Course Evaluations
All-Pairs Shortest Paths (Chapter 25)

• Weighted Directed Graph

• Could run a single-source shortest paths algorithm $|V|$ times
  - If graph has negative weight edges and cycles
    + Bellman-Ford algorithm runs in $\Theta(EV)$
    + All vertices $\Theta(EV^2)$. Dense graph: $\Theta(V^4)$
  - If graph has no negative edges (like for route finding)
    + Dijkstra’s algorithm runs in $\Theta(E \log V)$
    + All vertices $\Theta(EV \log V)$. Dense: $\Theta(V^3 \log V)$

• Let’s do better! Mapping applications depend on this!
Adjacency Matrix

• Most of the algorithms in this chapter use adjacency matrix
  - Vertices numbered 1 to $|V|$ (let $n = |V|$)
  - Matrix is $W = (w_{ij})$

$$w_{ij} = \begin{cases} 
0 & \text{if } i = j \\
\text{weight of directed edge } (i, j) & \text{if } i \neq j \text{ and } (i, j) \in E \\
\infty & \text{otherwise}
\end{cases}$$

• Output will be a $n \times n$ array $D = (d_{ij})$
  - $d_{ij}$ will be shortest-path weight from $i$ to $j$
  - Allowing negative weight edges, but no negative cycles

• Also need a predecessor matrix $\Pi = (\pi_{ij})$
  - $\pi_{ij} = \text{nil}$ if $i = j$ or no path from $i$ to $j$
  - otherwise, $\pi_{ij}$ is predecessor of $j$ on some shortest path from $i$
  - So row $i$ are all of the predecessors for shortest paths from $i$
Overview

⇒ Shortest Paths

• Shortest Paths and Matrix Multiplication

• Floyd-Warshall Algorithm
Dynamic Programming

• Characterize the structure of an optimal solution
• Recursively Define the value of an optimal solution
• Compute the value of an optimal solution in a bottom-up fashion
• Construct optimal solution from computed information
Optimal Substructure?

- Directed graph, negative edges, but no negative cycles
- Step 1 of dynamic programming
  - Characterize the optimal solution
- Say $p$ is shortest path from $u$ to $v$ and $p = \langle v_0, v_1, ... v_k \rangle$
  - for any $j$, path $\langle v_0, v_1, ..., v_j \rangle$ is shortest for $v_0$ to $v_j$ (Lemma 24.1)
  - but also for $i < j$, we have $\langle v_i, v_{i+1}, ..., v_j \rangle$ is shortest path for $v_i$ to $v_j$
  - Can use these optimal subpaths over and over again!
  - But how?
    + For any $i, j$ if we know that the last step goes from $k$ to $j$, overall path is optimal path from $i$ to $k$ plus edge $(k, j)$
    + But is optimal path from $i$ to $k$ any simpler?
    + It will have one less edge than path from $i$ to $j$
Recursively Define Value of an Optimal Solution

• Consider shortest paths up to length \( m \)
  - \( l^{(m)}_{ij} \) be min weight of any path from \( i \) to \( j \) that contains at most \( m \) edges
  - \( l^{(0)}_{ij} \) is 0 if \( i = j \) and \( \infty \) if \( i \neq j \)
  - \( l^{(m)}_{ij} = \min(l^{(m-1)}_{ij}, \min_{1 \leq k \leq n}(l^{(m-1)}_{ik} + w_{kj})) \)
  - \( l^{(m)}_{ij} = \min_{1 \leq k \leq n}(l^{(m-1)}_{ik} + w_{kj}) \) since can just add on \( w_{jj} \) which is 0

* Let \( L^{(m)} \) be the array with entries \( l^{(m)}_{ij} \). Can write \( L^{(m)} = (l^{(m)}_{ij}) \)
* What is \( L^{(1)} \)?
Shortest Path Weights

• If graph has no negative weight cycles
  - If \( j \) is reachable from \( i \), shortest path exists from \( i \) to \( j \) will have at most \( n-1 \) edges
  - \( \delta(i, j) = l_{ij}^{(n-1)} \) since we can just pad on \( w_{j,j} \)
  - In fact \( \delta(i, j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n+1)} = ... \)
Computing Shortest-path Bottom-up

• Start with $L^{(1)} = W$
  - Compute $L^{(2)}$, then $L^{(3)}$, then $L^{(4)}$ ...
  - Just need the previous one to compute the next one

**EXTEND-SHORTEST-PATHS** $(L, W)$

1. $n = L$.rows
2. let $L' = (l'_{ij})$ be a new $n \times n$ matrix
3. for $i = 1$ to $n$
4.   for $j = 1$ to $n$
5.     $l'_{ij} = \infty$
6.   for $k = 1$ to $n$
7.     $l'_{ij} = \min(l'_{ij}, l_{ik} + w_{kj})$
8. return $L'$

• Time complexity?

© P. Heeman, 2017
Rest of Code

**SLOW-ALL-PAIRS-SHORTEST-PATHS** (*W*)

1. \( n = W\.rows \)
2. \( L^{(1)} = W \)
3. **for** \( m = 2 \) **to** \( n - 1 \)
4. let \( L^{(m)} \) be a new \( n \times n \) matrix
5. \( L^{(m)} = \text{EXTEND-SHORTEST-PATHS}(L^{(m-1)}, W) \)
6. **return** \( L^{(n-1)} \)

- Can view this as: ESP(...,ESP(ESP(W,W),W)...,W)
Example

- \( L^{(1)} \) is just \( W \)
- \( L^{(2)} \) is like \( W \) but for hops of at most length 2

\[
L^{(1)} = \begin{pmatrix}
0 & 3 & 8 & \infty & -4 \\
\infty & 0 & \infty & 1 & 7 \\
\infty & 4 & 0 & \infty & \infty \\
2 & \infty & -5 & 0 & \infty \\
\infty & \infty & \infty & 6 & 0 \\
\end{pmatrix}
\]

\[
L^{(2)} = \begin{pmatrix}
0 & 3 & 8 & 2 & -4 \\
3 & 0 & -4 & 1 & 7 \\
\infty & 4 & 0 & 5 & 11 \\
2 & -1 & -5 & 0 & -2 \\
8 & \infty & 1 & 6 & 0 \\
\end{pmatrix}
\]

\[
L^{(3)} = \begin{pmatrix}
0 & 3 & 2 & 2 & -4 \\
3 & 0 & -4 & 1 & -1 \\
7 & 4 & 0 & 5 & 11 \\
2 & -1 & -5 & 0 & -2 \\
8 & 5 & 1 & 6 & 0 \\
\end{pmatrix}
\]

\[
L^{(4)} = \begin{pmatrix}
0 & 1 & -3 & 2 & -4 \\
3 & 0 & -4 & 1 & -1 \\
7 & 4 & 0 & 5 & 3 \\
2 & -1 & -5 & 0 & -2 \\
8 & 5 & 1 & 6 & 0 \\
\end{pmatrix}
\]
Overview

• Shortest Paths

⇒ Shortest Paths and Matrix Multiplication

• Floyd-Warshall Algorithm
Shortest Paths is like Matrix Multiplication

**EXTEND-SHORTEST-PATHS** \((L, W)\)

1. \(n = L\.\text{rows}\)
2. let \(L' = (l'_{ij})\) be a new \(n \times n\) matrix
3. for \(i = 1\) to \(n\)
   4. for \(j = 1\) to \(n\)
      5. \(l'_{ij} = \infty\)
   6. for \(k = 1\) to \(n\)
      7. \(l'_{ij} = \min(l'_{ij}, l_{ik} + w_{kj})\)
8. return \(L'\)

**SQUARE-MATRIX-MULTIPLY** \((A, B)\)

1. \(n = A\.\text{rows}\)
2. let \(C\) be a new \(n \times n\) matrix
3. for \(i = 1\) to \(n\)
   4. for \(j = 1\) to \(n\)
      5. \(c_{ij} = 0\)
   6. for \(k = 1\) to \(n\)
      7. \(c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}\)
8. return \(C\)

- Pretty similar
  - \(+\) ⇒ \(\times\) and \(\min\) ⇒ \(+\)
  - Identity for \(\min\) ⇒ identify for \(+\)
  - In fact, just as matrix \(\times\) is associative, so is Extend-Shortest-Paths
  
  \[ L^{(4)} = ESP(ESP(ESP(W,W),W),W) = ESP(EPS(W,W),EPS(W,W)) \]
Towards a Faster Implementation

- $L^{(1)}$ is just $W$
- $L^{(2)}$ is like $W$ but for hops of at most length 2
- To compute $L^{(4)}$, can call routine on $L^{(2)}$ and $L^{(2)}$
- To compute $L^{(8)}$, can call routine on $L^{(4)}$ and $L^{(4)}$
- Can compute $L^{(n-1)}$ in $O(\lfloor \log n \rfloor)$ steps
- Overall time is $O(V^3 \log(V))$

* Is this impressive? Dijkstra’s on all vertices also is $O(V^3 \log(V))$
Overview

• Shortest Paths
• Shortest Paths and Matrix Multiplication
⇒ Floyd-Warshall Algorithm
Structure of a Shortest Path

• Previously, characterized the optimal substructure for a shortest path from $s \rightsquigarrow v$ as $s \rightsquigarrow u \rightarrow v$
  - Consider paths of shorter and shorter lengths
  - Applied dynamic programming in bottom-up approach

• Think of the binary back-pack problem
  - How did we formulate the subproblems?
Different Optimal Substructure Approach

• Say $G$ has $n$ vertices: $\{1, \ldots, n\}$

• Consider subset $\{1, \ldots, k\} = V_k$

• For any pair of vertices $i, j$
  + Consider paths whose intermediate vertices are in $\{1, \ldots, k\}$
  + Say $p$ is a minimum weight path in that set

• Case 1: $k$ is not an intermediate vertex in $p$
  + All intermediate vertices of $p$ are in $\{1, \ldots, k - 1\}$
  + Shortest path from $i$ to $j$ with all intermediate vertices in $V_{k-1}$ is a shortest path with all intermediate vertices in $V_k$

• Case 2: $k$ is an intermediate vertex in $p$
  + We can assume that $k$ just appears once in $p$
  + Can decompose $p$ into $i \xrightarrow{p_1} k \xrightarrow{p_2} j$ where all intermediate vertices of $p_1$ and $p_2$ are in $V_{k-1}$
Recursive Solution

- Let $d_{ij}^{(k)}$ be the weight of a shortest path from $i$ to $j$ for which all intermediate vertices are in $V_k$.

- $d_{ij}^{(0)} = w_{ij}$ since cannot have any intermediate vertices $V_0 = \emptyset$
  - Can at most have one edge: $(i, j)$ if it is in $E$
  - If no edge $(i, j)$, $w_{ij} = \infty$

- $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
  - WOW!
FLOYD-WARSHALL (W)

1. \( n = W.rows \)
2. \( D^{(0)} = W \)
3. for \( k = 1 \) to \( n \)
   4. let \( D^{(k)} = (d^{(k)}_{ij}) \) be a new \( n \times n \) matrix
   5. for \( i = 1 \) to \( n \)
   6. for \( j = 1 \) to \( n \)
   7. \( d^{(k)}_{ij} = \min(d^{(k-1)}_{ij}, d^{(k-1)}_{ik} + d^{(k-1)}_{kj}) \)
8. return \( D^{(n)} \)

• Time complexity?
Determining the Paths

• Need to keep track of the predecessors

• $\Pi^{(i)}$ corresponds to $D^{(i)}$ for $0 \leq i \leq n$
  
  - $\pi_{ij}^{(i)}$ predecessor of $j$ on shortest path from vertex $i$ with all intermediate vertices in $V_k$

• $\pi_{ij}^{(0)} = ??$

• How do we modify code?