Part VI

Graph Algorithms (II)

songyou@buaa.edu.cn

Graph Algorithms

- **Elementary Graph Algorithms**
  - **Representations of Graphs**
  - **BFS, DFS**
  - **Sort Topologically**
- **Single-Source Shortest Paths**

  Finding shortest paths from a given source vertex to all other vertices (BF, TS, DJ).
- **All-Pairs Shortest Paths**

  Computing shortest paths between every pair of vertices.
- **Maximum Flow**

25 All-Pairs Shortest Paths

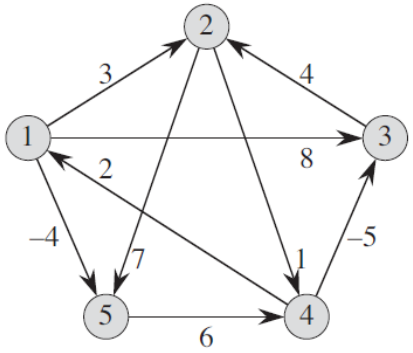
- How to find shortest paths between all pairs of vertices in a graph.
- This problem might arise in making a table of distances between all pairs of cities for a road atlas (地图集).
- We can solve an all-pairs shortest-paths problem by running a single-source shortest-paths algorithm $|V|$ times, once for each vertex as the source.

25 All-Pairs Shortest Paths

- Most of the algorithms in this chapter use an adjacency-matrix representation.
- The input is an $n \times n$ matrix $\boldsymbol{W}$ representing the edge weights of an n-vertex directed graph $G = (V, E)$, where

$$w_{ij} = \begin{cases} 0 & \text{if } i = j \ , \\ \text{the weight of directed edge } (i, j) & \text{if } i \neq j \text{ and } (i, j) \in E \ , \\ \infty & \text{if } i \neq j \text{ and } (i, j) \notin E \ . \end{cases} \qquad (25.1)$$

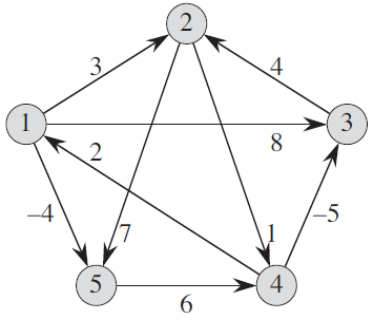
- **shortest-path weights:** The output is an $n \times n$ matrix $\boldsymbol{D} = (d_{ij})$.
  $d_{ij} = \delta(i, j)$ at termination.



$$\begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

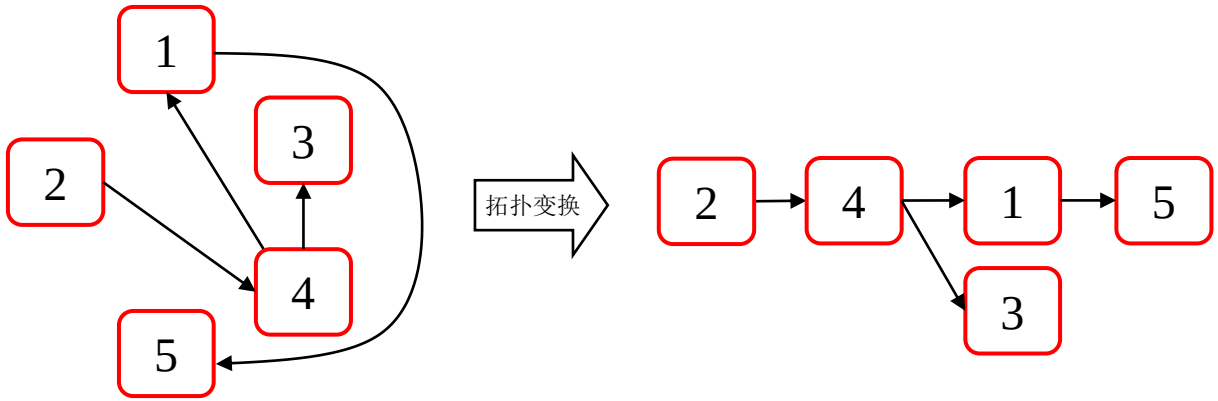
25 All-Pairs Shortest Paths

- We need to compute not only **the shortest-path weights *D*,**
- but also a ***predecessor matrix*** :
  $\Pi = (\pi_{ij})$, where π_{ij} is NIL if either $i = j$ or there is no path from i to j , and otherwise π_{ij} is the predecessor of j on some shortest path from i.
  前驱矩阵： π_{ij} 是从 i 到 j 的最短路径中 j 的前驱节点
- The subgraph induced by the ith row of the Π matrix should be a shortest-paths tree with root i
  (第i行是以i为根节点的最短路径树).
  For example, choose line 2



$$\begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} D$$

$$\begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix} \Pi$$



25.1 Shortest paths and matrix multiplication*

A dynamic-programming algorithm based on **matrix multiplication**.

The structure of a shortest path

Vertices i and j are distinct, then we decompose shortest path p into

$$\delta(i, j) = i \overset{p'}{\rightsquigarrow} k \rightarrow j$$

then p' is a shortest path i to k, and so

$$\delta(i, j) = \delta(i, k) + w_{kj} .$$

25.1 Shortest paths and matrix multiplication*

A recursive solution to the all-pairs shortest-paths problem

$l_{kj}^{(m)}$: the minimum weight of any path from i and j that contains at most m edges.

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

$$l_{ij}^{(m)} = \min_{1 \le k \le n} \{ l_{ik}^{(m-1)} + w_{kj} \} \ .$$

The graph contains no negative-weight cycles, then

$$\delta(i, j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n+1)} = \cdots \ .$$

25.1 Shortest paths and matrix multiplication*

Computing the shortest-path weights bottom up

Inputs : $W = (w_{ij})$

compute a series of matrices $L^{(i)}$, $i = 1, \ldots, n\text{-}1$. $L^{(m)} = (l_{kj}^{(m)})$. The final matrix $L^{(n\text{-}1)}$ contains the actual shortest-path weights.

$L^{(1)} = W$.

```
EXTEND-SHORTEST-PATHS(L, W)
1  n = L.rows
2  let L' = (l'_ij) be a new n × n matrix
3  for i = 1 to n
4       for j = 1 to n
5            l'_ij = ∞
6            for k = 1 to n
7                 l'_ij = min(l'_ij, l_ik + w_kj)
8  return L'
```

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

$$l_{ij}^{(m)} = \min_{1 \le k \le n} \{ l_{ik}^{(m-1)} + w_{kj} \} \ .$$

$$\delta(i, j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n+1)} = \cdots \ .$$

25.1 Shortest paths and matrix multiplication*

Computing the shortest-path weights bottom up

```
EXTEND-SHORTEST-PATHS(L, W)
1  n = L.rows
2  let L' = (l'_ij) be a new n × n matrix
3  for i = 1 to n
4       for j = 1 to n
5            l'_ij = ∞
6            for k = 1 to n
7                 l'_ij = min(l'_ij, l_ik + w_kj)
8  return L'
```

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

$$l_{ij}^{(m)} = \min_{1 \le k \le n} \{ l_{ik}^{(m-1)} + w_{kj} \} \ .$$

$$\delta(i, j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n+1)} = \cdots \ .$$

```
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 × n matrix
5       L^(m) = EXTEND-SHORTEST-PATHS(L^(m−1), W)
6  return L^(n−1)
```

Running time?

$\Theta(n^4)$

25.1 Shortest paths and matrix multiplication*

Improving the running time from matrix multiplication.

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

$$l_{ij}^{(m)} = \min_{1 \le k \le n} \{ l_{ik}^{(m-1)} + w_{kj} \} \ .$$

$$\delta(i,j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n+1)} = \cdots \ .$$

product $C = A \cdot B$ of two $n \times n$ matrices A and B

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj} \ .$$

Observe that if we make the substitutions

$$\begin{aligned} l^{(m-1)} &\to a \ , \\ w &\to b \ , \\ l^{(m)} &\to c \ , \\ \min &\to + \ , \\ + &\to \cdot \end{aligned}$$

EXTEND-SHORTEST-PATHS(L, W)

```
1  n = L.rows
2  let L' = (l'_ij) be a new n × n matrix
3  for i = 1 to n
4      for j = 1 to n
5          l'_ij = ∞
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.rows
2  let C be a new n × 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 · b_kj
8  return C
```

25.1 Shortest paths and matrix multiplication*

Improving the running time from matrix multiplication.

$$
\begin{aligned}
L^{(1)} &= L^{(0)} \cdot W &&= W\,, \\
L^{(2)} &= L^{(1)} \cdot W &&= W^2\,, \\
L^{(3)} &= L^{(2)} \cdot W &&= W^3\,, \\
&\vdots \\
L^{(n-1)} &= L^{(n-2)} \cdot W &&= W^{n-1}\,.
\end{aligned}
$$

n to *n*lg*n*

$$
\begin{aligned}
L^{(1)} &= W\,, \\
L^{(2)} &= W^2 &&= W \cdot W\,, \\
L^{(4)} &= W^4 &&= W^2 \cdot W^2 \\
L^{(8)} &= W^8 &&= W^4 \cdot W^4\,, \\
&\vdots \\
L^{(2^{\lceil \lg(n-1) \rceil})} &= W^{2^{\lceil \lg(n-1) \rceil}} &&= W^{2^{\lceil \lg(n-1) \rceil - 1}} \cdot W^{2^{\lceil \lg(n-1) \rceil - 1}}\,.
\end{aligned}
$$

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 × n matrix
5      L^(m) = EXTEND-SHORTEST-PATHS(L^(m−1), W)
6  return L^(n−1)
```

FASTER-ALL-PAIRS-SHORTEST-PATHS(W)

```
1  n = W.rows
2  L^(1) = W
3  m = 1
4  while m < n − 1
5      let L^(2m) be a new n × n matrix
6      L^(2m) = EXTEND-SHORTEST-PATHS(L^(m), L^(m))
7      m = 2m
8  return L^(m)
```

$\Theta(n^4)$ ⇒ $\Theta(n^3 \lg n)$

25.1 Shortest paths and matrix multiplication*

- A dynamic-programming algorithm based on **matrix multiplication** , $\Theta(V^4)$.

- "Repeated squaring," $\Theta(V^3 \lg V)$.

25.2 The Floyd-Warshall algorithm (DP)

The structure of a shortest path

- The Floyd-Warshall algorithm considers the intermediate vertices of a shortest path, where an *intermediate* vertex of a simple path $p = \langle v_1, v_2, \ldots, v_l \rangle$ is any vertex of p other than v_1 or v_l, that is, any vertex in the set $\{v_2, v_3, \ldots, v_{l-1}\}$.
  简单路径 p 的端点是v_1和v_l，其他点是 p 的“之间”顶点

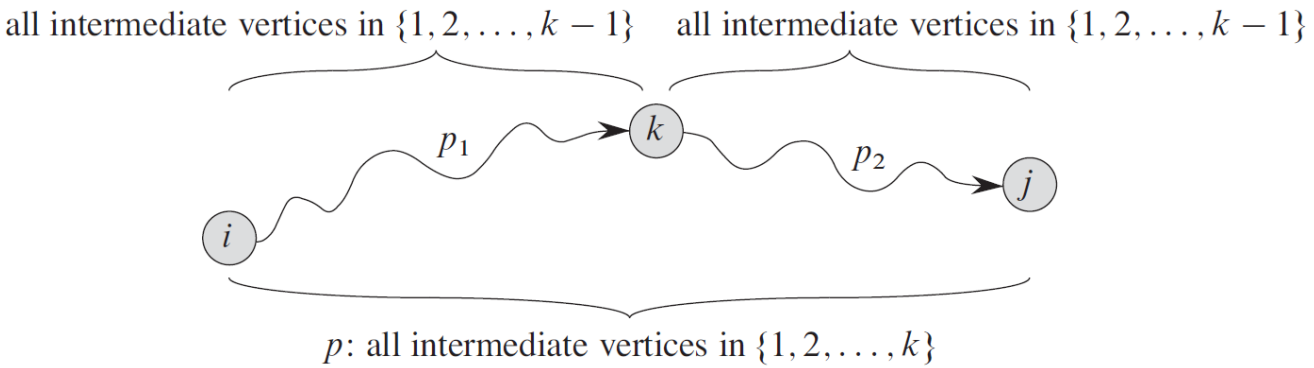
- Consider a subset $\{1, 2, \ldots, k\}$ of vertices for some k. For any pair of vertices $i, j \in V$, consider all paths from i to j whose intermediate vertices are all drawn from $\{1, 2, \ldots, k\}$, and let p be a minimum-weight path from among them. (Path p is simple.)
  考虑 i to j 的所有路径，其“之间”顶点from $\{1, 2, \ldots, k\}$，
  p的所有路径中最短的一个

25.2 The Floyd-Warshall algorithm (DP)

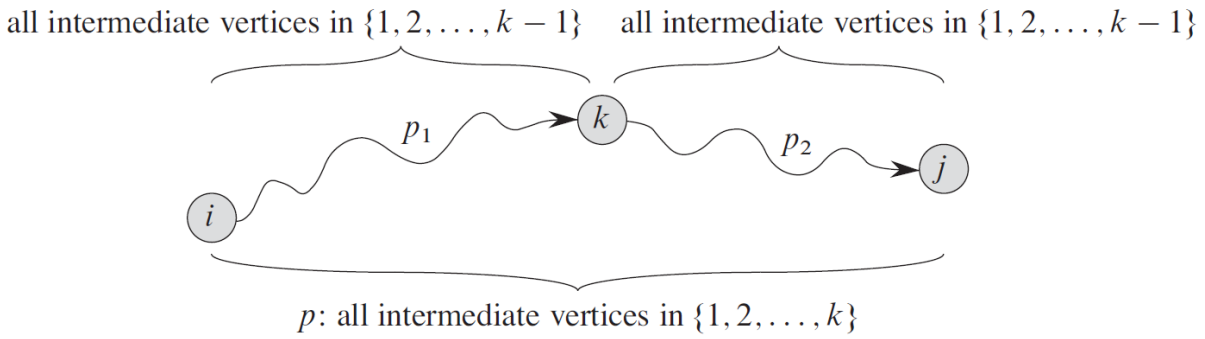
The structure of a shortest path

For any pair of vertices (i, j), all paths from i to j whose intermediate vertices are all drawn from $\{1, 2, \ldots, k\}$, and let p be a minimum path.

- **If k is not** an intermediate vertex of path p, then all intermediate vertices of p are in the set $\{1, 2, \ldots, k\text{-}1\}$. Thus, a *st-path-*$\delta(i, j)$ with all intermediate vertices in the set $\{1, 2, \ldots, k\text{-}1\}$ is also a *st-path-*$\delta(i, j)$ with all intermediate vertices in the set $\{1, 2, \ldots, k\}$.
- **If k is …** , then we decompose p into $i \overset{p_1}{\rightsquigarrow} k \overset{p_2}{\rightsquigarrow} j$. p_1 is a $\delta(i, k)$ with all intermediate vertices in the set $\{1, 2, \ldots, k\text{-}1\}$. Similarly, for p_2 , …



25.2 The Floyd-Warshall algorithm (DP)



A recursive solution to the all-pairs shortest-paths problem

Let $d_{ij}^{(k)}$ be the weight of a ***st-path-δ(i, j)*** for which all intermediate vertices are in the set {1, 2, … , k}. When $k = 0$, a ***path-p(i, j)*** has no intermediate vertices at all. Such a path has at most one edge. We define recursively

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 , \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 1 . \end{cases}$$

Because for any path, all intermediate vertices are in the set {1, 2, … , n}, the matrix $D^{(n)} = (d_{ij}^{(n)})$ gives the final answer: $d_{ij}^{(n)} = \delta(i, j)$ for all $i, j \in V$.

25.2 The Floyd-Warshall algorithm (DP)

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 , \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 1 . \end{cases}$$

Direct recursion algorithm? complexity?

Computing the shortest-path weights bottom up

We can use the following bottom-up procedure to compute the values $d_{ij}^{(k)}$ in order of increasing values of k.

```
FLOYD-WARSHALL(W)
1  n = W.rows
2  D^(0) = W
3  for k = 1 to n
4      let D^(k) = (d_ij^(k)) be a new n × n matrix
5      for i = 1 to n
6          for j = 1 to n
7              d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
8  return D^(n)
```

running time?

$\Theta(n^3)$

25.2 The Floyd-Warshall algorithm (DP)

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0, \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 1. \end{cases}$$

Constructing a shortest path

We compute a sequence of matrices $\Pi^{(0)}, \Pi^{(1)}, \ldots, \Pi^{(n)}$, where $\Pi = \Pi^{(n)}$ and we define $\pi_{ij}^{(k)}$ as the predecessor of vertex j on a shortest path from vertex i with all intermediate vertices in the set $\{1, 2, \ldots, k\}$.

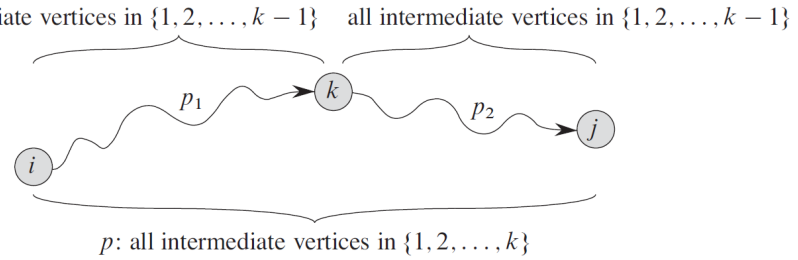
$$\pi_{ij}^{(0)} = \begin{cases} \text{NIL} & \text{if } i = j \text{ or } w_{ij} = \infty, \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty. \end{cases}$$

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, \\ \pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}. \end{cases}$$

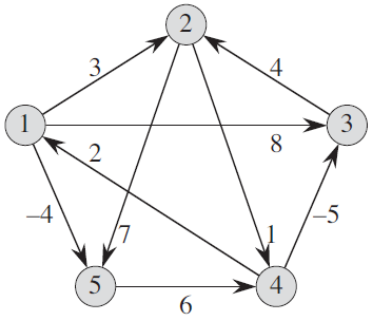
对最短路径$i \to j$,
1) k 不在最短路径上，最短路径的顶点是$\{1, 2, \ldots, k\text{-}1\}$，因此 $\pi^{(k)}=\pi^{(k\text{-}1)}$

2) k 在最短路径上，$i \to j$ 中 j 的前驱 $\pi_{ij}^{(k)}$ 显然就是的$k \to j$ 中 j 的前驱 $\pi_{kj}^{(k\text{-}1)}$

Exercises…



25.2 The Floyd-Warshall algorithm (DP)



$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0\,, \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 1\,. \end{cases}$$

$$\pi_{ij}^{(0)} = \begin{cases} \text{NIL} & \text{if } i = j \text{ or } w_{ij} = \infty\,, \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty\,. \end{cases}$$

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\,, \\ \pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\,. \end{cases}$$

$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

The Floyd-Warshall vs matrix multiplication

Floyd-Warshall $\Theta(n^3)$

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 , \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 1 . \end{cases}$$

上标和下标都是k，一个n遍历（k from 1 to n）实现两个维度的计算

```
FLOYD-WARSHALL(W)
1  n = W.rows
2  D^(0) = W
3  for k = 1 to n
4      let D^(k) = (d_ij^(k)) be a new n × n matrix
5      for i = 1 to n
6          for j = 1 to n
7              d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
8  return D^(n)
```

Floyd-Warshall is fast. Why?

matrix multiplication $\Theta(n^4)$

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

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

下标k和上标m，两个n遍历

```
EXTEND-SHORTEST-PATHS(L, W)
1  n = L.rows
2  let L' = (l'_ij) be a new n × n matrix
3  for i = 1 to n
4      for j = 1 to n
5          l'_ij = ∞
6          for k = 1 to n
7              l'_ij = min(l'_ij, l_ik + w_kj)
8  return L'
```

```
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 × n matrix
5      L^(m) = EXTEND-SHORTEST-PATHS(L^(m−1), W)
6  return L^(n−1)
```

25.2 The Floyd-Warshall algorithm

Transitive closure of a directed graph*

25.3 Johnson's algorithm for sparse graphs*

Summary of Graph Algorithms

- **Queue (Priority Queue)**
- **Enumeration (BFS, …)**
- **Recursion (DFS, …)**
- **Dynamic Programming (All-Pairs Shortest Paths)**
- **Greedy Strategy (Single-Source Shortest Paths)**
- **Relaxation**
- **Aggregate analysis**
- **…**