All-pairs-shortest-paths

- Directed graph $G = (V, E)$, weight function $w : E \to \mathbb{R}$, $|V| = n$
- Weight of path $p = (v_1, v_2, \ldots, v_k)$ is $w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$
- Assume G contains no negative-weight cycles
- **Goal:** create $n \times n$ matrix of shortest path distances $\delta(u, v)$, $u, v \in V$
- **1st idea:** use single-source-shortest-path alg (i.e., Bellman-Ford); but it's too slow, $O(n^4)$ on dense graph

Adjacency-matrix representation of graph:

- $n \times n$ adjacency matrix $W = (w_{ij})$ of edge weights
- assume

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

In the following, we only want to compute lengths of shortest paths, not construct the paths.

Dynamic programming approach, four steps:

1. Structure of a shortest path: Subpaths of shortest paths are shortest paths.

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

Proof. Decompose p into

$$v_1 \overset{p_{1i}}{\leadsto} v_i \overset{p_{ij}}{\leadsto} v_j \overset{p_{jk}}{\leadsto} v_k.$$

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

$$v_1 \overset{p_{1i}}{\leadsto} v_i \overset{p'_{ij}}{\leadsto} v_j \overset{p_{jk}}{\leadsto} v_k$$

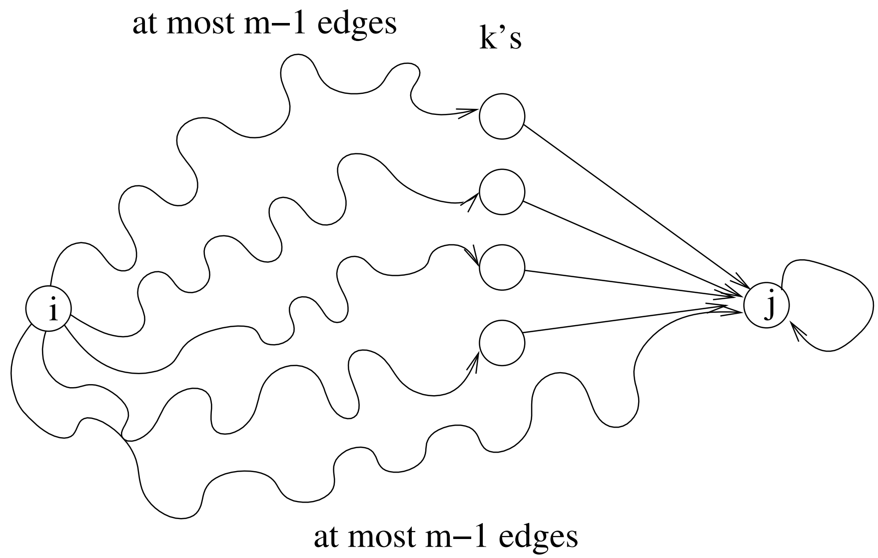
is path from v_1 to v_k whose weight $w(p_{1i}) + w(p'_{ij}) + w(p_{jk})$ is less than $w(p)$, a contradiction.

2. Recursive solution and 3. Compute opt. value (bottom-up)

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

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

$$d_{ij}^{(m)} = \min_k \left\{ d_{ik}^{(m-1)} + w_{kj} \right\}$$



We're looking for $\delta(i,j) = d_{ij}^{(n-1)} = d_{ij}^{(n)} = d_{ij}^{(n+1)} = \cdots$

Alg. is straightforward, running time is $O(n^4)$ ($n-1$ passes, each computing n^2 d's in $\Theta(n)$ time)

Unfortunately, no better than before...

Approach is similar to **matrix multiplication:**

$C = A \cdot B$, $n \times n$ matrices, $c_{ij} = \sum_k a_{ik} \cdot b_{kj}$, $O(n^3)$ operations

Replacing "$+$" with "min" and "$\cdot$" with "$+$" gives

$$c_{ij} = \min_k \{a_{ik} + b_{kj}\},$$

very similar to

$$d_{ij}^{(m)} = \min_k \{d_{ik}^{(m-1)} + w_{kj}\}$$

Hence $D^{(m)} = D^{(m-1)}$ "$\times$" W.

Floyd-Warshall algorithm

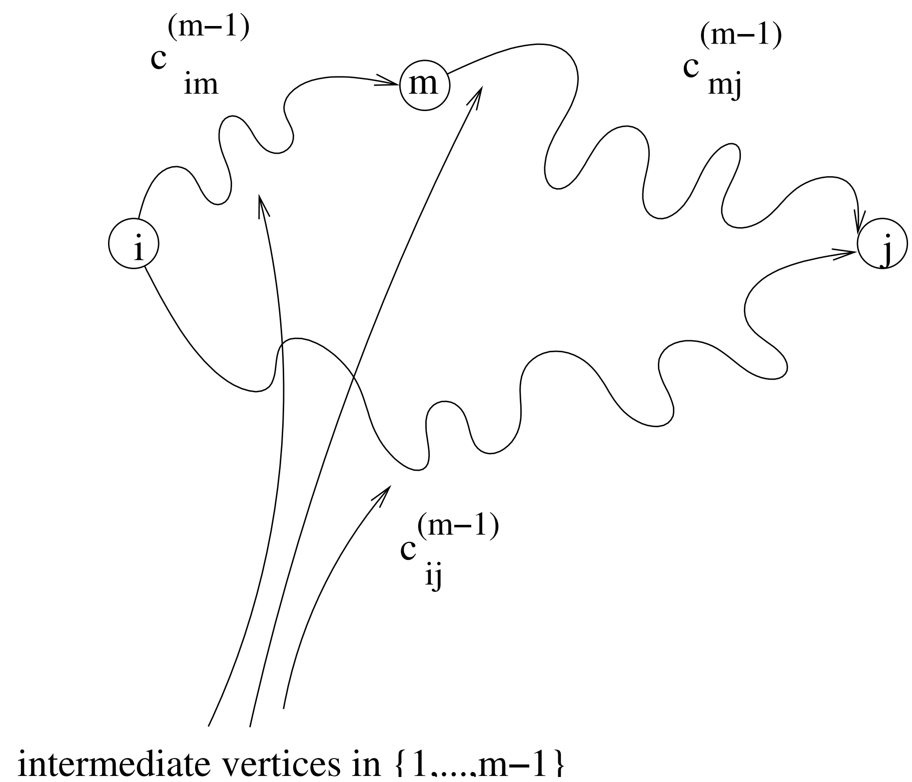
Also DP, but faster (factor $\log n$)

Define $c_{ij}^{(m)}$ = weight of a shortest path from i to j with **intermediate vertices** in $\{1, 2, \ldots, m\}$.

Then $\delta(i, j) = c_{ij}^{(n)}$

Compute $c_{ij}^{(n)}$ in terms of smaller ones, $c_{ij}^{(<n)}$:

$$c_{ij}^{(0)} = w_{ij}$$
$$c_{ij}^{(m)} = \min\left(c_{ij}^{(m-1)},\ c_{im}^{(m-1)} + c_{mj}^{(m-1)}\right)$$



Difference from previous algorithm: needn't check *all* possible intermediate vertices. Shortest path simply either includes m or doesn't.

Pseudocode:

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

Superscripts dropped, start loop with $c_{ij} = c_{ij}^{(m-1)}$, end with $c_{ij} = c_{ij}^{(m)}$

Time: $\Theta(n^3)$, simple code

Best algorithm to date is $O(V^2 \log V + VE)$

Note: for dense graphs ($|E| \approx |V|^2$) can get APSP (with Floyd-Warshall) for same cost as getting SSSP (with Bellman-Ford)! ($\Theta(VE) = \Theta(n^3)$)