Another approach to the APSP problem, using adjacency matrix representation:

\[ G = (V, E, w) \] is represented with an adjacency matrix \( A \) whose \((i, j)\)th entry is

\[
    a_{ij} = \begin{cases} 
    w(i, j) & \text{if } (i, j) \in E \\
    \infty & \text{if } i \neq j \text{ and } (i, j) \notin E \\
    0 & \text{if } i = j 
    \end{cases}
\]

Goal is compute a \(|V| \times |V|\) matrix \( D \) whose \((i, j)\)th entry is

\[ d_{ij} = \text{weight of shortest } i \to j \text{ path} \]  
(denoted \( \text{dist}(i, j) \) as before)

(See text for how to compute the paths themselves.)

Let \( n = |V| \) from now on.

Use dynamic programming: \[ \text{Floyd-Warshall} \]

1. Develop recursive expression for solution.

Let \( D^K \) be the matrix whose elements are

\[
    d_{ij}^K = \text{weight of shortest } i \to j \text{ path, all of whose intermediate nodes are in } \{1, 2, \ldots, K\}.
\]
**Ex:**

\[ d_{4}^{0} = \infty \] since 1 and 4 are not adjacent

\[ d_{4}^{1} = \infty \] since letting 1 be intermediate doesn't help

\[ d_{4}^{2} = 15 \] since 1, 2, 4 has wt. 15

\[ d_{4}^{3} = 10 \] since 1, 2, 3, 4 has wt. 10

\[ d_{4}^{4} = 10 \]

**Goal:** \( D^{n} \) (allow all nodes to be potential intermediate nodes)

**Basis:** \( D^{0} = A \) (the adjacency matrix, i.e., shortest \( i \rightarrow j \) paths w/ no intermediate nodes)

**Recursion:** \( D^{k} \) has the entries

\[ d_{ij}^{k} = \min \{ d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1} \} \]

**Why?**

\[ \begin{array}{c}
\text{all intermediate nodes are in } \{ i, \ldots, k-1, k \} \\
\end{array} \]

Now consider \( k \) as a possible intermediate node.

**Take whichever \( i \rightarrow j \) path is shorter.**
2. Determine dependencies among sub-problems.

\[ D^0 \rightarrow D^1 \rightarrow D^2 \rightarrow \ldots \rightarrow D^n \]

i.e., compute \( D^0 \), then \( D^1 \), then \( D^2 \), etc.

3. Bottom-up solving of sub-problems consistent with these dependencies, storing intermediate results in a table.

Here we'd need a 1-dimensional table of \( n+1 \) entries. Each entry is an \( n \times n \) matrix.

1. \( D^0 := A \)

2. for \( k := 1 \) to \( n \) do
   
   /* compute \( D^k \) using \( D^{k-1} \)

3. for \( i := 1 \) to \( n \) do

4. for \( j := 1 \) to \( n \) do

5. \[ d_{ij}^k := \min (d_{ij}^{k-1}, d_{ik}^k + d_{kj}^k) \]
   
   endfor

   endfor

   endfor

6. return \( D^n \)

**Running Time:**

1. \( O(n^2) \)

2 - 4. 3 nested loops \( \Rightarrow O(n^3) \) iterations

5. \( O(1) \)

\( O(n^3) \) total
4. Problem-specific optimizations:

   Space is $O(n^3)$.

   Can reduce to $O(n^2)$ by just using 2 matrices.

   Can even improve to just 1 matrix - see HW prob.

Ex:

$$A = \begin{bmatrix}
0 & 4 & 11 \\
6 & 0 & 2 \\
3 & 7 & 0 \\
\end{bmatrix}$$

$$D^0 \quad D^1 \quad D^2 \quad D^3$$

Same as $A$

$$\begin{bmatrix}
0 & 4 & 11 \\
6 & 0 & 2 \\
3 & 7 & 0 \\
\end{bmatrix}$$

improved

$$\begin{bmatrix}
0 & 4 & 6 \\
5 & 0 & 2 \\
3 & 7 & 0 \\
\end{bmatrix}$$

Sometimes, instead of wanting to know the shortest path between 2 nodes, all you want to know is whether there exists a path between them. I.e., want to compute the transitive closure of $G = (V,E)$.

Formally it is $G^* = (V, E^*)$, where $E^* = \{(i,j) : \text{exists a path in } G\}$.
One approach: add some weights to the edges, do an APSP alg., and then check for each i and j whether the answer is finite or infinite.

A slightly more direct version of this is to modify Floyd-Warshall so that

- the D matrices are boolean,

\[ d_{ij}^k = 1 \text{ iff there is a path from } i \text{ to } j \text{ whose intermediate nodes are in } \{1, 2, \ldots, k\} \]

- the calculation in line 5 becomes

\[ d_{ij}^k := d_{ij}^{k-1} \text{ OR } (d_{ik}^{k-1} \text{ AND } d_{kj}^{k-1}) \]