Graph Algorithms: Chapter 25 Part 4: All-Pairs Shortest Paths
All-pairs:

- Find shortest path
  - from \( u \)
  - to \( v \)
  - for all \( u, v \in V \)
All-pairs:

- **Given**
  - directed graph $G = (V, E)$
  - weight function $w : E \rightarrow \mathbb{R}$, $|V| = n$

- **Goal**
  - create $n \times n$ matrix of shortest-path distances $\delta(u, v)$
All-pairs:

- Could run BELLMAN-FORD once from each vertex:
  - $O(V^2E)$
  - $O(V^4)$ if graph is *dense* ($E = \Theta(V^2)$)
If no negative-weight edges, could run Dijkstra’s algorithm once from each vertex:

- $O(VE \lg V)$ with binary heap
  - $O(V^3 \lg V)$ if dense
- $O(V^2 \lg V + VE)$ with Fibonacci heap
  - $O(V^3)$ if dense

We’ll see how to do in $O(V^3)$ in all cases

- with no fancy data structure
Shortest paths and matrix multiplication

- Assume $G$ given as adjacency matrix of weights:
  - $W = (w_{ij})$
  - with vertices numbered 1 to $n$

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

- matrix
  - \( D = (d_{ij}) \)
  - \( d_{ij} = \delta(i, j) \)
- Won’t worry about predecessors
  - see book
- Will use dynamic programming first
Optimal substructure:

- Recall:
  - Subpaths of shortest paths are shortest paths
Recursive solution:

- Let $l_{ij}^{(m)} =$ weight of shortest path $i \sim \rightarrow j$ that contains $\leq m$ edges

- $m = 0 \Rightarrow$
  - $l_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j, \\ \infty & \text{if } i \neq j. \end{cases}$
10

- \( m \geq 1 \Rightarrow \)

  \[
  l_{ij}^{(m)} = \min (l_{ij}^{(m-1)}, \min_{l \leq k \leq n} \{l_{ik}^{(m-1)} + w_{kj}\})
  \]

  \((k \text{ is all possible predecessors of } j)\)

  \[
  = \min_{l \leq k \leq n} \{l_{ik}^{(m-1)} + w_{kj}\}
  \]

  \(\text{since } w_{jj} = 0 \text{ for all } j.\)
• Observe that when $m = 1$
  • $\Rightarrow$ must have $l_{ij}^{(1)} = w_{ij}$
• Conceptually, when path restricted to at most 1 edge
  • $\Rightarrow$ weight of shortest path $i \rightsquigarrow j$ must be $w_{ij}$
And the math works out, too

- $l_{ij}^{(1)} = \min_{1 \leq k \leq n} \{l_{ik}^{(0)} + w_{kj}\}$
- $= l_{ii}^{(0)} + w_{ij}$
  
  ($l_{ii}^{(0)}$ is the only non-$\infty$ among $l_{ik}^{(0)}$)

$= w_{ij}$
All simple shortest paths contain \( \leq n - 1 \) edges.

\[ \delta(i, j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n+1)} = \ldots \]
Compute a solution bottom-up:

- Compute $L^{(1)}$, $L^{(2)}$, . . . , $L^{(n-1)}$
- Start with $L^{(1)} = W$
  - since $l_{ij}^{(1)} = w_{ij}$
- Go from $L^{(m-1)}$ to $L^{(m)}$:
**EXTEND**\((L, W, n)\)

- create \(L'\), an \(n \times n\) matrix
- for \(i \leftarrow 1\) to \(n\)
  - do for \(j \leftarrow 1\) to \(n\)
    - do \(l'_{ij} \leftarrow \infty\)
    - for \(k \leftarrow 1\) to \(n\)
      - do \(l'_{ij} \leftarrow \min(l'_{ij}, l_{ik} + w_{kj})\)
  - return \(L'\)
Compute each $L^{(m)}$:

- SLOW-APSP($W, n$)
- $L^{(1)} \leftarrow W$
- for $m \leftarrow 2$ to $n - 1$
  - do $L^{(m)} \leftarrow$ EXTEND($L^{(m - 1)}, W, n$)
- return $L^{(n - 1)}$
Time:

- EXTEND: $\Theta(n^3)$
- SLOW-APSP: $\Theta(n^4)$
Observation:

- EXTEND is like matrix multiplication:
  - \( L \rightarrow A \)
  - \( W \rightarrow B \)
  - \( L' \rightarrow C \)
  - \( \text{min} \rightarrow + \)
  - \( + \rightarrow \cdot \)
  - \( \infty \rightarrow 0 \)
create $C$, $n \times n$ matrix

- for $i \leftarrow 1$ to $n$
  - do for $j \leftarrow 1$ to $n$
    - do $c_{ij} \leftarrow 0$
      - for $k \leftarrow 1$ to $n$
        - do $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$
So, can view EXTEND as just like matrix multiplication!

Why do we care?
Because

- Goal: compute $L^{(n-1)}$ as fast as can
- Don’t need to compute *all* intermediates
  - $L^{(1)}, L^{(2)}, \ldots, L^{(n-2)}$
- Suppose had matrix $A$ and wanted to compute $A^{n-1}$
  - (like calling EXTEND $n - 1$ times)
- Could compute $A, A^2, A^4, A^8, \ldots$
If knew $A^m = A^{n-1}$ for all $m \geq n - 1$

could just finish with $A^r$,

where $r$ is smallest power of 2

that’s $\geq n - 1$

($r = 2^\lceil \lg(n-1) \rceil$)
FASTER-APSP\( (W, n) \)

- \( L^{(1)} \leftarrow W \)
- \( m \leftarrow 1 \)
- \( \textbf{while } m < n - 1 \)
  - \( \textbf{do } L^{(2m)} \leftarrow \text{EXTEND}(L^{(m)}, L^{(m)}, n) \)
  - \( m \leftarrow 2m \)
- \( \textbf{return } L^{(m)} \)
- OK to overshoot
- since products don’t change after $L^{(n-1)}$
- **Time:** $\Theta(n^3 \lg n)$
Floyd-Warshall algorithm

- Different dynamic-programming approach
- For path $p = v_1, v_2, \ldots, v_l$
  - *intermediate vertex*
    - = any vertex of $p$ other than
      - $v_1$ or $v_l$
Let $d_{ij}^{(k)}$ = shortest-path weight of
- any path $i \sim \rightarrow j$
- with all intermediate vertices
  - in \{1, 2, \ldots, k\}
Consider shortest path $i \sim p \rightarrow j$
- with all intermediate vertices
  - in \{1, 2, \ldots, k\}:
• If $k$ not intermediate vertex
  • $\Rightarrow$ all intermediate vertices of $p$ are
    • in $\{1, 2, \ldots, k - 1\}$
• • If $k$ is intermediate vertex:
  
  $i \rightarrow p_1 \rightarrow k \rightarrow p_2 \rightarrow j$

  all intermediate vertices in $\{1, 2, \ldots, k-1\}$
Recursive formulation

- \( d_{ij}^{(k)} = \)
  - \( w_{ij} \) if \( k = 0 \)
  - \( \min (d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) \) if \( k \geq 1 \)
- \( d_{ij}^{(0)} = w_{ij} \)
  - because can’t have intermediate vertices
  - \( \rightarrow \leq 1 \) edge
- Want \( D(n) = (d_{ij}^{(n)}) \)
- since all vertices numbered \( \leq n \)
Compute bottom-up

- Compute in increasing order of $k$: 
FLOYD-WARSHALL($W, n$)

- $D(0) \leftarrow W$
- for $k \leftarrow 1$ to $n$
  - do for $i \leftarrow 1$ to $n$
    - do for $j \leftarrow 1$ to $n$
      - do $d_{ij}^{(k)} \leftarrow \min (d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
  - return $D(n)$
Can drop superscripts
  (See Exercise 25.2-4 in text.)

Time: $\Theta(n^3)$
Transitive closure

- Given $G = (V, E)$, directed.
- Compute $G^* = (V, E^*)$
  - $E^* = \{(i, j) : \text{there is a path } i \rightarrow j \text{ in } G\}$
- Could
  - assign weight of 1 to each edge
  - then run FLOYD-WARSHALL
    - If $d_{ij} < n$, then there is a path $i \rightarrow j$
    - Otherwise, $d_{ij} = \infty$ and there is no path
Simpler way:

- Substitute other values and operators in FLOYD-WARSHALL
  - Use unweighted adjacency matrix
  - $\min \rightarrow \lor$ (OR)
  - $+ \rightarrow \land$ (AND)
\[ t_{ij}^{(k)} = \]
- 1 if there is path \( i \leadsto j \)
- with all intermediate vertices in \( \{1, 2, \ldots, k\} \)
- 0 otherwise
\[ t^{(0)}_{ij} = \begin{cases} 0 & \text{if } i \neq j \text{ and } (i, j) \notin E \\ 1 & \text{if } i = j \text{ or } (i, j) \in E \end{cases} \]

\[ t^{(k)}_{ij} = \begin{cases} t^{(k)}_{ij} \lor \left( t^{(k-1)}_{ik} \land t^{(k-1)}_{kj} \right) & \end{cases} \]
TRANSITIVE-CLOSURE($E, n$)

- for $i \leftarrow 1$ to $n$
  - do for $j \leftarrow 1$ to $n$
    - do if $i = j$ or $(i, j) \in E[G]$
      - then $t_{ij}^{(0)} \leftarrow 1$
      - else $t_{ij}^{(0)} \leftarrow 0$
  - for $k \leftarrow 1$ to $n$
    - do for $i \leftarrow 1$ to $n$
      - do for $j \leftarrow 1$ to $n$
        - do $t_{ij}^{(k)} \leftarrow t_{ij}^{(k)} \lor (t_{ik}^{(k-1)} \land t_{kj}^{(k-1)})$
    - return $T^{(n)}$
\[ \text{Time: } \Theta(n^3) \]
- but simpler operations than FLOYD-WARSHALL
Johnson’s algorithm

- **Idea:** If graph sparse, pays to run Dijkstra’s alg once from each vertex
- If use Fibonacci heap for priority queue
  - running time down to $O(V^2 \lg V + V E)$
  - = better than FLOYD-WARSHALL’s $\Theta(V^3)$ time
    - If $E = o(V^2)$
But Dijkstra’s alg requires
  all edge weights nonnegative
Donald Johnson :
  how to make equivalent graph
  w/ all edge weights $\geq 0$
Reweighting

- **Compute new weight function** $w^\wedge$ such that
  1. For all $u, v \in V$, $p = \text{shortest path } u \rightarrow v$ using $w$ \iff $p = \text{shortest path } u \rightarrow v$ using $w^\wedge$
  2. For all $(u, v) \in E$, $w^\wedge (u, v) \geq 0$
(1) : suffices to find shortest paths with $w^\wedge$
(2) : can do so by running Dijkstra’s alg from each vertex
How to come up with $w^\wedge$?
Lemma shows it’s easy to get property (1):
Lemma (Reweighting doesn’t change shortest paths)

- Given
  - directed, weighted graph $G = (V, E)$
  - $w: E \rightarrow \mathbb{R}$
- Let $h$ be any function s.t. $h: V \rightarrow \mathbb{R}$
- For all $(u, v) \in E$, define
  - $w^\bullet (u, v) = w(u, v) + h(u) - h(v)$
Let \( p = \langle v_0, v_1, \ldots, v_k \rangle \) = any path \( v_0 \rightarrow v_k \)

Then, \( p = \) shortest path \( v_0 \rightarrow v_k \) with \( w \)

\( \iff \) \( p = \) shortest path \( v_0 \rightarrow v_k \) with \( w^\wedge \)

(Formally, \( w(p) = \delta(v_0, v_k) \iff w^\wedge = \delta^\wedge(v_0, v_k) \), where \( \delta^\wedge = \) shortest-path weight with \( w^\wedge \).

Also, \( G \) has negative-weight cycle w/ \( w \)

\( \iff \) \( G \) has negative-weight cycle w/ \( w^\wedge \)
Proof

First, show \( w^\wedge(p) = w(p) + h(v_0) - h(v_k) \):

\[
\begin{align*}
\hat{w}(p) &= \sum_{i=1}^{k} \hat{w}(v_{i-1}, v_i) \\
&= \sum_{i=1}^{k} (w(v_{i-1}, v_i) + h(v_{i-1}) - h(v_i)) \\
&= \sum_{i=1}^{k} w(v_{i-1}, v_i) + h(v_0) - h(v_k) \quad \text{(sum telescopes)} \\
&= w(p) + h(v_0) - h(v_k).
\end{align*}
\]
- any path \(v_0 \sim p \rightarrow v_k\) has
  - \(w^\wedge(p) = w(p) + h(v_0) - h(v_k)\)
- Since \(h(v_0)\) and \(h(v_k)\) don’t depend on path from \(v_0\) to \(v_k\)
  - if one path \(v_0 \sim p \rightarrow v_k\) shorter than another with \(w\)
  - also shorter with \(w^\wedge\)
Now show $\exists$ negative-weight cycle with $w$ $\iff$ $\exists$ negative-weight cycle with $w^\wedge$:

Let cycle $c = \langle v_0, v_1, \ldots, v_k \rangle$

- $v_0 = v_k$

$\implies$

$w^\wedge(c) = w(c) + h(v_0) - h(v_k) = w(c)$
\[ \Rightarrow \text{ } c \text{ has negative-weight cycle w/ } w \iff \text{ it has negative-weight cycle w/ } w^\wedge \]

\[ \square \text{ (lemma)} \]
So, now to get property (2)
just need to come up with function

- $h : V \rightarrow \mathbb{R}$

s.t. when compute

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

- $\Rightarrow \geq 0$
Do what did for difference constraints:

- $G' = (V', E')$
- $V = V \cup \{s\}$, $s = \text{new vertex}$
- $E = E \cup \{(s, v) : v \in V\}$.
- $w(s, v) = 0$ for all $v \in V$
- No edges enter s
- $\Rightarrow G'$ has same set of cycles as $G$
- In particular,
  - $G'$ has negative-weight cycle
  - $\iff G$ does
- Define $h(v) = \delta(s, v)$ for all $v \in V$
Claim

- \( w^\delta (u, v) = w (u, v) + h(u) - h(v) \geq 0 \)

**Proof** By triangle inequality,

- \( \delta(s, v) \leq \delta(s, u) + w(u, v) \)
- \( h(v) \leq h(u) + w(u, v) \)
- \( \Rightarrow w(u, v) + h(u) - h(v) \geq 0 \)
- \( \square \) (claim)
Johnson’s algorithm

- form $G'$
- run BELLMAN-FORD on $G'$ to compute $\delta(s, v)$ for all $v \in V$
  - if BELLMAN-FORD returns FALSE
    - then $G$ has a negative-weight cycle
  - else
    - compute $w'(u, v) = w(u, v) + \delta(s, u) - \delta(s, v)$
      - for all $(u, v) \in E$
    - Cont. ....
for each vertex $u \in V$
- do run Dijkstra’s alg from $u$ using weight function $w^\wedge$
  - to compute $\delta^\wedge(u, v)$ for all $v \in V$

for each vertex $v \in V$
- do // Compute entry $d_{uv}$ in matrix $D$
- $d_{uv} = \delta^\wedge(u, v) + \delta(s, v) - \delta(s, u)$
- because if $p$ is a path $u \rightarrow v$,
  - then $w^\wedge(p) = w(p) + h(v_0) - h(v_k)$
Johnson’s algorithm

form $G'$
run Bellman-Ford on $G'$ to compute $\delta(s, v)$ for all $v \in V$
if Bellman-Ford returns FALSE
then $G$ has a negative-weight cycle
else
compute $\hat{w}(u, v) = w(u, v) + \delta(s, u) - \delta(s, v)$ for all $(u, v) \in E$
for each vertex $u \in V$
do run Dijkstra’s algorithm from $u$ using weight function $\hat{w}$
to compute $\hat{\delta}(u, v)$ for all $v \in V$
for each vertex $v \in V$
do $\triangleright$ Compute entry $d_{uv}$ in matrix $D$
\[d_{uv} = \hat{\delta}(u, v) + \delta(s, v) - \delta(s, u)\]
because if $p$ is a path $u \leadsto v$,
then $\hat{w}(p) = w(p) + h(u) - h(v)$
Time:

- $\Theta(V + E)$ to compute $G$
- $O(VE)$ to run BELLMAN-FORD
- $\Theta(E)$ to compute $w^\wedge$
- $O(V^2 \log V + VE)$ to run Dijkstra’s alg $|V|$ times (using Fibonacci heap)
- $\Theta(V^2)$ to compute $D$ matrix
- **Total:** $O(V^2 \log V + VE)$
Table 21.4 Costs of shortest-paths algorithms

This table summarizes the cost (worst-case running time) of various shortest-paths algorithms considered in this chapter. The worst-case bounds marked as conservative may not be useful in predicting performance on real networks, particularly the Bellman-Ford algorithm, which typically runs in linear time.

<table>
<thead>
<tr>
<th>weight constraint</th>
<th>algorithm</th>
<th>cost</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>single-source</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nonnegative</td>
<td>Dijkstra</td>
<td>$V^2$</td>
<td>optimal (dense networks)</td>
</tr>
<tr>
<td>nonnegative</td>
<td>Dijkstra (PFS)</td>
<td>$E \log V$</td>
<td>conservative bound</td>
</tr>
<tr>
<td>acyclic</td>
<td>source queue</td>
<td>$E$</td>
<td>optimal</td>
</tr>
<tr>
<td>no negative cycles</td>
<td>Bellman–Ford</td>
<td>$VE$</td>
<td>room for improvement?</td>
</tr>
<tr>
<td>none</td>
<td>open</td>
<td>?</td>
<td>NP-hard</td>
</tr>
<tr>
<td>all-pairs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nonnegative</td>
<td>Floyd</td>
<td>$V^3$</td>
<td>same for all networks</td>
</tr>
<tr>
<td>nonnegative</td>
<td>Dijkstra (PFS)</td>
<td>$VE \log V$</td>
<td>conservative bound</td>
</tr>
<tr>
<td>acyclic</td>
<td>DFS</td>
<td>$VE$</td>
<td>same for all networks</td>
</tr>
<tr>
<td>no negative cycles</td>
<td>Floyd</td>
<td>$V^3$</td>
<td>same for all networks</td>
</tr>
<tr>
<td>no negative cycles</td>
<td>Johnson</td>
<td>$VE \log V$</td>
<td>conservative bound</td>
</tr>
<tr>
<td>none</td>
<td>open</td>
<td>?</td>
<td>NP-hard</td>
</tr>
</tbody>
</table>

source: Sedgewick, Graph Algorithms
Transitive Closure (Matrix): Unweighted, Directed Graph

Transitive Closure concepts will be useful for All-Pairs Shortest Path calculation in directed, weighted graphs.

“self-loops” added for algorithmic purposes

Transitive Closure Graph contains edge (u,v) if there exists a directed path in G from u to v.

Source: Sedgewick, Graph Algorithms
Transitive Closure (Matrix)

Boolean Matrix Product: and, or replace *, +

source: Sedgewick, Graph Algorithms
Algorithm 1: Find $\Gamma, \Gamma^2, \Gamma^3, ..., \Gamma^{|V|-1}$  
Time: $O(|V|^4)$

Algorithm 2: Find $\Gamma, \Gamma^2, \Gamma^4, ..., \Gamma^{|V|}$  
Time: $O(|V|^3\lg|V|)$

Algorithm 3: [Warshall]  
Time: $O(|V|^3)$

for $i \leftarrow 0$ to $|V|-1$
for $s \leftarrow 0$ to $|V|-1$
for $t \leftarrow 0$ to $|V|-1$
if $\Gamma[s][i]$ and $\Gamma[i][t]$  
then $\Gamma[s][t] \leftarrow 1$

source: Sedgewick, Graph Algorithms
Transitive Closure (Matrix)

Warshall

source: Sedgewick, Graph Algorithms

good for dense graphs
Transitive Closure (Matrix)

Correctness by Induction on $i$:

**Inductive Hypothesis**: $i$th iteration of loop sets $\Gamma[s][t]$ to 1 iff there’s a directed path from $s$ to $t$ with (internal) indices at most $i$.

**Inductive Step for $i+1$ (sketch)**: 2 cases for path $<s…t>$

- Internal indices at most $i$
  - Covered by inductive hypothesis in prior iteration so $\Gamma[s][t]$ already set to 1

- An internal index exceeds $i$ ($= i+1$)
  - $\Gamma[s][i+1], \Gamma[i+1][t]$ set in a prior iteration so $\Gamma[s][t]$ set to 1 in current iteration

Source: Sedgewick, Graph Algorithms