طراحی الگوریتم ها

جلسه ۱۶ و ۱۷

ملکی مجد

مباحث

مساله کوتاه ترین مسیرها بین همه جفت راس ها

- All-Pairs Shortest Paths
  - Definition
  - Using single source shortest paths
- A dynamic-programming algorithm based on matrix multiplication
  - *Step m : Paths with at most m edges*
- The Floyd-Warshall algorithm
  - *Step k: Paths with intermediate vertices 1 to k*

مبحث All Pairs Shortest Path از فصل ۲۵ کتاب CLRS تدریس می شود.

All-Pairs Shortest Paths

the problem of finding shortest paths between all pairs of vertices in a graph.

Problem

we are given **a weighted**, **directed** graph *G = (V, E)*
with a weight function $\boldsymbol{w} : \boldsymbol{E} \rightarrow \boldsymbol{R}$ that maps edges to real-valued weights.

We wish to find, for **every pair** of vertices $u, v \in V$, **a shortest (least-weight) path** from *u* to *v*, where the weight of a path is the sum of the weights of its constituent edges.

We typically want the **output in tabular** form:

the entry in u's row and v's column should be the weight of a shortest path from u to v.

Solve by SSP
(use Bellman-Ford and Dijkstra's algorithms)

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.

- If all edge weights are nonnegative
  - we can use Dijkstra's algorithm.
    - min-priority queue : the running time is $O(V^3 + V\,E) = O(V^3)$.
    - binary min-heap : the running time of $O(V\,E\lg V)$,
    - Fibonacci heap : the running time of $O(V^2\lg V + V\,E)$.
- If negative-weight edges are allowed
  - we must run the slower Bellman-Ford algorithm
    - The resulting running time is $O(V^2 E)$,

Be noted

- Unlike the single-source algorithms, which assume an adjacency-list representation of the graph, most of the algorithms in this topic (All-Pairs Shortest Paths) use an **adjacency-matrix** representation.

Assumption

we assume that the **vertices are numbered $1, 2, \ldots, |V|$,** so that the input is an $n \times n$ matrix $W = (w_{ij})$ **representing the edge weights** of an n-vertex directed graph $G = (V, E)$.

- $w_{ij} =$

$$\begin{cases} \mathbf{0} & \text{if } i = j, \\ \textbf{weight of directed edge}(\boldsymbol{i}, \boldsymbol{j}) & \text{if } i \neq j \text{ and } (i, j) \in E \\ \infty & \text{if } i \neq j \text{ and } (i, j) \notin E \end{cases}$$

Output: $\mathbf{D}$ and $\mathbf{\Pi}$

- The **tabular output** of the all-pairs shortest-paths algorithms presented in this chapter is **an $\boldsymbol{n} \times \boldsymbol{n}$ matrix** $\boldsymbol{D} = (d_{ij})$,
- where entry d_{ij} contains the weight of a shortest path from vertex i to vertex j .
- If we let $\boldsymbol{\delta(i, j)}$ denote the shortest path weight from vertex i to vertex j, then

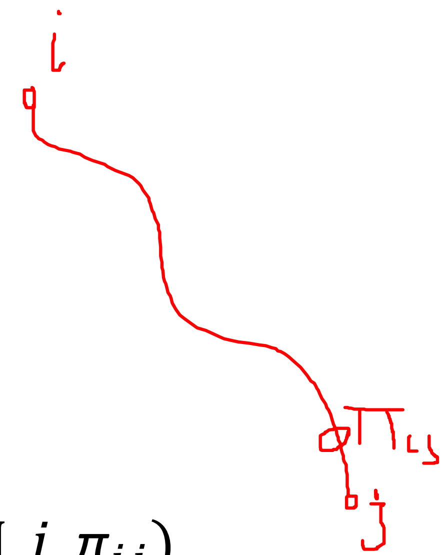
$$\boldsymbol{d_{ij} = \delta(i, j)} \text{ at termination.}$$

- To solve the all-pairs shortest-paths problem on an input adjacency matrix, we need to compute not only the shortest-path weights 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.

Print a path

(from i to j based on matrix *predecessor*)

```
PRINT-ALL-PAIRS-SHORTEST-PATH(Π, i, j )
if i = j
      then print i
      else if π_ij = NIL
            then print no path from i to j exists
            else PRINT-ALL-PAIRS-SHORTEST-PATH(Π, i, π_ij)
                  print j
```



A dynamic-programming algorithm based on matrix multiplication

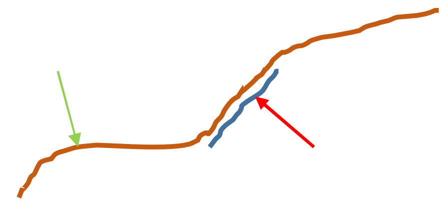
For the all-pairs shortest-paths problem

Dynamic-Programming

the steps of a dynamic-programming algorithm

1. **Characterize the structure of an optimal solution.**
2. Recursively define the value of an optimal solution.
3. Compute the value of an optimal solution in a bottom-up fashion.
4. Construct an optimal solution from computed information.

The structure of a shortest path



- **All subpaths of a shortest path are shortest paths**
- Consider a shortest path p from vertex i to vertex j and suppose that p contains at most m edges.
  - Assuming that there are no negative-weight cycles, m is finite.

- For path p
  - If $i = j$, then p has weight 0 and no edges.
  - If vertices i and j are distinct, then **we decompose path p into** $\boxed{i \overset{p'}{\leadsto} k \rightarrow j}$

- p' is a shortest path from i to k, and so $\boldsymbol{\delta(i, j) = \delta(i, k) + w_{kj}}$.
  (p' now contains at most $m - 1$ edges)

the steps of a dynamic-programming algorithm

1. Characterize the structure of an optimal solution.
2. **Recursively define the value of an optimal solution**.
3. Compute the value of an optimal solution in a bottom-up fashion.
4. Construct an optimal solution from computed information.

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

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

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

(use m-1 edges or m edges?)

$$\boxed{i \overset{p'}{\leadsto} k \to j}$$

$$
\begin{aligned}
l_{ij}^{(m)} &= \min\left(l_{ij}^{(m-1)}, \min_{1 \le k \le n} \left\{l_{ik}^{(m-1)} + w_{kj}\right\}\right) \\
&= \min_{1 \le k \le n} \left\{l_{ik}^{(m-1)} + w_{kj}\right\} .
\end{aligned}
$$

the steps of a dynamic-programming algorithm

1. Characterize the structure of an optimal solution.
2. Recursively define the value of an optimal solution.
3. **Compute the value of an optimal solution in a bottom-up fashion.**
4. Construct an optimal solution from computed information.

Computing the shortest-path weights bottom up

extend path

با استفاده از کوتاهترین میسرها به طول **m-1**، کوتاهترین مسیرها به طول **m** را محاسبه کنیم
الگوریتم ارایه شده در زیر، به عنوان *زیرالگوریتم* استفاده خواهد شد.

EXTEND-SHORTEST-PATHS(L, W)

$n \leftarrow rows[L]$
let $L' = (l'_{ij})$ be an $n \times n$ matrix
for $i \leftarrow 1$ **to** n
    do for $j \leftarrow 1$ **to** n
        do $l'_{ij} \leftarrow \infty$
            for $k \leftarrow 1$ **to** n
                do $l'_{ij} \leftarrow \min(l'_{ij}, l_{ik} + w_{kj})$
return L'

extending shortest paths edge by edge

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

All-Pairs Shortest Paths algorithm

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

1. $n \leftarrow rows[W]$
2. $L^{(1)} \leftarrow W$
3. **for** $m \leftarrow 2$ ***to*** $n - 1$
4. **Do** $L^{(m)} \leftarrow$ EXTEND-SHORTEST-PATHS($L^{(m-1)}, W$)
5. **return** $L^{(n-1)}$

$\boldsymbol{Time\ complexity\ of\ computing\ L^{(n-1)}:\ \Theta(n^4)}$

Computing the shortest-path weights bottom up

similarity to matrix multiplication

EXTEND-SHORTEST-PATHS(L, W)

$n \leftarrow rows[L]$

let $L' = (l'_{ij})$ be an $n \times n$ matrix

for $i \leftarrow 1$ **to** n

  do for $j \leftarrow 1$ **to** n

    do $l'_{ij} \leftarrow \infty$

      for $k \leftarrow 1$ **to** n

        do $l'_{ij} \leftarrow \min(l'_{ij}, l_{ik} + w_{kj})$

return L'

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

Improving the running time

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

$\Theta(n^3 \lg n)$ algorithm
with technique of *repeated squaring*.

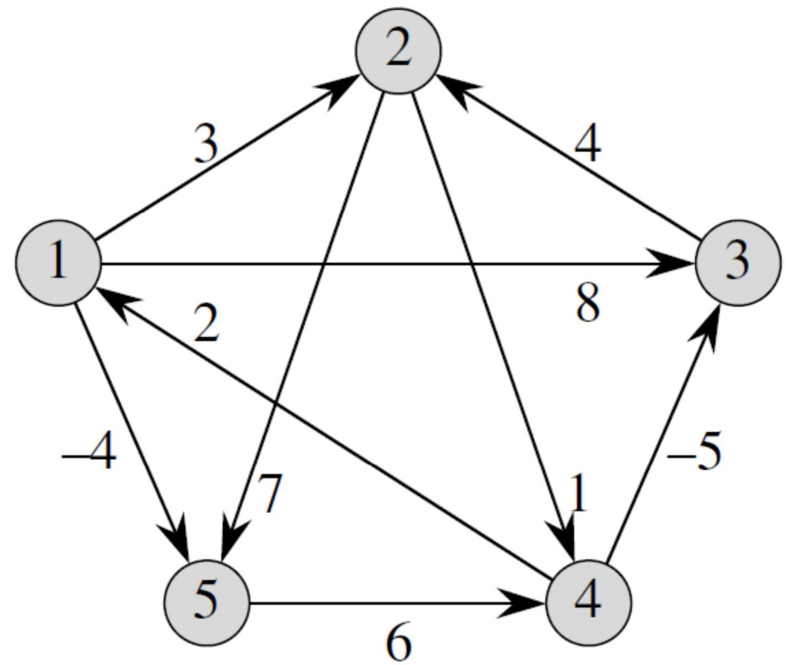
FASTER-ALL-PAIRS-SHORTEST-PATHS(W)

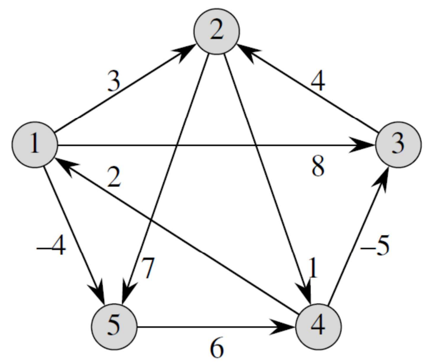
$n \leftarrow rows[W]$
$L^{(1)} \leftarrow W$
$m \leftarrow 1$
while $m < n - 1$
    do $L^{(2m)} \leftarrow$ EXTEND-SHORTEST-PATHS$(L^{(m)}, L^{(m)})$
        $m \leftarrow 2m$
return $L^{(m)}$

example

$$L^{(0)} = \begin{bmatrix} 0 & \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix}$$

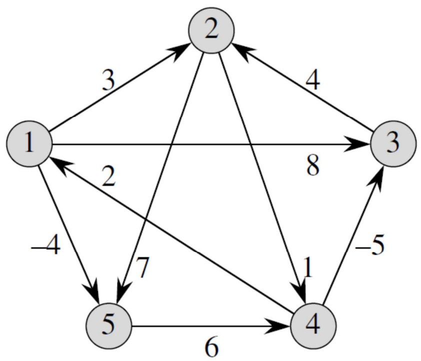
$$L^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$





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

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



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

(=W)

2

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

5

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

Sample problems:

- Modify FASTER-ALL-PAIRS-SHORTEST-PATHS so that it can detect the presence of a negative-weight cycle.

- Give an efficient algorithm to find the length (number of edges) of a minimum length negative-weight cycle in a graph.
  - Hint: consider the diagonal of matrix

The Floyd-Warshall algorithm

Use dynamic-programming

negative-weight edges may be present,

but we assume that there are no negative-weight cycles

The structure of a shortest path

definition + assumption

- An ***intermediate*** vertex of a simple path *p* is any vertex of *p* other than *start* and *destination*
  - 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, . . . , *k*},** and let p be a minimum-weight path from among them.

مسیری را فرض می کنیم که راس های میانی فقط از راس های ۱ تا k انتخاب شده باشند
میخواهیم به صورت بازگشتی این مسیر را از زیرمسیرهایی که فقط از ۱ تا k-1 انتخاب شده باشند، پیدا کنیم

The structure of a shortest path relationship

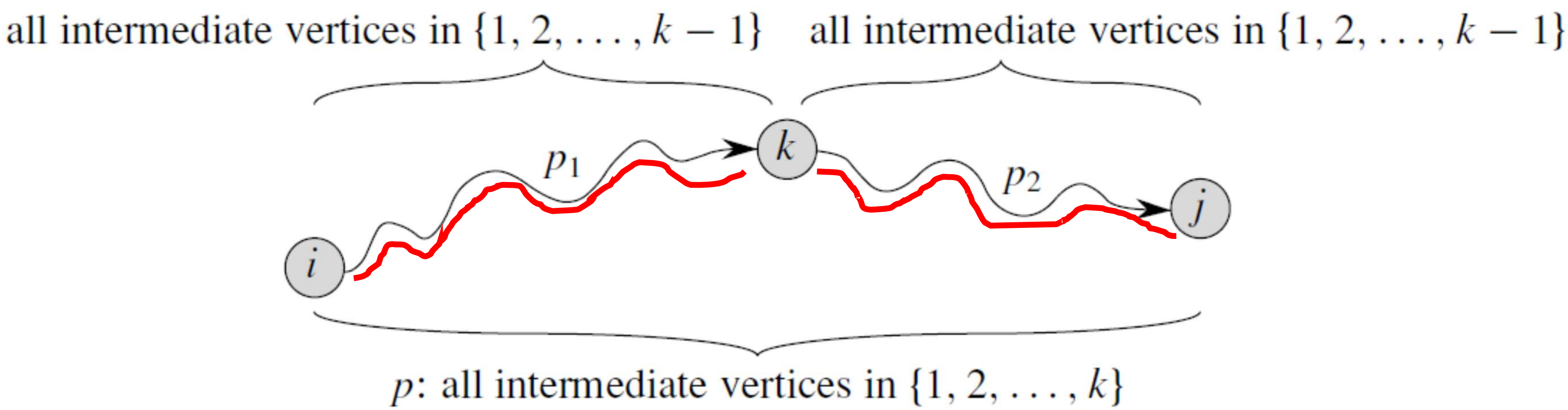
- **whether or not** *k* is an intermediate vertex of path *p*
  - *Yes*
  - *No*

❖If *k* **is not** an intermediate vertex of path *p*

- all intermediate vertices of path *p* are in the set $\{1, 2, \ldots, k - 1\}$.

❖If *k* **is** an intermediate vertex of path *p*

- we **break *p* down into** paths **p1 from *i to k*** *and path* ***p*2 from k to *j***
- *p1* is a shortest path from *i* to *k* with all intermediate vertices in the set $\{1, 2, \ldots, k - 1\}$.
- *p*2 is a shortest path from vertex *k* to vertex *j* with all intermediate vertices in the set $\{1, 2, \ldots, k - 1\}$.



**قسمت های مشخص شده با خط قرمز فقط از راس های ۱ تا k-1 عبور کرده است
و زیرمسیرهایی برای حل بازگشتی مساله مشخص کرده ایم.**

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

$$d_{ij}^{(0)} = w_{ij}$$

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

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

عدد مشخص شده در پرانتز بالای d نشان دهنده ی این است که از راس ۱ تا چه راسی در ساخت مسیر استفاده شده است!

Computing the shortest-path weights bottom up

FLOYD-WARSHALL(W)

$n \leftarrow rows[W]$
$D^{(0)} \leftarrow W$
for $k \leftarrow 1$ **to** n
    do for $i \leftarrow 1$ **to** n
        do for $j \leftarrow 1$ **to** n
            do $d_{ij}^{(k)} \leftarrow \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$
return $D^{(n)}$

Example

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

result

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

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

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

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

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

Time complexity

- The Floyd-Warshall algorithm runs in time $\Theta(n3)$
- Code is tight, with no elaborate data structures, and so the constant hidden in the Θ-notation is small.
- The Floyd-Warshall algorithm is quite practical for even moderate-sized input graphs.

Constructing a shortest path

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

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

Transitive closure of a directed graph

- Given a directed graph $G = (V, E)$ with vertex set $V = \{1, 2, \ldots, n\}$, we may wish to find out
  - whether there is **a path in G from i to j** for all vertex pairs $i, j \in V$.
- The ***transitive closure*** of G*:*
  - *The edge* (i, j) *means that* there is a path from vertex i to vertex j in G
- **A recursive definition:**

$$t_{ij}^{(0)} = \begin{cases} 0 & \text{if } i \neq j \text{ and } (i, j) \notin E \, , \\ 1 & \text{if } i = j \text{ or } (i, j) \in E \, , \end{cases}$$

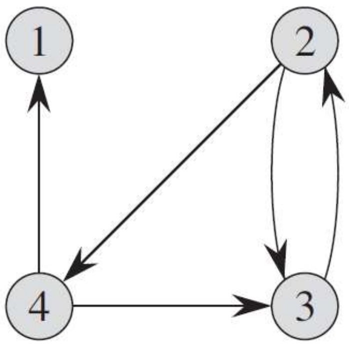
and for $k \geq 1$,

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}\right) \, .$$

Transitive closure
By using Floyd-Warshall

TRANSITIVE-CLOSURE(G)

$n \leftarrow |V[G]|$
for $i \leftarrow 1$ **to** n
    do for $j \leftarrow 1$ **to** n
        do if $i = j$ or $(i, j) \in E[G]$
            then $t_{ij}^{(0)} \leftarrow 1$
            else $t_{ij}^{(0)} \leftarrow 0$
for $k \leftarrow 1$ **to** n
    do for $i \leftarrow 1$ **to** n
        do for $j \leftarrow 1$ **to** n
            do $t_{ij}^{(k)} \leftarrow t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}\right)$
return $T^{(n)}$



$$T^{(0)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

$$T^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad T^{(4)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Sample problems:

- How can the output of the Floyd-Warshall algorithm be used to detect the presence of a negative-weight cycle?

- Give an $O(V\ E)$-time algorithm for computing the transitive closure of a directed graph $G = (V, E)$.