Lecture: All Pairs Shortest Paths

- Definition
- Optimal Substructure
- Dynamic Programming Formulation
- Floyd-Warshall Algorithm

These notes are hand-written, unedited and sketchy. They are primarily used for, and based on my lectures.

If you find any bug, impreciseness, or a rare poor-/mis-interpretation of facts, please let me know. I will be grateful for any additional comments you have that are intended to make the quality of the notes better.

Please note that I will provide my hand-written lecture notes only for a subset of my lectures, not for all lectures. Therefore, it is your responsibility to attend all the lectures, take notes regularly, and ask me and/or the TAs if you have any questions.

Thank you!
--Chittu
All-Pairs Shortest Paths (APSPs)

Given: A directed graph $G = (V, E)$, weight function $w: E \rightarrow \mathbb{R}$.

Wlog, $V = \{1, 2, \ldots, n\}$.

Goal: $n \times n$ matrix $D = (d_{ij})$ of shortest path distances, so that $d_{ij} = 0$ if vertices $i$ and $j$.

First Idea:
- Negative edge weights:
  - Run Bellman-Ford once for each vertex.
  - Time: $O(V^2 E)$ which is $O(V^4)$ if $G$ is dense i.e., $E = O(V^2)$.
- Edge weights are positive:
  - Run Dijkstra once for each vertex.
  - Time: $O(VE \log V)$, and for dense graphs $O(V^3 \log V)$, when binary heap is used.
  - $O(V^2 \log V + VE)$, and for dense graphs $O(V^3)$, when Fibonacci heap is used.

We will see that we can compute APSPs in $O(V^3)$ time without using any fancy datastructures.

Second Idea: Use dynamic programming
- $W = (w_{ij})_{n \times n}$ is the adjacency matrix representation of $G$.
- $w_{ij} = \begin{cases} 
0 & \text{if } i = j \\
\text{weight of } (i, j) & \text{if } i \neq j \text{ and } (i, j) \in E \\
\infty & \text{if } i \neq j \text{ and } (i, j) \notin E 
\end{cases}$

Optimal Substructure:
Subpaths of shortest paths are shortest paths. We proved this earlier.
Recursive Formulation:

Define $l_{ij}^{(m)}$ = weight of a shortest path from $i$ to $j$ with $\leq m$ edges

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

- $m > 1 \Rightarrow l_{ij}^{(m)} = \min \left\{ l_{ij}^{(m-1)}, \min_{1 \leq k \leq n} \left\{ l_{ik}^{(m-1)} + w_{kj} \right\} \right\}$

- Observation: $l_{ij}^{(1)} = w_{ij}$. That is, if the SP has length 1, then it must be the weight of the edge $(i, j)$.

- All simple shortest paths contain $\leq n-1$ edges

$\Rightarrow \delta(i,j) = l_{ij}^{(n-1)} = l_{ij}^{(0)} = l_{ij}^{(n+1)} = \ldots$

- Computing $l_{ij}^{(m)}$

$\begin{cases} l_{ij}^{(0)} = \min_{1 \leq k \leq n} \{ l_{ik}^{(m-1)} + w_{kj} \} \\ \text{for } k \leftarrow 1 \text{ to } n \\ \text{if } d_{ij} > d_{ik} + a_{kj} \}

\begin{cases} d_{ij} \leftarrow d_{ik} + a_{kj} \end{cases}$

Drawing Parallel on Matrix Multiplication:

- Matrix multiplication of $n \times n$ matrices: $C = A \times B$

$\begin{cases} c_{ij} \leftarrow 0 \\ \text{for } k \leftarrow 1 \text{ to } n \\ c_{ij} \leftarrow c_{ij} + a_{ik} \times b_{kj} \}

\begin{cases} \text{in } \circ \text{ replace } + \text{ by } \min \text{ and } \times \text{ by } + \text{ to get } A. \end{cases}$
• So can be implemented similar to matrix multiplication.

• Our goal is to compute \( L^{(n-1)} = (L_{ij}^{(n-1)})_{nn} \).

• \( L^{(0)} = W \). We know this as initial condition (matrix).

• How do we compute \( L^{(n-1)} \)?

  **Method 1:** Iteratively multiply \( L^{(0)} \) \( n \) times. \( O(n^4) \).

  **Method 2:** We can do better!

  Do repeated squaring: \( L^{(1)}, L^{(2)}, L^{(4)}, \ldots, L^{(2^t)} \) such that 
  \( 2^{t-1} < n-1 \leq 2^t \).

  Why does this work?

  \( \therefore \delta_{ij} = L_{ij}^{(n-1)} = L_{ij}^{(n)} = L_{ij}^{(n+1)} = \ldots \)

  So we don’t care if \( 2^t \) exceeds \( n-1 \).

  Time for method 2: \( O(n^3 \log n) \).

• Exercise: Write the pseudocode for Method 2 above.

**Perpetual Question:** Can we do better?

• Smart dynamic programming! Floyd-Warshall Algorithm!
Floyd-Warshall Algorithm:

- **Smart dynamic programming!**

- Define $d^{(k)}_{ij} =$ shortest path weight for any path $i \rightarrow j$
  with all intermediate vertices in $\{1, 2, \ldots, k\}$.

- Note intermediate vertices do not contain the vertex $i \rightarrow j$.

- For the path $i \rightarrow j$ with intermediate vertices in $\{1, 2, \ldots, k\}$:
  - If $k$ is not an intermediate vertex, then the intermediate vertices are in $\{1, 2, \ldots, k-1\}$.
  
  
  - If $k$ is an intermediate vertex:

  
  
  all intermediate vertices are in $\{1, 2, \ldots, k-1\}$.

- **Recursive Formulation**:

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

\[d^{(0)}_{ij} = w_{ij}\]

Goal is to compute $D = (d^{(n)}_{ij})$. 

Meeting notes Page 5
Computing APSPs Bottom-Up: Pseudocode for Floyd-Warshall Algorithm

\text{FLOYD-WARSHALL}(W, n)

1. \(J^0 \leftarrow W\)
2. \text{for } k \leftarrow 1 \text{ to } n
3. \text{Let } J^{(k)} = (d_{ij}^{(k)}) \text{ be a new } n \times n \text{ matrix}
4. \text{for } i \leftarrow 1 \text{ to } n
5. \text{for } j \leftarrow 1 \text{ to } n
6. \quad d_{ij}^{(k)} \leftarrow \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}
7. \text{return } J^n

- You can drop the superscripts.
- Easy to code.
- No fancy data structures used.
- Efficient with Time = \(\Theta(n^3)\).

An Application: Computing Transitive Closure of a Directed Graph

- Given \(G = (V, E)\), asked to compute \(G^* = (V, E^*)\).
  \[E^* = \{ (i,j) \mid i \rightarrow i \text{ in } G \} \]
- \text{First Method: Use FLOYD-WARSHALL on } G \text{ with all } w_{ij} = 1.
  \text{If } d_{ij} < n \text{ then } i \rightarrow i \text{. Otherwise, } d_{ij} = \infty \text{ is no path!}
- \text{Second Method: Use unweighted adjacency matrix. min} \rightarrow V. \ + \rightarrow \Lambda.
  - \(t_{ij}^{(k)} = \begin{cases} 1 & \text{if } (i,j) \text{ with intermediate vertices in } \{1, \ldots, k\} \\ 0 & \text{Otherwise.} \end{cases}\)
  - \(t_{ij}^{(0)} = \begin{cases} 0 & \text{if } i \neq j \text{ and } (i,j) \notin E \\ 1 & \text{if } i = j \text{ or } (i,j) \in E \end{cases}\)
  - \(t_{ij}^{(k)} = t_{ij}^{(k-1)} \lor (t_{ik}^{(k-1)} \land t_{kj}^{(k-1)})\). Simpler operations! Time = \(\Theta(n^3)\)