Today:
- All Pairs Shortest Paths
Reading Assignments

• Today’s class:
  – Chapter 25.1-25.2

• Reading assignment for next class:
  – Chapter 16.1-16.2

• Announcement: Exam 1 is on Tues, Feb. 18
  – Will cover everything up through dynamic programming
All Pairs Shortest Paths (APSP)

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

- **goal**: create an $n \times n$ matrix $L = (l_{ij})$ of shortest path distances i.e., $l_{ij} = \delta(i, j)$

- **trivial solution**: run a SSSP algorithm $n$ times, one for each vertex as the source.
All Pairs Shortest Paths (APSP)

- **all edge weights are nonnegative**: use Dijkstra’s algorithm
  - Priority Queue = linear array: $O(V^3 + VE) = O(V^3)$
  - Priority Queue = binary heap: $O(V^2 \log V + EV \log V) = O(V^3 \log V)$
    - better only for sparse graphs
  - Priority Queue = Fibonacci heap: $O(V^2 \log V + EV) = O(V^3)$
    - better only for sparse graphs

- **negative edge weights**: use Bellman-Ford algorithm
  - $O(V^2 E) = O(V^4)$ on dense graphs
Shortest Paths and Matrix Multiplication

Assumption: negative edge weights may be present, but no negative weight cycles.

(Step 1) Structure of a Shortest Path (new Optimal Substructure argument):

• Consider a shortest path $p_{ij}^m$ from $v_i$ to $v_j$ such that $|p_{ij}^m| \leq m$
  ▶ i.e., path $p_{ij}^m$ has at most $m$ edges.

• no negative-weight cycle $\Rightarrow$ all shortest paths are simple
  $\Rightarrow$ $m$ is finite $\Rightarrow$ $m \leq |V| - 1$

• $i = j$ $\Rightarrow$ $|p_{ii}| = 0$ & $\omega(p_{ii}) = 0$

• $i \neq j$ $\Rightarrow$ decompose path $p_{ij}^m$ into $p_{ik}^{m-1}$ & $v_k \rightarrow v_j$, where $|p_{ik}^{m-1}| \leq m - 1$
  ▶ $p_{ik}^{m-1}$ should be a shortest path from $v_i$ to $v_k$ by optimal substructure property.
  ▶ Therefore, $\delta(i, j) = \delta(i, k) + \omega_{kj}$
Shortest Paths and Matrix Multiplication

(Step 2): A Recursive Solution to All Pairs Shortest Paths Problem:

- $l_{ij}^m$ = minimum weight of any path from $v_i$ to $v_j$ that contains at most “$m$” edges.

- $m = 0$: There exists a shortest path from $v_i$ to $v_j$ with no edges $\leftrightarrow i = j$.

  $$l_{ij}^0 = \begin{cases} 
  0 & \text{if } i = j \\
  \infty & \text{if } i \neq j
  \end{cases}$$

- $m \geq 1$: $l_{ij}^m = \min \{ l_{ij}^{m-1}, \min_{1 \leq k \leq n \land k \neq j} \{l_{ik}^{m-1} + \omega_{kj}\}\}$
  
  $$= \min_{1 \leq k \leq n} \{l_{ik}^{m-1} + \omega_{kj}\} \text{ for all } v_k \in V,$$

  since $\omega_{jj} = 0$ for all $v_j \in V.$
Shortest Paths and Matrix Multiplication

• To consider all possible shortest paths with \( \leq m \) edges from \( v_i \) to \( v_j \)
  ▶ consider shortest path with \( \leq m - 1 \) edges, from \( v_i \) to \( v_k \), where \((v_k, v_j) \in E\)
Shortest Paths and Matrix Multiplication

(Step 3) Computing the shortest-path weights bottom-up:

- Given $W = L^1$, compute a series of matrices $L^2, L^3, ..., L^{n-1}$, where $L^m = (l_{ij}^m)$ for $m = 1, 2, ..., |V| - 1$
  - final matrix $L^{n-1}$ contains actual shortest path weights, i.e., $l_{ij}^{n-1} = \delta(i, j)$

- **SLOW-APSP**($W$)
  
  $L^1 \leftarrow W$
  
  for $m \leftarrow 2$ to $n-1$ do
  
  $L^m \leftarrow \text{EXTEND}(L^{m-1}, W)$
  
  return $L^{n-1}$
EXTEND \((L, W)\)
\[ L = (l_{ij}) \text{ is an } n \times n \text{ matrix} \]
\[
\text{for } i \leftarrow 1 \text{ to } n \text{ do} \\
\hspace{1cm} \text{for } j \leftarrow 1 \text{ to } n \text{ do} \\
\hspace{2cm} l_{ij} \leftarrow \infty \\
\hspace{2cm} \text{for } k \leftarrow 1 \text{ to } n \text{ do} \\
\hspace{3cm} l_{ij} \leftarrow \min\{l_{ij}, l_{ik} + \omega_{kj}\} \\
\]
\[\text{return } L\]

MATRIX-MULT \((A, B)\)
\[ C = (c_{ij}) \text{ is an } n \times n \text{ result matrix} \]
\[
\text{for } i \leftarrow 1 \text{ to } n \text{ do} \\
\hspace{1cm} \text{for } j \leftarrow 1 \text{ to } n \text{ do} \\
\hspace{2cm} c_{ij} \leftarrow 0 \\
\hspace{2cm} \text{for } k \leftarrow 1 \text{ to } n \text{ do} \\
\hspace{3cm} c_{ij} \leftarrow c_{ij} + a_{ik} \times b_{kj} \\
\]
\[\text{return } C\]
Shortest Paths and Matrix Multiplication

- Relation to matrix multiplication $C = A \times B$: $c_{ij} = \sum_{1 \leq k \leq n} a_{ik} \times b_{kj}$,
  - $L^{m-1} \leftrightarrow A$ & $W \leftrightarrow B$ & $L^m \leftrightarrow C$
  - “min” $\leftrightarrow “+”$ & “+” $\leftrightarrow “\times”$ & “$\infty$” $\leftrightarrow “0”$

- Thus, we compute the sequence of matrix products
  - $L^1 = L^0 \times W = W$; note $L^0$ = identity matrix,
  - $L^2 = L^1 \times W = W^2$
  - $L^3 = L^2 \times W = W^3$
  - $\vdots$
  - $L^{n-1} = L^{n-2} \times W = W^{n-1}$

- Running time: $\Theta(V^4)$
  - each matrix product: $\Theta(|V|^3)$
  - number of matrix products: $|V| - 1$
Shortest Paths and Matrix Multiplication

Example:
Shortest Paths and Matrix Multiplication

\[ L^1 = L^0 W \]
Shortest Paths and Matrix Multiplication

\[ L^2 = L^1W \]
Shortest Paths and Matrix Multiplication

\[
L^3 = L^2 W
\]
Shortest Paths and Matrix Multiplication

\[ L^4 = L^3 W \]
Idea: goal is not to compute all $L^m$ matrices
▶ we are interested only in matrix $L^{n-1}$

Recall: no negative-weight cycles $\Rightarrow L^m = L^{n-1}$ for all $m \geq |V| - 1$

We can compute $L^{n-1}$ with only $\lceil \log(n-1) \rceil$ matrix products as

\[
\begin{align*}
L^1 &= W \\
L^2 &= W^2 = W \times W \\
L^4 &= W^4 = W^2 \times W^2 \\
L^8 &= W^8 = W^4 \times W^4 \\
&\quad \vdots \\
L^{2^\lceil \log(n-1) \rceil} &= L^2^{\lceil \log(n-1) \rceil} = L^2^{\lceil \log(n-1) \rceil - 1} \times L^2^{\lceil \log(n-1) \rceil - 1}
\end{align*}
\]

This technique is called repeated squaring.
Improving Running Time Through Repeated Squaring

- **FASTER-APSP (W)**
  
  \[
  L^1 \leftarrow W \\
  m \leftarrow 1 \\
  \text{while } m < n-1 \text{ do} \\
  \quad L^{2m} \leftarrow \text{EXTEND}(L^m, L^m) \\
  \quad m \leftarrow 2m \\
  \text{return } L^m
  \]

- Final iteration computes \( L^{2m} \) for some \( n-1 \leq 2m \leq 2n-2 \Rightarrow L^{2m} = L^{n-1} \)

- **Running time**: \( \Theta(n^3 \lg n) = \Theta(V^3 \lg V) \)

  - each matrix product: \( \Theta(n^3) \)
  - # of matrix products: \( \lceil \lg(n-1) \rceil \)
  - simple code, no complex data structures, small hidden constants in \( \Theta \)-notation.
Exercise

Give an efficient algorithm to find the length (number of edges) of a minimum-length negative-weight cycle in a graph.
Floyd-Warshall Algorithm

Assumption: negative-weight edges, but no negative-weight cycles

(Step 1) The Structure of a Shortest Path (yet another optimal substructure argument):

- Definition: intermediate vertex of a path \( p = < v_1, v_2, v_3, \ldots, v_k > \)
  - any vertex of \( p \) other than \( v_1 \) or \( v_k \).

- \( p_{ij}^m \): a shortest path from \( v_i \) to \( v_j \) with all intermediate vertices from \( V_m = \{ v_1, v_2, \ldots, v_m \} \)

- Relationship between \( p_{ij}^m \) and \( p_{ij}^{m-1} \)
  - depends on whether \( v_m \) is an intermediate vertex of \( p_{ij}^m \)

  - Case 1: \( v_m \) is not an intermediate vertex of \( p_{ij}^m \)
    - all intermediate vertices of \( p_{ij}^m \) are in \( V_{m-1} \)
    - \( p_{ij}^m = p_{ij}^{m-1} \)
- Case 2: \( v_m \) is an intermediate vertex of \( p_{ij}^m \)

- decompose path as \( v_i \rightarrow v_m \rightarrow v_j \)

\[ \Rightarrow p_1 : v_i \rightarrow v_m \quad \& \quad p_2 : v_m \rightarrow v_j \]

- by opt. structure property both \( p_1 \) & \( p_2 \) are shortest paths.

- \( v_m \) is not an intermediate vertex of \( p_1 \) & \( p_2 \)

\[ \Rightarrow p_1 = p_{im}^{m-1} \quad \& \quad p_2 = p_{mj}^{m-1} \]
(Step 2) A Recursive Solution to APSP Problem:

- \( d_{ij}^m = \omega(p_{ij}) \): weight of a shortest path from \( v_i \) to \( v_j \) with all intermediate vertices from \( V_m = \{ v_1, v_2, \ldots, v_m \} \).

- Note: \( d_{ij}^n = \delta(i, j) \) since \( V_n = V \)
  - i.e., all vertices are considered for being intermediate vertices of \( p_{ij}^n \).
Floyd-Warshall Algorithm

• Compute $d_{ij}^m$ in terms of $d_{ij}^k$ with smaller $k < m$

• $m = 0$ : $V_0 = \text{empty set}$
  \[ \Rightarrow \text{path from } v_i \text{ to } v_j \text{ with no intermediate vertex.} \]
  i.e., $v_i$ to $v_j$ paths with at most one edge
  \[ \Rightarrow d_{ij}^0 = \omega_{ij} \]

• $m \geq 1$ : $d_{ij}^m = \min \{ d_{ij}^{m-1}, d_{im}^{m-1} + d_{mj}^{m-1} \}$
Floyd-Warshall Algorithm

(Step 3) Computing Shortest Path Weights Bottom Up:

FLOYD-WARSHALL( W )

- \( D^0, D^1, \ldots, D^n \) are \( n \times n \) matrices

for \( m \leftarrow 1 \) to \( n \) do

for \( i \leftarrow 1 \) to \( n \) do

for \( j \leftarrow 1 \) to \( n \) do

\[ \text{d}_{ij}^m \leftarrow \min \{ \text{d}_{ij}^{m-1}, \text{d}_{im}^{m-1} + \text{d}_{mj}^{m-1} \} \]

return \( D^n \)
Floyd-Warshall Algorithm

FLOYD-WARSHALL ( W )

► D is an $n \times n$ matrix

D ← W

for $m \leftarrow 1$ to $n$ do
    for $i \leftarrow 1$ to $n$ do
        for $j \leftarrow 1$ to $n$ do
            if $d_{ij} > d_{im} + d_{mj}$ then
                $d_{ij} \leftarrow d_{im} + d_{mj}$

return D
Floyd-Warshall Algorithm

• Maintaining \( n \) \( D \) matrices can be avoided by dropping all superscripts.
  
  \( m \)-th iteration of outermost for-loop
  
  begins with \( D = D^{m-1} \)

  ends with \( D = D^m \)

  computation of \( d_{ij}^m \) depends on \( d_{im}^{m-1} \) and \( d_{mj}^{m-1} \).

  no problem if \( d_{im} \) & \( d_{mj} \) are already updated to \( d_{im}^m \) & \( d_{mj}^m \)

  since \( d_{im}^m = d_{im}^{m-1} \) & \( d_{mj}^m = d_{mj}^{m-1} \).

• Running time : ⋁( \( n^3 \) ) = ⋁( \( V^3 \) )

  simple code, no complex data structures, small hidden constants
Reading Assignments

• Reading assignment for next class:
  – Chapter 16.1-16.2

• Announcement: Exam 1 is on Tues, Feb. 18
  – Will cover everything up through dynamic programming