All pair shortest path - no http://www.math.ucr.edu/~lucavitam/longest edges no -ve length edges.

Input $G = (V, E)$, EC (Edge Cost) matrix

Method 1:

Find shortest paths $p \le l$ edges from $v_i \rightarrow v_j$ for all $i,j$ for $l = 1, 2, \ldots n-1$ in turn.

$S_{l, C, i, j} = EC_{i, j}$. $i \neq j$: $S_{l, C, i, i} = 0$

$S_{l, C, i, j} = \min \{ S_{l, C, i, j}, S_{l, C, i, k} + S_{l, C, k, j} \}$

for $l = 1, 2, \ldots \text{for each } l$

$S$: overall.

For $l \ge 3$

Improved $l$ algorithm:

$S_{l, C, i, j} = \min \{ S_{l-1, C, i, j} \}, S_{l, C, i, k} + S_{l, C, k, j} \}$

After $\text{flagn} \text{iterations have computed}$

$S_{\text{flagn}} = S_{n}$

Correctiveness: best $l+1$ edge path $\rightarrow$ best $l$ edge path

$S_{l+1, C, i, j} = S_{l, C, i, j}$

best edge path
Alternative solution that achieves $O(n^3)$ runtime.

**Observation:** As a shortest path has no -ve length cycles there is a cycle free shortest path.

![Cycle has length 0]

So each vertex occurs at most once on the path.

Hence, shortest path from $v_i \rightarrow v_j$ has one of two forms:

1. It does not include $v_n$.

2. It includes $v_n$.

![Path with vertex $v_n$]

Hence, if can solve shortest path problem with $v_n$ excluded as intermediate vertex, can then solve it with $v_n$ allowed.

Let $S_{n-1}(i,j)$ denote shortest path from $v_i \rightarrow v_j$ with $v_n$ excluded as intermediate vertex.

$S_n(i,j)$ denote length shortest path from $v_i \rightarrow v_j$.

$$S_n(i,j) = \min \{ S_{n-1}(i,j), S_n(i,n) + S_n(n,j) \}$$
More generally:

Let $S_k[i,j]$ denote the length of the shortest $v_i \rightarrow v_j$ path with $v_{k+1}, v_{k+2}, \ldots, v_n$ excluded as intermediate vertices.

Then $S_k[i,j] = \min \{ S_{k-1}[i,j], S_{k-1}[i,k] + S_{k-1}[k,j] \}$

And so $S[i,j] = ECC[i,j]$ - no intermediate vertices allowed.

Just another dynamic program.
Program (Because the code is so simple, an iterative version is better than a recursive version.)

\[
\begin{align*}
\text{for } j & \leftarrow 1 \text{ to } n \text{ do} \\
& \quad \text{for } i \leftarrow 1 \text{ to } n \text{ do} \\
& \quad \quad S_{i,j} \leftarrow \infty \quad [i,j] \\
& \quad \text{end} \\
& \quad \text{end} \\
& \quad \text{end} \\
& \text{end} \\
& \text{for } k \leftarrow 1 \text{ to } n \text{ do} \\
& \quad \text{for } j \leftarrow 1 \text{ to } n \text{ do} \\
& \quad \quad \text{for } i \leftarrow 1 \text{ to } n \text{ do} \\
& \quad \quad \quad S_{i,j} \leftarrow \min \{ S_{i,j} \}, S_{i,h} + S_{h,j} \} \\
& \quad \text{end} \\
& \quad \text{end} \\
& \text{end} \\
\end{align*}
\]

Clearly \( O(n^3) \) run time: main line repeated \( n^3 \) times, at a cost of \( O(1) \) each time.

Improvement

One \( S \) array suffices.

For replacing \( S_{k,j} \) by \( S_{k} \) cannot do harm - it is only potentially finding a shorter path sooner.

But, in fact

\[
\begin{align*}
S_{k,i} \leftarrow \min \{ S_{k,i}, S_{k,j} \} \\
S_{k,j} \leftarrow \min \{ S_{k,i} + S_{i,j} \}
\end{align*}
\]

and so the overwriting has no effect.
Example

$S_0 = EC = \begin{bmatrix} a & b & c \\ 0 & 1 & 13 \\ 11 & 0 & 3 \\ 2 & 11 & 0 \end{bmatrix}$

$S_1 = \begin{bmatrix} 0 & 1 & 13 \\ 11 & 0 & 3 \\ 2 & 3 & 0 \end{bmatrix}$

$S_2 = \begin{bmatrix} 0 & 1 & 4 \\ 11 & 0 & 3 \\ 2 & 3 & 0 \end{bmatrix}$

$S_3 = \begin{bmatrix} 0 & 1 & 4 \\ 5 & 0 & 3 \\ 2 & 3 & 0 \end{bmatrix}$
Finding Shortest Paths

In the chain array \( C[i,n,1:n] \) (a.k.a. Path Recovery array \( P \)) keep choice of intermediate vertex \( j \).

Initialize \( C[i,j] = 0 \) (no intermediate vertex).

\[
\begin{align*}
\text{for } k & = 1 \text{ to } n \text{ do} \\
\text{for } j & = 1 \text{ to } n \text{ do} \\
\text{for } i & = 1 \text{ to } n \text{ do} \\
\text{if } s[i,j] > s[i,k] + s[k,j] \\
\quad \text{then do} \\
\quad s[i,j] & = s[i,k] + s[k,j]; \\
\quad C[i,j] & = k \\
\text{end} \\
\text{end} \\
\text{end}
\end{align*}
\]
Print Path \((i, j)\) (or PP(i, j) for short)
\([*\text{prints path from } i \text{ to } j \text{ excluding } j \text{ }*\])

\[PP(i, j)\]
\[k \leftarrow C(i, j);\]
\[\begin{array}{l}
\text{if } k = 0 \text{ then } \text{print}(i) \quad (* \text{print path from } i \text{ to } j \text{ excluding } j *)
\end{array}\]
\[\text{else do}\]
\[\begin{array}{l}
PP(i, k) \quad (* \text{print path from } i \text{ to } k \text{ excluding } k *)
\end{array}\]
\[PP(k, j) \quad (* \text{print path from } k \text{ to } j \text{ excluding } j *)\]
\[\text{end}\]
\[\text{end}\]

\[FP(i, j) = \text{FullPath}(i, j) \quad (* \text{print path from } i \text{ to } j *)\]

\[PP(i, j) \quad (* \text{print path from } i \text{ to } j \text{ excluding } j *)\]
\[\text{print}(j)\]
\[\text{end}\]
Other solutions

Save 2nd vertex on best path (i.e. first edge)

Initialize: \( C[i,j] = \begin{cases} 0 & \text{if } E[i,j] < \infty \\ \infty & \text{otherwise} \end{cases} \)

Update line:

\[
\text{if } S[i,j] > S[i,h] + S[h,j] \\
\text{then do} \\
S[i,j] \leftarrow S[i,h] + S[h,j] \\
C[i,j] \leftarrow C[i,h]
\]

\text{PP}(i,j) (* prints all of } i \to j \text{ path except } i * \\
\text{let } l \leftarrow C[i,j] \\
\text{Print}(l); \\
\text{if } j \neq l \text{ then } \text{PP}(l,j) (* prints all of } l \to j \text{ path except } l * \\
\text{end}

\text{FPP}(i,j) (* prints all of } i \to j \text{ path *) \\
\text{Print}(l); \\
\text{PP}(i,j) (* prints all of } i \to j \text{ path except } i * \\
\text{end}

Similarly, can save next to last vertex.