Lecture Notes for Chapter 25:
All-Pairs Shortest Paths

Chapter 25 overview

Given a directed graph $G = (V, E)$, weight function $w : E \to \mathbb{R}, |V| = n$.
Goal: create an $n \times n$ matrix of shortest-path distances $\delta(u, v)$.
Could run BELLMAN-FORD once from each vertex:

- $O(V^2 E)$—which is $O(V^4)$ if the 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 that $G$ is 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 is matrix $D = (d_{ij})$, where $d_{ij} = \delta(i, j)$. Won’t worry about predecessors—see book.

Will use dynamic programming at first.

**Optimal substructure:** Recall: subpaths of shortest paths are shortest paths.

**Recursive solution:** Let $l_{ij}^{(m)}$ = weight of shortest path $i \leadsto j$ that contains $\leq m$ edges.

- $m = 0$
  - there is a shortest path $i \leadsto j$ with $\leq m$ edges if and only if $i = j$
  - $l_{ij}^{(0)} = \begin{cases} 
0 & \text{if } i = j \\
\infty & \text{if } i \neq j.
\end{cases}$
• \( m \geq 1 \)
  \[
  l_{ij}^{(m)} = \min \left( l_{ij}^{(m-1)}, \min_{1 \leq k \leq n} \left\{ l_{ik}^{(m-1)} + w_{kj} \right\} \right)
  \]
  \( (k \text{ is all possible predecessors of } j) \)
  \[
  = \min_{1 \leq k \leq n} \left\{ l_{ij}^{(m-1)} + w_{kj} \right\}
  \]
  since \( w_{jj} = 0 \) for all \( j \).

• Observe that when \( m = 1 \), must have \( l_{ij}^{(1)} = w_{ij} \).
  Conceptually, when the path is restricted to at most 1 edge, the weight of the shortest path \( i \leadsto j \) must be \( w_{ij} \).
  And the math works out, too:
  \[
  l_{ij}^{(1)} = \min_{1 \leq k \leq n} \left\{ l_{ik}^{(0)} + w_{kj} \right\}
  \]
  \( (l_{ij}^{(0)} \text{ is the only non-}\infty \text{ among } l_{ik}^{(0)}) \)
  \[
  = l_{ij}^{(0)} + w_{ij}
  \]
  All simple shortest paths contain \( \leq n - 1 \) edges
  \[
  \Rightarrow \delta(i, j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n+1)} = \ldots
  \]

**Compute a solution bottom-up:** Compute \( L^{(1)}, L^{(2)}, \ldots, L^{(n-1)} \).

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 \( L^{(m)} \leftarrow \text{EXTEND}(L^{(m-1)}, W, n) \)
return \( L^{(n-1)} \)

**Time:**

• \( \text{EXTEND: } \Theta(n^3) \).
• \( \text{SLOW-APSP: } \Theta(n^4) \).
**Observation:** EXTEND is like matrix multiplication:

\[
\begin{align*}
L & \rightarrow A \\
W & \rightarrow B \\
L' & \rightarrow C \\
\min & \rightarrow + \\
+ & \rightarrow . \\
\infty & \rightarrow 0
\end{align*}
\]

create \( C \), an \( n \times n \) matrix

for \( i \leftarrow 1 \) to \( n \)
   for \( j \leftarrow 1 \) to \( n \)
      \( c_{ij} \leftarrow 0 \)
      for \( k \leftarrow 1 \) to \( n \)
         \( c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj} \)

So, we can view EXTEND as just like matrix multiplication!

Why do we care?

Because our goal is to compute \( L^{(n-1)} \) as fast as we can. Don’t need to compute all the intermediate \( L^{(1)}, L^{(2)}, L^{(3)}, \ldots, L^{(n-2)} \).

Suppose we had a matrix \( A \) and we wanted to compute \( A^{n-1} \) (like calling EXTEND \( n-1 \) times).

Could compute \( A, A^2, A^4, A^8, \ldots \)

If we knew \( A^m = A^{n-1} \) for all \( m \geq n - 1 \), could just finish with \( A^r \), where \( r \) is the smallest power of 2 that’s \( \geq n - 1 \). (\( r = 2^{\lceil \log (n-1) \rceil} \))

**FASTER-APSP(\( W, n \))**

\[
\begin{align*}
L^{(1)} & \leftarrow W \\
m & \leftarrow 1 \\
while m < n - 1 \\
   do L^{(2m)} & \leftarrow \text{EXTEND}(L^{(m)}, L^{(m)}, n) \\
   m & \leftarrow 2m \\
return L^{(m)}
\end{align*}
\]

OK to overshoot, since products don’t change after \( L^{(n-1)} \).

**Time:** \( \Theta(n^3 \log n) \).

---

**Floyd-Warshall algorithm**

A different dynamic-programming approach.

For path \( p = (v_1, v_2, \ldots, v_l) \), an **intermediate vertex** is any vertex of \( p \) other than \( v_1 \) or \( v_l \).

Let \( d_{ij}^{(k)} \) = shortest-path weight of any path \( i \sim j \) with all intermediate vertices in \( \{1, 2, \ldots, k\} \).

Consider a shortest path \( i \rightarrow^p j \) with all intermediate vertices in \( \{1, 2, \ldots, k\} \):
• If $k$ is not an intermediate vertex, then all intermediate vertices of $p$ are in $\{1, 2, \ldots, k - 1\}$.  
• If $k$ is an intermediate vertex:

\[
\begin{array}{c}
\text{all intermediate vertices in } \{1, 2, \ldots, k - 1\}
\end{array}
\]

**Recursive formulation**

\[
d^{(k)}_{ij} = \begin{cases} 
  w_{ij} & \text{if } k = 0, \\
  \min \left( d^{(k-1)}_{ij}, d^{(k-1)}_{ik} + d^{(k-1)}_{kj} \right) & \text{if } k \geq 1.
\end{cases}
\]

(Have $d^{(0)}_{ij} = w_{ij}$ because can’t have intermediate vertices $\Rightarrow \leq 1$ edge.)  
Want $D^{(n)} = (d^{(n)}_{ij})$, since all vertices numbered $\leq n$.  

**Compute bottom-up**

Compute in increasing order of $k$:

\[
\text{FLOYD-WARSHALL}(W, n) \\
D^{(0)} \leftarrow W \\
\text{for } k \leftarrow 1 \text{ to } n \text{ do for } i \leftarrow 1 \text{ to } n \text{ do for } j \leftarrow 1 \text{ to } n \text{ do } d^{(k)}_{ij} \leftarrow \min \left( d^{(k-1)}_{ij}, d^{(k-1)}_{ik} + d^{(k-1)}_{kj} \right)
\]

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 $G^* = (V, E^*)$.  
• $E^* = \{(i, j) : \text{there is a path } i \rightsquigarrow j \text{ in } G\}$.  
Could assign weight of 1 to each edge, then run FLOYD-WARSHALL.  
• If $d_{ij} < n$, then there is a path $i \rightsquigarrow j$.  
• 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)
- \( t^{(k)}_{ij} = \begin{cases} 1 & \text{if there is path } i \sim j \text{ with all intermediate vertices in } \{1, 2, \ldots, k\}, \\ 0 & \text{otherwise}. \end{cases} \)
- \( t^{(0)}_{ij} = \begin{cases} 0 & \text{if } i \neq j \text{ and } (i, j) \notin E, \\ 1 & \text{if } i = j \text{ or } (i, j) \in E. \end{cases} \)
- \( t^{(k)}_{ij} = t^{(k-1)}_{ij} \lor (t^{(k-1)}_{ik} \land t^{(k-1)}_{kj}). \)

**TRANSITIVE-CLOSURE** \((E, n)\)

```plaintext
for i ← 1 to n
    do for j ← 1 to n
        do if i = j or (i, j) ∈ E[\(G\)]
            then \( t^{(0)}_{ij} \leftarrow 1 \)
                else \( t^{(0)}_{ij} \leftarrow 0 \)

for k ← 1 to n
    do for i ← 1 to n
        do for j ← 1 to n
            do \( t^{(k)}_{ij} \leftarrow t^{(k-1)}_{ij} \lor (t^{(k-1)}_{ik} \land t^{(k-1)}_{kj}) \)

return \( T^{(n)} \)
```

**Time:** \( \Theta(n^3) \), but simpler operations than FLOYD-WARSHALL.

**Johnson’s algorithm**

**Idea:** If the graph is sparse, it pays to run Dijkstra’s algorithm once from each vertex.

If we use a Fibonacci heap for the priority queue, the running time is down to \( O(V^2 \lg V + VE) \), which is better than FLOYD-WARSHALL’s \( \Theta(V^3) \) time if \( E = o(V^2) \).

But Dijkstra’s algorithm requires that all edge weights be nonnegative.
Donald Johnson figured out how to make an equivalent graph that does have all edge weights \( \geq 0 \).

**Reweighting**

Compute a new weight function \( \hat{w} \) such that

1. For all \( u, v \in V, p \) is a shortest path \( u \sim v \) using \( w \) if and only if \( p \) is a shortest path \( u \sim v \) using \( \hat{w} \).
2. For all \((u, v) \in E, \hat{w}(u, v) \geq 0\).
Property (1) says that it suffices to find shortest paths with \( \hat{w} \). Property (2) says we can do so by running Dijkstra’s algorithm from each vertex.

How to come up with \( \hat{w} \)?

Lemma shows it’s easy to get property (1):

**Lemma (Reweighting doesn’t change shortest paths)**

Given a directed, weighted graph \( G = (V, E), w : E \to \mathbb{R} \). Let \( h \) be any function such that \( h : V \to \mathbb{R} \). For all \((u, v) \in E\), define

\[
\hat{w}(u, v) = w(u, v) + h(u) - h(v) .
\]

Let \( p = \langle v_0, v_1, \ldots, v_k \rangle \) be any path \( v_0 \sim v_k \).

Then, \( p \) is a shortest path \( v_0 \sim v_k \) with \( w \) if and only if \( p \) is a shortest path \( v_0 \sim v_k \) with \( \hat{w} \). (Formally, \( w(p) = \hat{\delta}(v_0, v_k) \) if and only if \( \hat{w} = \hat{\delta}(v_0, v_k) \), where \( \hat{\delta} \) is the shortest-path weight with \( \hat{w} \).)

Also, \( G \) has a negative-weight cycle with \( w \) if and only if \( G \) has a negative-weight cycle with \( \hat{w} \).

**Proof** First, we’ll show that \( \hat{w}(p) = w(p) + h(v_0) - h(v_k) \):

\[
\hat{w}(p) = \sum_{i=1}^{k} \hat{w}(v_{i-1}, v_i) = \sum_{i=1}^{k} (w(v_{i-1}, v_i) + h(v_{i-1}) - h(v_i)) = \sum_{i=1}^{k} w(v_{i-1}, v_i) + h(v_0) - h(v_k) \text{ (sum telescopes)} = w(p) + h(v_0) - h(v_k) .
\]

Therefore, any path \( v_0 \xrightarrow{p} v_k \) has \( \hat{w}(p) = w(p) + h(v_0) - h(v_k) \). Since \( h(v_0) \) and \( h(v_k) \) don’t depend on the path from \( v_0 \) to \( v_k \), if one path \( v_0 \sim v_k \) is shorter than another with \( w \), it’s also shorter with \( \hat{w} \).

Now show there exists a negative-weight cycle with \( w \) if and only if there exists a negative-weight cycle with \( \hat{w} \):

- Let cycle \( c = \langle v_0, v_1, \ldots, v_k \rangle \), where \( v_0 = v_k \).
- Then

\[
\hat{w}(c) = w(c) + h(v_0) - h(v_k) = w(c) \text{ (since } v_0 = v_k) .
\]

Therefore, \( c \) has a negative-weight cycle with \( w \) if and only if it has a negative-weight cycle with \( \hat{w} \).** (lemma)

So, now to get property (2), we just need to come up with a function \( h : V \to \mathbb{R} \) such that when we compute \( \hat{w}(u, v) = w(u, v) + h(u) - h(v) \), it’s \( \geq 0 \).

Do what we did for difference constraints:

- \( G' = (V', E') \)
Lecture Notes for Chapter 25: All-Pairs Shortest Paths 25-7

- $V' = V \cup \{s\}$, where $s$ is a new vertex.
- $E' = E \cup \{(s, v) : v \in V\}$.
- $w(s, v) = 0$ for all $v \in V$.
- Since no edges enter $s$, $G'$ has the same set of cycles as $G$. In particular, $G$ has a negative-weight cycle if and only if $G$ does.

Define $h(v) = \delta(s, v)$ for all $v \in V$.

**Claim**
\[ \hat{w}(u, v) = w(u, v) + h(u) - h(v) \geq 0. \]

**Proof** By the triangle inequality,
\[
\delta(s, v) \leq \delta(s, u) + w(u, v)
\]
\[
h(v) \leq h(u) + w(u, v).
\]
Therefore, $w(u, v) + h(u) - h(v) \geq 0$. \hfill ■

**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 \[ \hat{\delta}(u, v) = \hat{\delta}(u, v) + \hat{\delta}(s, v) - \hat{\delta}(s, u) \]
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 $G'$.
- $O(V E)$ to run BELLMAN-FORD.
- $\Theta(E)$ to compute $\hat{w}$.
- $O(V^2 \lg V + VE)$ to run Dijkstra’s algorithm $|V|$ times (using Fibonacci heap).
- $\Theta(V^2)$ to compute $D$ matrix.

**Total:** $O(V^2 \lg V + VE)$. 