Dynamic Programming (All pairs shortest path)

17 November 2015
All-pairs shortest paths

- Directed graph $G = (V, E)$, weight function $w : E \to \mathbb{R}$, $|V| = n$
- Assume $G$ contains no negative-weight cycles
- **Goal:** create $n \times n$ matrix of shortest path distances $\delta(u, v)$, $u, v \in V$
- Adjacency-matrix representation of graph:
  - $n \times n$ adjacency matrix $W = (w_{ij})$ of edge weights
  - assume $w_{ij} = \begin{cases} 0 & \text{if } i = j \\ \text{weight of } (i, j) & \text{if } i \neq j \text{ and } (i, j) \in E \\ \infty & \text{if } i \neq j \text{ and } (i, j) \notin E \end{cases}$

- Weight of path $p = (v_1, v_2, \ldots, v_k)$ is $w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$
1. Structure of a shortest path

Subpaths of shortest paths are shortest paths

**Lemma.** Let \( p = (v_1, v_2, \ldots, v_\ell) \) be a shortest path from \( v_1 \) to \( v_\ell \), let \( p_{ij} = (v_i, v_{i+1}, \ldots, v_j) \) for \( 1 \leq i \leq j \leq \ell \) be subpath from \( v_i \) to \( v_j \). Then, \( p_{ij} \) is shortest path from \( v_i \) to \( v_j \).
1. Structure of a shortest path

Subpaths of shortest paths are shortest paths

**Lemma.** Let \( p = (v_1, v_2, \ldots, v_\ell) \) be a shortest path from \( v_1 \) to \( v_\ell \), let \( p_{ij} = (v_i, v_{i+1}, \ldots, v_j) \) for \( 1 \leq i \leq j \leq \ell \) be subpath from \( v_i \) to \( v_j \). Then, \( p_{ij} \) is shortest path from \( v_i \) to \( v_j \).

**Proof.** Decompose \( p \) into

\[
v_1 \xrightarrow{p_{1i}} v_i \xrightarrow{p_{ij}} v_j \xrightarrow{p_{j\ell}} v_\ell.
\]

Then, \( w(p) = w(p_{1i}) + w(p_{ij}) + w(p_{j\ell}) \). Assume there is cheaper \( p'_{ij} \) from \( v_i \) to \( v_j \) with \( w(p'_{ij}) < w(p_{ij}) \). Then

\[
v_1 \xrightarrow{p_{1i}} v_i \xrightarrow{p'_{ij}} v_j \xrightarrow{p_{j\ell}} v_\ell
\]

is path from \( v_1 \) to \( v_\ell \) whose weight \( w(p_{1i}) + w(p'_{ij}) + w(p_{j\ell}) \) is less than \( w(p) \), a contradiction.
2. Recursive solution to Compute opt. value (bottom-up)

Let $d_{ij}^{(m)}$ = weight of shortest path from $i$ to $j$ that uses at most $m$ edges.

\[
\begin{align*}
\delta_{ij}^{(0)} &= \begin{cases} 
0 & \text{if } i = j \\
\infty & \text{if } i \neq j
\end{cases} \\
\delta_{ij}^{(m)} &= \min_k \left\{ \delta_{ik}^{(m-1)} + w_{kj} \right\}
\end{align*}
\]
2. Recursive solution to Compute opt. value (bottom-up)

Let \( d_{ij}^{(m)} \) = weight of shortest path from \( i \) to \( j \) that uses at most \( m \) edges.

\[
\begin{align*}
    d_{ij}^{(0)} &= \begin{cases} 
        0 & \text{if } i = j \\
        \infty & \text{if } i \neq j 
    \end{cases} \\
    d_{ij}^{(m)} &= \min_k \left\{ d_{ik}^{(m-1)} + w_{kj} \right\}
\end{align*}
\]

**Note:** the shortest path from \( i \) to \( j \) can use at most \( n - 1 \) edges (\( n \) is the number of vertices).

Hence, we’re looking for

\[ \delta(i, j) = d_{ij}^{(n-1)} \]

The algorithm is straightforward, running time is \( \mathcal{O}(n^3) \) (\( n - 1 \) passes, each computing \( n^2 \) \( d \)'s in \( \Theta(n) \) time)
Implementing all pairs shortest path algorithm

\textbf{All-pair-shortest}(G,A)

1. \textbf{for} \ i = 1 \ to \ n
2. \ \ \textbf{for} \ j = 1 \ to \ n
3. \ \ \ W[i][j] = A[i][j]
   \quad // \text{if there is no arc from } i \text{ to } j \text{ then } A[i][j] = \infty
4. \ \ \textbf{for} \ k = 1 \ to \ n
5. \ \ \textbf{for} \ i = 1 \ to \ n
6. \ \ \textbf{for} \ j = 1 \ to \ n
7. \ \ \ \textbf{if} \ ( W[i][k] + W[k][j] < W[i][j] )
8. \ \ \ W[i][j] = W[i][k] + W[k][j]

Time complexity $O(n^3)$
**Finding all pairs shortest path**

### All-pairs-shortest(G,A)

1. **for** \( i = 1 \) to \( n \)

2. **for** \( j = 1 \) to \( n \)

3. \( W[i][j] = A[i][j], \; P[i][j] = j; \)
   // if there is no arc from \( i \) to \( j \) then \( A[i][j] = \infty \)

4. **for** \( k = 1 \) to \( n \)

5. **for** \( i = 1 \) to \( n \)

6. **for** \( j = 1 \) to \( n \)

7. **if** \( W[i][k] + W[k][j] < W[i][j] \)

8. \( W[i][j] = W[i][k] + A[k][j], \; P[i][j] = k \)

**Time complexity** \( \mathcal{O}(n^3) \)
Dynamic Programming (All pairs shortest path)

\[
\begin{array}{cccccccc}
\text{a} & \text{b} & \text{c} & \text{d} & \text{e} & \text{t} \\
\hline
\text{a} & 0 & -4 & \infty & \infty & \infty & -3 \\
\text{b} & \infty & 0 & \infty & -1 & -2 & \infty \\
\text{c} & \infty & 8 & 0 & \infty & \infty & 3 \\
\text{d} & 6 & \infty & \infty & 0 & \infty & 4 \\
\text{e} & \infty & \infty & -3 & \infty & 0 & 2 \\
\text{t} & \infty & \infty & \infty & \infty & \infty & 0 \\
\end{array}
\]
Dynamic Programming (All pairs shortest path)
Dynamic Programming (All pairs shortest path)

\[
\begin{array}{ccccccc}
    & a & b & c & d & e & t \\
\hline
    a & 0 & -4 & \infty & -5 & -6 & -3 \\
b & \infty & 0 & \infty & -1 & -2 & \infty \\
c & \infty & 8 & 0 & 7 & 6 & 3 \\
d & 6 & 2 & \infty & 0 & 0 & 3 \\
e & \infty & \infty & -3 & \infty & 0 & 2 \\
t & \infty & \infty & \infty & \infty & \infty & 0 \\
\end{array}
\]

\[k = 2 = b\]
**Dynamic Programming (All pairs shortest path)**

The image shows a graph and a table representing the shortest paths between all pairs of vertices. The graph includes vertices labeled 'a', 'b', 'c', 'd', 'e', and 't', with weighted edges connecting them. The table below the graph is a matrix where each cell represents the shortest path between two vertices. For example, the shortest path from 'a' to 'b' is 0, from 'a' to 'c' is -4, and so on. The table also shows the lengths of the edges in the graph. The bottom row of the table, labeled 'k = 3 = c', indicates the shortest path from vertex 'c' to itself, which is 3.
**Dynamic Programming (All pairs shortest path)**

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>t</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>-4</td>
<td>∞</td>
<td>-5</td>
<td>-6</td>
<td>-3</td>
</tr>
<tr>
<td>b</td>
<td>5</td>
<td>0</td>
<td>∞</td>
<td>-1</td>
<td>-2</td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td>13</td>
<td>8</td>
<td>0</td>
<td>7</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>d</td>
<td>6</td>
<td>2</td>
<td>∞</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>e</td>
<td>10</td>
<td>5</td>
<td>-3</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ k = 4 = d \]
Dynamic Programming (All pairs shortest path)

\[ k = 5 = e \]
Dynamic Programming (All pairs shortest path)
Application: Transitive closure of a directed graph

given: a directed graph $G = (V, E)$
output: the transitive closure $G^* = (V, E^*)$,
where

$$(i, j) \in E^* \text{ if } \text{there is a path from } i \text{ to } j \text{ in } G$$

Easy to compute using the above algorithm:

assign weight 0 to all edges in $E$
run the All-pairs-shortest path algorithm
$C(i, j)$ is an edge in $E^*$ if and only if $c_{ij} = 0$

Note: only value in matrices $C$ are 0 and $\infty$, which can be interpreted as boolean values and operations $+$ and $\min$ can be replaced by logical operations AND and OR.

Dynamic Programming (All pairs shortest path)
given: a directed graph $G = (V, E)$
output: the **transitive closure** = a directed graph $G^* = (V, E^*)$, where

$$(i, j) \in E^* \quad \text{if} \quad \text{there is a path from } i \text{ to } j \text{ in } G$$

Easy to compute using the above algorithm:

- assign weight 0 to all edges in $E$ (if $(i, j) \notin E$, then $w_{ij} = \infty$)
- run the *All-pairs-shortest path* algorithm $\rightarrow C$
- $(i, j)$ is an edge in $E^*$ if and only if $c_{ij} = 0$
Application: Transitive closure of a directed graph

given: a directed graph $G = (V, E)$
output: the **transitive closure** $\text{a directed graph } G^* = (V, E^*)$, where

$$(i, j) \in E^* \text{ if there is a path from } i \text{ to } j \text{ in } G$$

Easy to compute using the above algorithm:

- assign weight 0 to all edges in $E$ (if $(i, j) \notin E$, then $w_{ij} = \infty$)
- run the *All-pairs-shortest path* algorithm $\rightarrow C$
- $(i, j)$ is an edge in $E^*$ if and only if $c_{ij} = 0$

*Note:* only value in matrices $C$ are 0 and $\infty$, which can be interpreted as boolean values and operations “$+$” and “$\text{min}$” can be replaced by logical operations AND and OR