All-Pairs Shortest Paths (Ch. 25)

The all-pairs shortest path problem (APSP)
input: a directed graph \( G = (V, E) \) with edge weights

goal: find a minimum weight (shortest) path between every pair of
vertices in \( V \)

Can we do this with algorithms we've already seen?

Solution 1: run Dijkstra's algorithm \( V \) times, once with each \( v \in V \) as the
source node (requires no negative-weight edges in \( E \))

If \( G \) is dense with an array implementation of \( Q \)

\( O(V \cdot V^2) = O(V^3) \) time

If \( G \) is sparse with a binary heap implementation of \( Q \)

\( O(V \cdot ((V + E) \cdot \log V)) = O(V^2 \log V + VE \log V) \) time

Solution 2: run the Bellman-Ford algorithm \( V \) times (negative edge
weights allowed), once from each vertex.

\( O(V^2 E) \), which on a dense graph is \( O(V^4) \)

Solution 3: use an algorithm designed for the APSP problem.
E.g., Warshall's and Floyd's Algorithms

introduces a dynamic programming technique that
uses adjacency matrix representation of \( G = (V, E) \)

Warshall's Transitive Closure Algorithm

The transitive closure of a directed graph with \( n \) vertices (nodes) can be
defined as the \( n \)-by-\( n \) boolean matrix \( T = \{t_{ij}\} \), in which the element in
the \( i \)th row and the \( j \)th column is 1 if there exists a nontrivial directed
path from the \( i \)th node to the \( j \)th node; otherwise \( t_{ij} \) is 0.

A non-trivial directed path is a directed path of a positive length. The
boolean matrix representing the graph has 1 in its \( i \)th row and \( j \)th column
iff there is a directed edge from \( i \) to \( j \).

\[
A = \begin{pmatrix}
0 & 1 & 1 \\
0 & 0 & 1 \\
1 & 0 & 0
\end{pmatrix}
\]

Warshall's Transitive Closure Algorithm

Input: Adjacency matrix \( A \) of \( G \) as matrix of 1s and 0's
Output: Transitive Closure or reachability matrix \( R \) of \( G \)

Assumes vertices are numbered 1 to \( |V|, |V| = n \) and there are no edge
weights. Finds a series of boolean matrices \( R(0), ..., R(n) \)

Solution for \( R(n) \):
Define \( r_{ij}^{(k)} \) as the element in the \( i \)th row and \( j \)th column to be 1
iff there is a path between vertices \( i \) and \( j \) using only vertices numbered \( \leq k \).

\( R(0) = A \), original adjacency matrix (1's are direct outgoing edges)
\( R(n) \) the matrix we want to compute

\( R(k) \)'s elements are:

\[
R(0)[i, j] = r_{ij}^{(0)} = r_{ij}^{(0-1)} \lor r_{i\cdot}^{(k-1)} \land r_{\cdot j}^{(k-1)}
\]

Warshall's Algorithm

To create \( R(1) \), we use \( R(0) \).
- There is a 1 in row 3, col 1 and row 1, col 2, (meaning there is
  a directed edge from 3 to 1 and from 1 to 2) so put a 1 in
  position 3,2.
- Also, there is a 1 in row 3, col 1, and row 1, col 3, (meaning
  there is a directed edge from 3 to 1 and from 1 to 3) so put a
  1 in position 3,3

\[
R(0) = A = \begin{pmatrix}
0 & 1 & 1 \\
0 & 0 & 1 \\
1 & 0 & 0
\end{pmatrix}
\]

Matrix \( R(0) \) contains the nodes reachable in one hop

Warshall's Algorithm

Matrix \( R(1) \) contains the nodes reachable in one hop or
in 2 hops on paths that go through vertex 1.

For \( R(2) \), there is no change because 1 can get to 3
through 2 but there is already a direct path between 1 and 3.
**Warshall's Algorithm**

Matrix $R^{(2)}$ contains the nodes reachable in one hop or in multiple hops on paths that go through vertices 1 or 2.

For $R^{(3)}$, there is a 1 in row 1, col 3 and row 3, col 1, so put a 1 in position 1,1. Also, there is a 1 in row 2, col 3 and row 3, col 1, so put a 1 in position 2,1. Also, there is a 1 in row 2, col 3 and row 3, col 2, so put a 1 in position 2,2.

**Floyd's APSP Algorithm**

Input: Adjacency matrix $A$ of graph $G$
Output: Shortest path matrix $D^{(n)}$ and predecessor matrix $P^{(n)}$

Assumes vertices are numbered 1 to $|V|$

Relies on the *Optimal Substructure Property*:
All sub-paths of a shortest path are shortest paths.

**Solution for $D$: (distance matrix)**

Define $D^{(0)}[i,j] = d^{(0)}_{ij}$ as the minimum weight of any path from vertex $i$ to vertex $j$, such that all intermediate vertices are in $\{1, 2, 3, \ldots, k\}$.

$D^{(0)} = A$, original adjacency matrix (only paths are single edges)

$D^{(n)}$ the matrix we want to compute

$D^{(n)}$'s elements are: $D^{(n)}[i,j] = d^{(n)}_{ij} = \min(d^{(k-1)}_{ij}, d^{(k-1)}_{ik} + d^{(k-1)}_{kj})$

**Recursive Solution for $D^{(k)}$**

The only intermediate nodes on the paths from $i$ to $j$, $i$ to $k$, or $k$ to $j$ are in the set of vertices $\{1, 2, 3, \ldots, k-1\}$.

If $k$ is included in shortest $i$ to $j$ path, then a shortest path has been found that includes $k$.

If $k$ is not included in shortest $i$ to $j$ path, then the shortest path still only includes vertices in the set $1\ldots k-1$. 

**Floyd's APSP Algorithm**

Use adjacency matrix $A$ for $G = (V, E)$:

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

**Observation**: When $G$ contains no negative-weight cycles, all shortest paths consist of at most $n - 1$ edges.
**Floyd's APSP Algorithm**

Use adjacency matrix $A$ to keep track of predecessors:

$$
\pi^{(0)}_{ij} = \begin{cases} 
  i & \text{if } i \neq j \text{ and } w(i,j) < \infty \\
  \emptyset & \text{if } i = j \text{ or } w(i,j) = \infty
\end{cases}
$$

$\pi_i$ is predecessor of $j$ on some shortest path from $i$

\[ D(0) = A = \begin{bmatrix}
1 & 2 & 3 & 4 & 5 \\
1 & 0 & 1 & 0 & 0 \\
2 & 0 & 2 & 0 & 2 \\
3 & 0 & 0 & 3 & 3 \\
4 & 4 & 0 & 0 & 4 \\
5 & 0 & 5 & 5 & 0
\end{bmatrix} \]

\[ \Pi^{(0)} = \begin{bmatrix}
\emptyset & 1 & 1 & \emptyset & \emptyset \\
2 & \emptyset & 2 & \emptyset & \emptyset \\
3 & \emptyset & 3 & \emptyset & \emptyset
\end{bmatrix} \]

**Operation of F-APSP Algorithm**

1. $D(k) = A$

2. $\Pi(k) = \Pi(k-1)$

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

4. $\pi_{ij}(k) = \pi_{kj}(k-1)$

5. $\pi_{ij}(k) = \pi_{ij}(k-1)$

6. return $D(n)$

7. print $\pi_{ij}(n)$

8. print $j$

\[ D(0) = A = \begin{bmatrix}
0 & 4 & 11 \\
6 & 0 & 2 \\
3 & \infty & 0 \\
7 & & \\
\end{bmatrix} \]

\[ \Pi^{(0)} = \begin{bmatrix}
\emptyset & 1 & 1 & \emptyset & \emptyset \\
2 & \emptyset & 2 & \emptyset & \emptyset \\
3 & \emptyset & 3 & \emptyset & \emptyset
\end{bmatrix} \]

\[ D(1) = \begin{bmatrix}
0 & 4 & 11 \\
6 & 0 & 2 \\
3 & 7 & 0 \\
7 & & \\
\end{bmatrix} \]

\[ \Pi^{(1)} = \begin{bmatrix}
\emptyset & 1 & 1 & \emptyset & \emptyset \\
2 & \emptyset & 2 & \emptyset & \emptyset \\
3 & 1 & \emptyset & \emptyset
\end{bmatrix} \]
Operation of F-APSP Algorithm

F-APSP Algorithm

F-APSP Algorithm

F-APSP Algorithm

F-APSP Algorithm
Running Time of Floyd's-APSP

Lines 3 – 6: $|V|^3$ time for triply-nested for loops

**Overall running time** $= \Theta(V^3)$

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