All-Pairs Shortest Paths

The content of this presentation is a collection from various books and online resources, such as Introduction to Algorithms, by CLRS and the lecture notes of Dr. Kai and others.
Thanks to all for their valuable contributions.

All-Pairs Shortest Paths

- We consider the problem of finding shortest paths between all pairs of vertices in a graph.
- We are given a weighted, directed graph $G(V, E)$ with a weight function $w: E \rightarrow R$ 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.
- 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.

All-Pairs Shortest Paths

- If there are no negative cost edges, then we can apply Dijkstra's algorithm to each vertex (as the source) of the digraph.
- Recall that Dijkstra's algorithm runs in $\mathrm{O}((V+E)\log V)$, if implemented using binary heap.
- Thus, to find all pair shortest paths, $\mathrm{O}(V(V+E)\log V) = O(V^2 \log V + VE \log V)$.
- If the digraph is dense (i.e., the number of edges in G is close to the maximal number of edges $\binom{|V|}{2}$), then the complexity becomes $\mathrm{O}(V^3 \log V)$ algorithm.
- However, if we implement the Dijkstra's algorithm using Fibonacci heap the total running time = $O(V \lg V + E)$. Then the complexity of implementing all pair shortest paths for a dense graph becomes $\mathrm{O}(V^3)$.

All-Pairs Shortest Paths

- Similarly, if the graph has negative-weight edges, we must run the slower Bellman-Ford algorithm once from each vertex
- The resulting running time is $O(V^2E)$, which on a dense graph is $O(V^4)$.
- We shall see how to do better

All-Pairs Shortest Paths: Dynamic Programming

- So the questions that we need to ask:
  - How do we decompose the all-pairs shortest paths problem into subproblems?
  - How do we express the optimal solution of a subproblem in terms of optimal solutions to some sub-subproblems?
  - How do we use the recursive relation from (2) to compute the optimal solution in a bottom-up fashion?
  - How do we construct all the shortest paths?

The structure of an optimal solution

- Consider a shortest path p from vertex i to vertex j, and suppose that p contains at most m edges.
- We assume that there are no negative-weight cycles.
  - Hence $m \leq n - 1$ is finite.
- If $i = j$, then p has weight 0 and no edge
- If $i \neq j$, we decompose p into $i \overset{p\prime}{\to} k \to j$, where p' contains at most $m - 1$ edges.
- Moreover, p' is a shortest path from i to k and $\delta(i, j) = \delta(i, k) + w_{kj}$, where $\delta(i, j)$ dente the shortest weight path from i to j

Recursive solution to the all-pairs shortest-path problem

- let $l_{ij}^{(m)}$ be the minimum weight of any path from vertex i to vertex j that contains at most m edges.
- When $m = 0$, there is a shortest path from i to j with no edges if and only if $i = j$. Thus

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

- For $m \geq 1$, we compute $l_{ij}^{(m)}$ as the minimum of $l_{ij}^{(m-1)}$ (the weight of a shortest path from i to j consisting of at most $m - 1$ edges) and the minimum weight of any path from i to j consisting of at most m edges, obtained by looking at all possible predecessors k of j.

Recursive solution to the all-pairs shortest-path problem

- We have two cases:
- Consider a shortest path from i to j of length $l_{ij}^{(m)}$.
  - The shortest path from i to j has at most $(m-1)$ edges. In that case, we have $l_{ij}^{(m)} = l_{ij}^{(m-1)} = l_{ij}^{(m-1)} + w_{jj}$.
  - The shortest path from i to j has (m) edges. Let k be the vertex before j on a shortest path. Then, $l_{ij}^{(m)} = l_{ik}^{(m-1)} + w_{kj}$.
- So, combining the two cases, we have:

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

Recursive solution to the all-pairs shortest-path problem

- Thus, we recursively define

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

- Since shortest path from i to j contains at most $n-1$ edges,

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

Computing the shortest-path weight bottom-up.

- Taking input matrix $W = (w_{ij})$, compute $L^{(1)}, L^{(2)}, \ldots, L^{(n-1)}$ where $L^{(m)} = (l_{ij}^{(m)})$ for all i and j. Observe that $l_{ij}^{(1)} = w_{ij}$ for all vertices $i, j \in V$, and so $L^{(1)} = W$

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

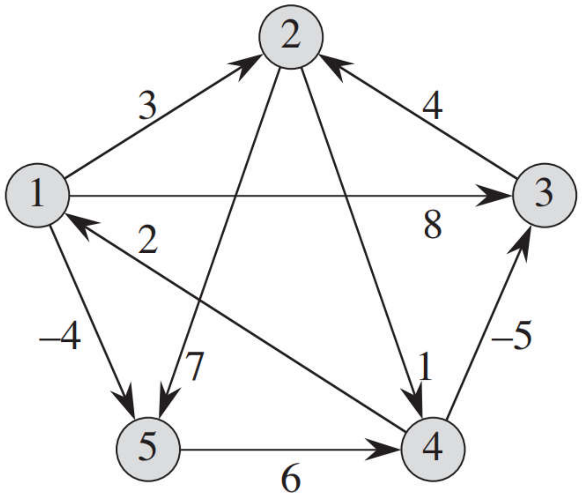
- In the following, given matrices $L^{(m-1)}$ and W, returns the matrix $L^{(m)}$.
- That is, it extends the shortest paths computed so far by one more edge.
- The procedure computes a matrix $L' = (l'_{ij})$, which it returns at the end.
- It does so by computing equation for all i and j, using L for $L^{(m-1)}$ and L' for $L^{(m)}$.
- Its running time is $\theta(n^3)$ due to the three nested **for** loops.

Computing the shortest-path weight bottom-up.

```
SLOW-ALL-PAIRS-SHORTEST-PATHS(W)
n = W.rows
L^(1) = W
for m = 2 to n - 1
    let L^(m) be a new n × n matrix
    L^(m) = EXTEND-SHORTEST-PATHS(L^(m-1), W)
return L^(n-1)
```

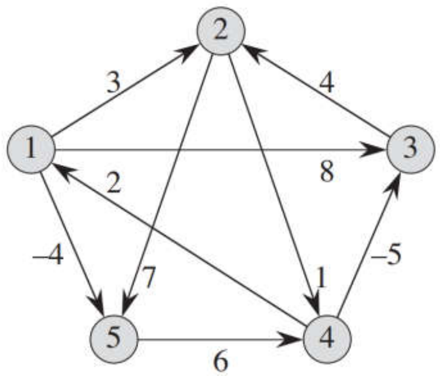
- The following procedure computes this sequence $\left(L^{(0)}, L^{(1)}, \dots, L^{(n-1)}\right)$ in $\theta(n^4)$ time.

Computing the shortest-path weight bottom-up.



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

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



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

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

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

$$l_{42}^{(2)} = \min\left(l_{42}^{(1)}, \min\left\{l_{4k}^{(1)} + w_{k2}\right\}\right)$$
$$= \min(\infty, min\{5, \infty, -1, \infty, \infty\}) = -1$$

To calculate SP from 4 to 2 with two edges, i.e., $l_{42}^{(2)}$ -> we find the min cost of the (path from 4 to node k with one edge) and (cost of k to 2)

Computing the shortest-path weight bottom-up.

- Notice that the above computation is very similar to matrix multiplication.

- That is, if we wish to compute $C = A.B$ of two $n \times n$ matrices A and B. Then, for $i, j = 1, 2, \ldots\ , n$, we compute:

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

- Observe that if we make the following substitutions (in the pseudocode), then we get the above equation for matrix multiplication.

$$l^{m-1} \to a, \qquad w \to b, \qquad l^{m} \to c, \qquad min \to +, \qquad + \to .$$

- Thus, if we make these changes to EXTEND-SHORTEST-PATHS and also replace ∞ (the identity for min) by 0 (the identity for $+$), we obtain the same $\theta(n^3)$-time procedure for multiplying square matrices.

Computing the shortest-path weight bottom-up.

- Letting $A \,.\, B$ denote the matrix "product" returned by EXTEND-SHORTEST-PATHS(A, B), we compute the sequence of $n-1$ matrices

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

Improving the running time

- The following procedure computes the above sequence of matrices by using this technique of ***repeated squaring***

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

- Since $2^{\lceil \lg(n-1) \rceil} \geq n-1$, the final product $L^{2^{\lceil \lg(n-1) \rceil}}$ is equal to $L^{(n-1)}$.
- Therefore, we can compute $L^{(n-1)}$ with only $\lceil \lg(n-1) \rceil$ matrix products.

Improving the running time

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

$n = W.rows$
$L^{(1)} = W$
$m = 1$
while $m < n - 1$
  let $L^{(2m)}$ be a new $n \times n$ matrix
  $L^{(2m)} =$ EXTEND-SHORTEST-PATHS$(L^{(m)}, L^{(m)})$
  $m = 2m$
return $L^{(m)}$

- Because each of the $\lceil \lg (n-1) \rceil$ matrix products takes $\theta(n^3)$ time, FASTERALL-PAIRS-SHORTEST-PATHS runs in $\theta\left(n^3 \lg n\right)$ time.

Floyd-Warshall algorithm

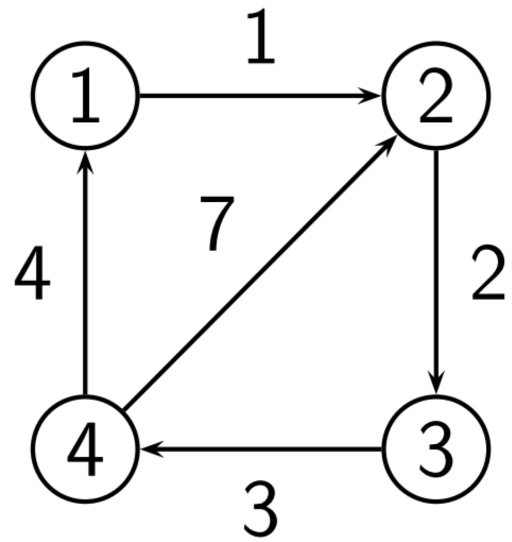
- We shall use a different dynamic-programming formulation to solve the all-pairs shortest-paths problem on a directed graph $G(V, E)$.
- The resulting algorithm, known as the *Floyd-Warshall algorithm*, runs in $\Theta(V^3)$ time.
- As before, negative-weight edges may be present, but we assume that there are no negative-weight cycles.
- The Floyd-Warshall algorithm considers the intermediate vertices of a shortest path, where an *intermediate* vertex of a simple path $p = < v_1, v_2, \dots, v_n >$ is any vertex of p other than v_1 or v_2.

Floyd-Warshall algorithm

- Let $V = \{1,2, \dots, n\}$. For any pair of vertices $i, j \in V$, consider all paths from i to j whose intermediate vertices are drawn from $\{1, 2, \dots, k\}$, and let p be a minimum weight path among them.
- The relationship depends on whether or not k is an intermediate vertex of path p.
- If k is not an intermediate vertex of path p, then all intermediate vertices of path p are in the set $\{1, 2, \dots, k\}$.
- If k is an intermediate vertex of path p, then we can decompose p into $i \overset{p_1}{\rightarrow} k \overset{p_2}{\rightarrow} j$.
- Thus, p_1 is a shortest path from i to k with all intermediate vertices in the set $\{1, 2, \dots, k\}$.
  - In other words, since vertex k is not an intermediate vertex of path p_1, all intermediate vertices of p_1 are in the set $\{1, 2, \dots, k\}$.
- Similarly, for p_2.

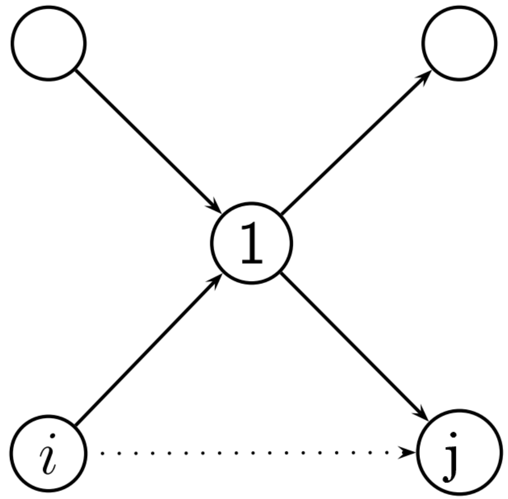
Floyd-Warshall algorithm

- In simple terms,
  - In each iteration, we ask… do we have a shortest path between i and j, with k as an intermediate vertex?
  - For example, consider the following figure at the top.
  - In $d_{ij}^{(0)}$, we ask what is the SP between i and j with no intermediate vertex, i.e., a path has at most one edge.
    - So, we see that $d_{12}^{(0)} = 1$, $d_{23}^{(0)} = 2$, …
    - That is, $d_{ij}^{(0)} = w_{ij}$. As discussed above.



Floyd-Warshall algorithm

- In $d_{ij}^{(1)}$, we ask do we have a SP between i and j with node 1 as an intermediate vertex? Two cases:
  - If $d_{ij}^{(0)} \leq d_{i1}^{(0)} + d_{1j}^{(0)}$, the $d_{ij}^{(0)} = d_{ij}^{(1)}$ remains unchanged.
  - However, if $d_{ij}^{(0)} > d_{i1}^{(0)} + d_{1j}^{(0)}$, then $d_{ij}^{(1)}$ is updated by the sum, as $d_{ij}^{(1)} = d_{i1}^{(0)} + d_{1j}^{(0)}$.



- Similarly, we find $d_{ij}^{(2)}$. That is, we check if we have a SP between i and j with node 2 as an intermediate vertex? Again, we will have two cases:
  - If $d_{ij}^{(1)} \leq d_{i2}^{(1)} + d_{2j}^{(1)}$, the $d_{ij}^{(2)} = d_{ij}^{(1)}$ remains unchanged.
  - However, if $d_{ij}^{(1)} \leq d_{i2}^{(1)} + d_{2j}^{(1)}$, then $d_{ij}^{(2)} = d_{i2}^{(1)} + d_{2j}^{(1)}$.

Floyd-Warshall algorithm

- Let $d_{ij}^{(k)}$ be the weight of a shortest path from vertex i to vertex j for which all intermediate vertices are in the set $\{1, 2, \dots, k\}$.
- When $k = 0$, a path from vertex i to vertex j no intermediate vertices at all.
- Such a path has at most one edge, and hence $d_{ij}^{(0)} = w_{ij}$.
- Following the above discussion, we define $d_{ij}^{(k)}$ recursively as:

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

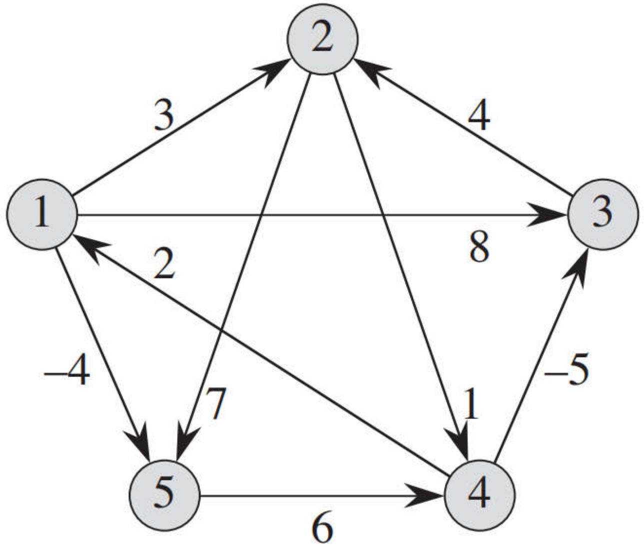
Floyd-Warshall algorithm

FLOYD-WARSHALL(W)

$n = W.rows$
$D^{(0)} = W$
for $k = 1$ **to** n
    let $D^{(k)} = \left(d_{ij}^{(k)}\right)$ be a new $n \times n$ matrix
    for $i = 1$ **to** n
        for $j = 1$ **to** n
            $d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$
return $D^{(n)}$

The algorithm runs in time $\theta(n^3)$.

Compute $D^{(1)}$



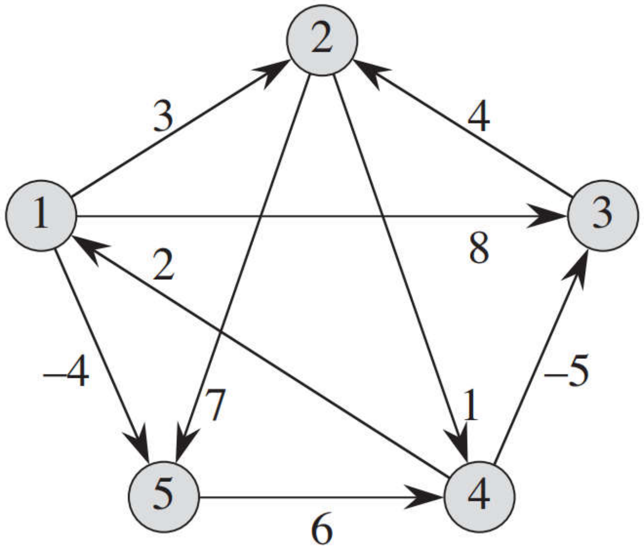
$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

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

- To calculate $D^{(1)}$ from $D^{(0)}$, via intermediate node 1.
  - No change in the first row and first column.
  - Diagonals will remain 0.

- $d_{23}^{(1)} = \min\left(d_{23}^{(0)}, d_{21}^{(0)} + d_{13}^{(0)}\right) = \min(\infty, \infty + 8) = \infty$
- $d_{32}^{(1)} = \min\left(d_{32}^{(0)}, d_{31}^{(0)} + d_{12}^{(0)}\right) = \min(4, \infty + 3) = 4$
- $d_{42}^{(1)} = \min\left(d_{42}^{(0)}, d_{41}^{(0)} + d_{12}^{(0)}\right) = \min(\infty, 2 + 3) = 5$
- $d_{45}^{(1)} = \min\left(d_{45}^{(0)}, d_{41}^{(0)} + d_{15}^{(0)}\right) = \min(\infty, 2 - 4) = -2$

Compute $D^{(2)}$



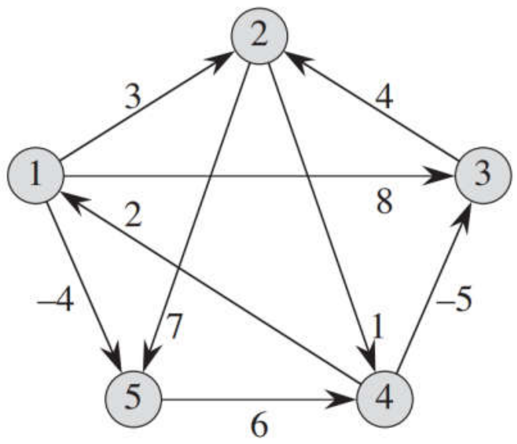
$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

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

- To calculate $D^{(2)}$ from $D^{(1)}$, via intermediate node 2.
  - No change in the 2nd row and 2nd column.
  - Diagonals will remain 0.

- $d_{13}^{(2)} = \min\left(d_{13}^{(1)}, d_{12}^{(1)} + d_{23}^{(1)}\right) = \min(8, 3 + \infty) = 8$
- $d_{14}^{(2)} = \min\left(d_{14}^{(1)}, d_{12}^{(1)} + d_{24}^{(1)}\right) = \min(\infty, 3 + 1) = 4$
- $d_{15}^{(2)} = \min\left(d_{15}^{(1)}, d_{12}^{(1)} + d_{25}^{(1)}\right) = \min(-4, 3 + 7) = -4$
- $d_{34}^{(2)} = \min\left(d_{34}^{(1)}, d_{32}^{(1)} + d_{24}^{(1)}\right) = \min(\infty, 4 + 1) = 5$
- $d_{35}^{(2)} = \min\left(d_{35}^{(1)}, d_{32}^{(1)} + d_{25}^{(1)}\right) = \min(\infty, 4 + 7) = 11$

Floyd-Warshall algorithm



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

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

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

π: is the predecessor matrix.

Floyd-Warshall algorithm

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

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

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