{w}(u, v) \geq 0$ for all $(u, v) \in E$.

**Note:** We cannot obtain the new weighting by simply adding a constant to all weights, sufficiently large to boost all negative weights into positive territory:

![Diagram of graph before and after weighting](image)
Lemma 25.1: Given a directed weighted $G = (V, E)$, $w : E \to \mathcal{R}$, Let $h : V \to \mathcal{R}$ be an arbitrary function. Define $\hat{w} : E \to \mathcal{R}$ via

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

Then

1. $p : u \leadsto v$ is a shortest $w$-weighted path from $u$ to $v$ if and only if $p$ is a shortest $\hat{w}$-weighted path from $u$ to $v$.

2. $G$ has a negative weight cycle via $w$ if and only if $G$ has a negative weight cycle via $\hat{w}$.

Proof. Let $p = (v_1, v_2, \ldots, v_k)$ be any path in $G$. We have

$$\hat{w}(p) = \sum_{i=2}^{k} \hat{w}(v_{i-1}, v_i) = \sum_{i=2}^{k} \left[w(v_{i-1}, v_i) + h(v_{i-1}) - h(v_i)\right]$$

$$= \left(\sum_{i=2}^{k} w(v_{i-1}, v_i)\right) + \left(\sum_{i=2}^{k} (h(v_{i-1}) - h(v_i))\right) = w(p) + h(v_1) - h(v_k)$$

$$\hat{w}(p) - w(p) = h(v_1) - h(v_k), \text{ regardless of path.}$$

The component $h(v_1) - h(v_k)$ is constant across all paths $p' : v_1 \leadsto v_k$.

Now suppose that $p$ is a shortest $\hat{w}$-weighted path from $v_1$ to $v_k$, and suppose there exists some path $q : v_1 \leadsto v_k$ with $w(q) < w(p)$. Then,

$$\hat{w}(p) - w(p) = h(v_1) - h(v_k) = \hat{w}(q) - w(q)$$

$$\hat{w}(p) - \hat{w}(q) = w(p) - w(q) > 0$$

$$\hat{w}(q) < \hat{w}(p),$$

a contradiction. Therefore $p$ is also a shortest $w$-weighted path from $v_1$ to $v_k$. 

14
Similarly, if \( p \) is a shortest \( w \)-weighted path from \( v_1 \) to \( v_k \) and there exists some path \( q : v_1 \rightarrow v_k \) with \( \hat{w}(q) < \hat{w}(p) \), we again derive a contradiction:

\[
\hat{w}(p) - w(p) = h(v_1) - h(v_k) = \hat{w}(q) - w(q) \\
\hat{w}(p) - \hat{w}(q) = w(p) - w(q) > 0 \\
\hat{w}(q) < w(p).
\]

Hence \( p \) is a shortest \( w \)-weighted path from \( v_1 \) to \( v_2 \) if and only if \( p \) is a shoted \( \hat{w} \)-weighted path from \( v_1 \) to \( v_k \).

For any cycle, say \( p = (v_1, v_2, \ldots, v_k = v_1) \),

\[
\hat{w}(p) = w(p) + h(v_1) - h(v_k) = w(p) + h(v_1) - h(v_1) = w(p).
\]

Hence if \( p \) is a negative cycle under the \( w \) weights, it is also a negative cycle under the \( \hat{w} \) weights, and vice versa. ■
Both path lengths shift by $3 - 0 = 3$, and therefore the lower remains the shorter.
Returning to Johnson’s Algorithm, we modify the input graph as suggested in the sketch.

Specifically, we construct graph $G' = (V', E')$, where

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

We extend the weight function to $E'$ by defining $w(s, v) = 0$ for all $v \in V$.

Observations.

1. $s$ has in-degree zero. Consequently, no shortest paths in $G'$, other than those with source $s$, contain $s$.
2. $G'$ has a negative weight cycle if and only if $G$ has a negative weight cycle.
Now, suppose \( G \) has no negative weight cycle. Consequently, \( G' \) has no negative weight cycle. Letting \( \delta(s, v) \) denote the length of the shortest path from \( s \) to \( v \) in \( G' \), we define \( h(v) = \delta(s, v) \). Then, we have

\[
\delta(s, v) \leq \delta(s, u) + w(u, v) \quad \text{(triangle inequality)}
\]
\[
h(v) \leq h(u) + w(u, v)
\]
\[
\hat{w}(u, v) = w(u, v) + h(u) - h(v) \geq 0, \quad \text{for all} \ (u, v) \in G'.
\]
Johnson($G = (V, E), w : E \to \mathcal{R}$)

Compute $G' = (V', E')$;  \hspace{1cm}  //  $\Theta(V + E)$

if Bellman-Ford($G', w, s$) = false {  \hspace{1cm}  //  $\Theta(VE)$
        print(“negative cycle”);
        return;
    }  

for $v \in V'$
      $v.h = v.d$;  \hspace{1cm}  //  $\delta(s, v)$ from Bellman Ford

for $(u, v) \in E'$
      //  $\Theta(V + E)$
      $(u, v).\hat{w} = (u, v).w + u.h - v.h$;  \hspace{1cm}  //  all non-negative

for $u \in G.V$ {
      Dijkstra($G, \hat{w}, u$);  \hspace{1cm}  //  $O(VE \lg V)$ puts $\hat{\delta}(u, v)$ into $v.d$

for $v \in G.V$ {
      $\Delta_{uv} = v.d + v.h - u.h$;  \hspace{1cm}  //  $\Theta(V^2)$
      $\Pi_{u,v} = v.\pi$
  }

return ($\Delta, \Pi$);
}

Observations.

1. When $\Delta_{uv}$ is assigned, we get

$\Delta_{uv} = \hat{\delta}(u, v) + h(v) - h(u) = [\delta(u, v) + h(u) - h(v)] + h(v) - h(u) = \delta(u, v)$,

the shortest distance from $u$ to $v$ as desired.

2. Since the naked Dijkstra is $O(E \lg V)$, we have $O(VE \lg V)$ for the loop in which that call is embedded.

3. The total is then $O(VE \lg V)$, which beats $V^3$ for sparse graphs ($E = O(V^{2-\epsilon})$):

$$\frac{V^{3-\epsilon} \lg V}{V^3} = \frac{\lg V}{V^\epsilon} \to 0.$$  

4. The author notes slightly better performance with Fibonacci heaps underlying Dijkstra’s minHeap.