Another dynamic programming algorithm for APSP:
Recall Floyd-Warshall: we keep increasing the size of the set from which intermediate nodes can be drawn
\[ d_{ij}^k = \min \{ d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1} \} \]

Shortest path distance from \( i \) to \( j \) using intermediate nodes in \( \{1, \ldots, k\} \)

We can consider a different approach: keep increasing the length of the paths from \( i \) to \( j \):
\[ d_{ij}^m = \min \{ d_{ij}^{m-1}, \min_{k \in \mathbb{N}} \{ d_{ik}^{m-1} + w(k, j) \} \} \]

Shortest path distance from \( i \) to \( j \) using paths of length \( \leq m \)

\[ d_{ij}^{m-1} (\text{allow all paths of length} \leq n-1; \text{since no negative weight cycles, this includes all possible shortest paths}) \]

Goal: \( L^{n-1} \)

Basis: \( L^1 = A \) (adjacency matrix with weights, i.e., shortest \( i \to j \) paths of length \( \leq 1 \))
Recursion: $L^m$ has entries

$$
l_{ij}^m = \min \left\{ l_{ij}^{m-1}, \min_{1 \leq k \leq n} \left( l_{ik}^{m-1} + w(k, j) \right) \right\}
$$

Dependencies: $L^1 \to L^2 \to L^3 \to \ldots \to L^{n-1}$

Pseudocode:

1. $L^1 := A$
2. For $m := 2$ to $n-1$ do
   - /* compute $L^m$ using $L^{m-1} x/$
     - For $i := 1$ to $n$ do
       - For $j := 1$ to $n$ do
         - For $k := 1$ to $n$ do
           - $l_{ij}^m := \min\{ l_{ij}^m, l_{ik}^m + w(k, j) \}$
       - Return $L^{m-1}$

Running Time is $O(V^4) = O(n^4)$.

Can reduce this by repeated squaring, i.e.,

- Only compute $L^2, L^4, L^8, L^{16}, \ldots, L^{x}$, where
  - $x$ is smallest power of 2 that is $\geq n-1$.

- By properties of shortest path, $L^x = L^{n-1}$.
- Result is an $O(V^3 \log V) = O(n^3 \log n)$ time alg.

This is not as good as Floyd-Warshall, but

- This algorithm does allow specializing to the SSSP algorithm, which Floyd-Warshall does not.

Idea is to consider just one value for $i$, the single source.
After dropping the $i$ parameter, the data structures $L^1, L^2, \ldots, L^{n-1}$ are now 1-D instead of 2-D.

$L^1 := A$ /* weighted adjacency matrix */
for $m = 2$ to $n-1$ do /* each iteration extends length of considered paths by 1 */
  for $j := 1$ to $n$ do /* consider every target node */
    $l_j^m := \infty$
    for $k := 1$ to $n$ do /* check all incoming neighbors of $j$ */
      $l_j^m := \min \{ l_j^m, l_k^{m-1} + w(k,j) \}$
  return $L^{n-1}$

Running Time is $O(n^3)$.
Reduce to $O(n^2 \log n)$ using repeated squaring.

Can reduce time further to $O(nm) = O(VE)$ by taking advantage of adjacency list to consider only those values of $k$ such that $(k,j)$ is an edge in the graph.

This is Bellman-Ford SSSP algorithm — go to slides.
(Read Sec. 24.1)