Algorithms

Kiho Choi

Fall, 2022

School of computing
Gachon University

5. Graph Algorithms III

- The Bellman-Ford algorithm: Single-Source Shortest Paths
- The Floyd-Warshall algorithm: All-Pairs Shortest Paths

Contents

- Single-source shortest paths
  - Bellman-Ford algorithm
- All Pairs Shortest Paths
  - Matrix multiplication
  - Floyd-Warshall algorithm

The Bellman-Ford Algorithm

- Dijkstra’s algorithm doesn’t work when there are negative edges:
  - Intuition – we can not be greedy any more on the assumption that the lengths of paths will only increase in the future
- Bellman-Ford algorithm detects negative cycles (returns *false*) or returns the shortest path-tree
  - Returns True if no negative-weight cycles reachable from s, False otherwise.

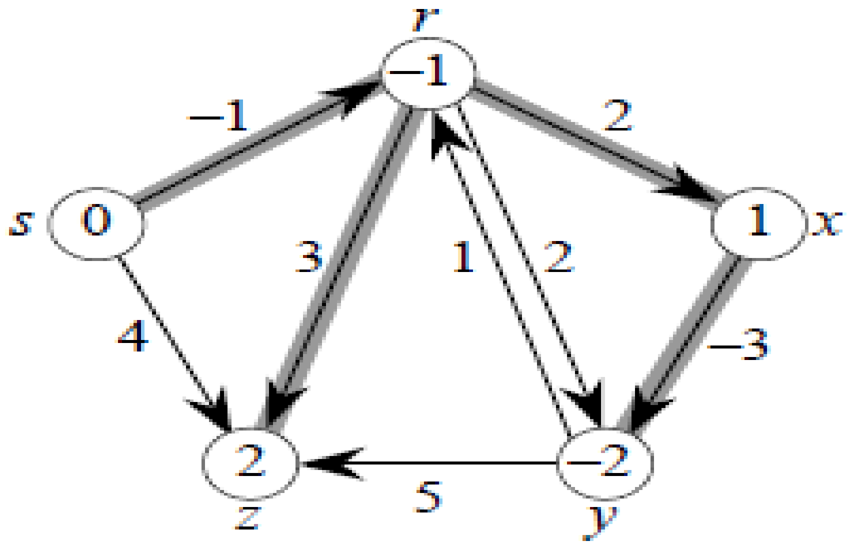
The Bellman-Ford Algorithm (cont'd)

```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i ← 1 to |V[G]| − 1
    do for each edge (u, v) ∈ E[G]
        do RELAX(u, v, w)
for each edge (u, v) ∈ E[G]
    do if d[v] > d[u] + w(u, v)
        then return FALSE
return TRUE
```

- ***Core:*** The first **for** loop relaxes all edges |*V*| -1 times.
- ***Time:*** *Θ (VE).*

The Bellman-Ford Algorithm (cont'd)

- Examples
  - Values you get on each pass and how quickly it converges depends on order of relaxation.
  - But guaranteed to converge after $|V|$ -1 passes, assuming no negative-weight cycles.



The Bellman-Ford Algorithm (cont'd)

Proof Use path-relaxation property.

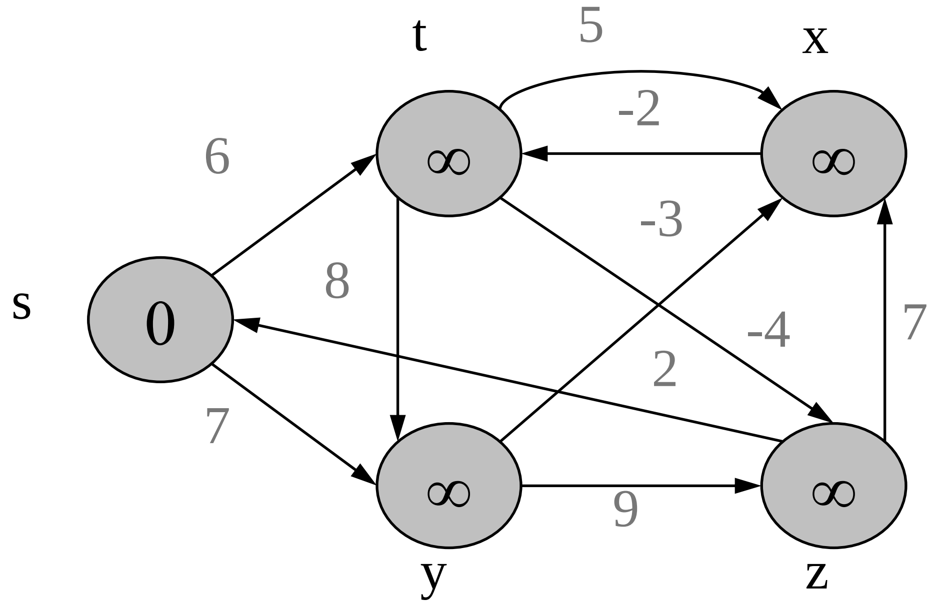
- Let v be reachable from s, and let $p = v_0, v_1, \ldots, v_k$ be a shortest path from s to v, where $v_0 = s$ and $v_k = v$. Since p is acyclic, it has $\leq |V| - 1$ edges, so $k \leq |V| - 1$.

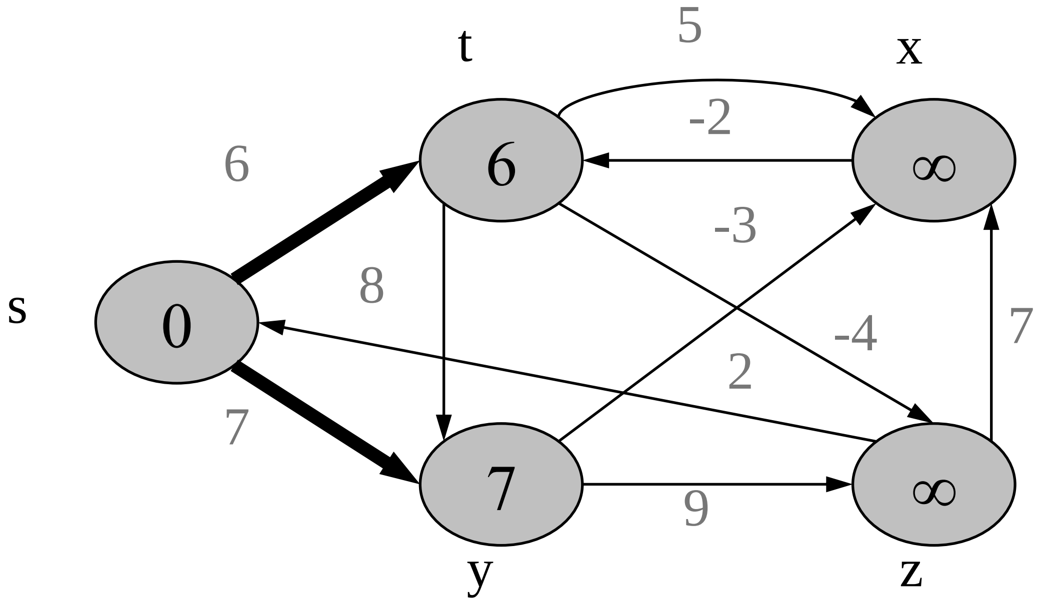
  Each iteration of the for loop relaxes all edges:
  - First iteration relaxes (v_0, v_1).
  - Second iteration relaxes (v_1, v_2).
  - K^{th} iteration relaxes (v_{k-1}, v_k).

  By the path-relaxation property,

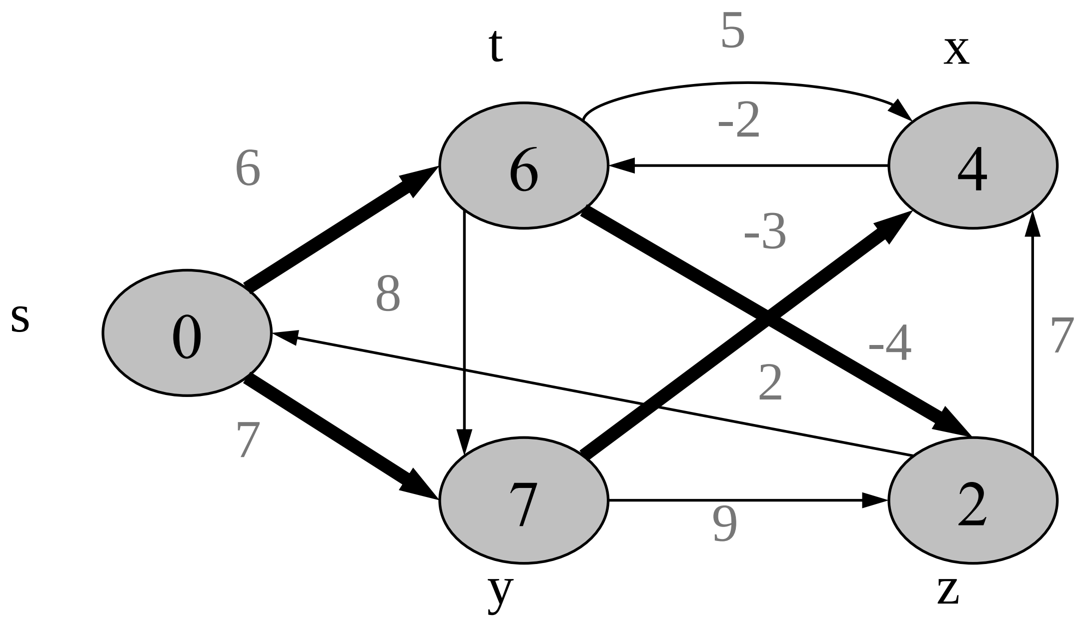
  $d[v] = d[v_k] = \delta(s, v_k) = \delta(s, v)$.

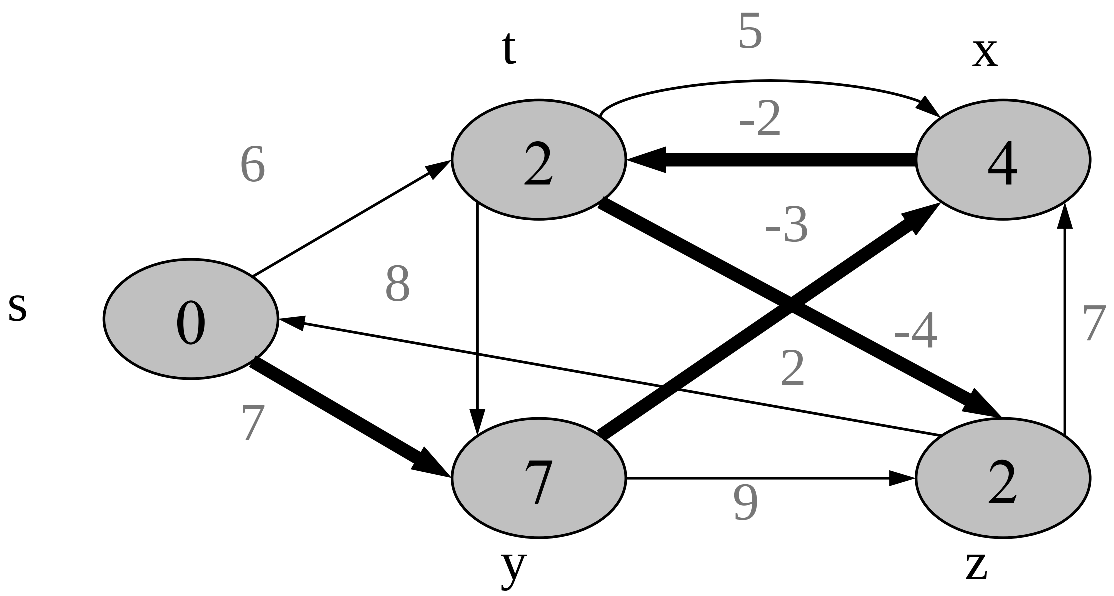
The Bellman-Ford Algorithm (cont'd)



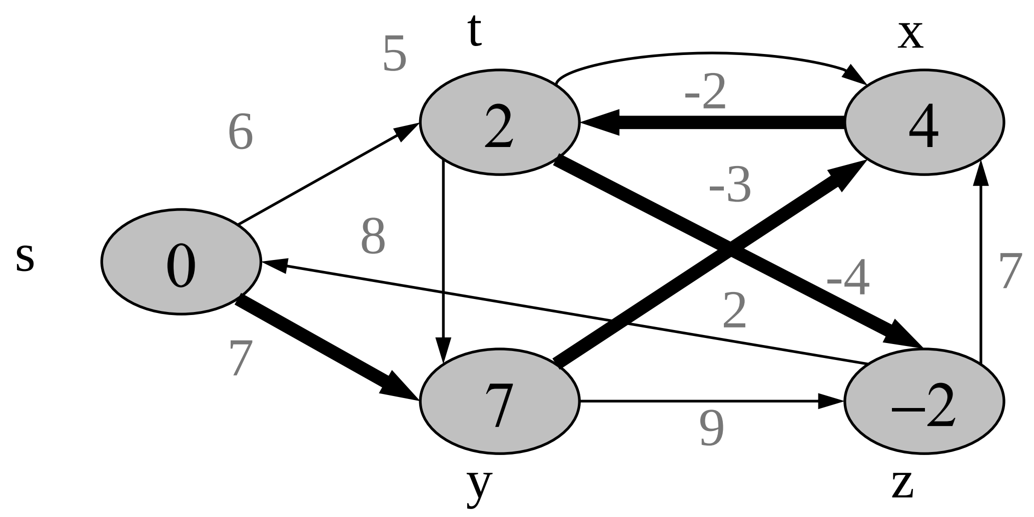


The Bellman-Ford Algorithm (cont'd)





The Bellman-Ford Algorithm (cont'd)



- Bellman-Ford running time:
  - (|V|-1)|E| + |E| = O(VE)

All Pairs Shortest Paths

motivation

- computer network
- aircraft network (e.g. flying time, fares)
- railroad network
- table of distances between all pairs of cities for a road atlas

single source shortest path algorithms

- if edges are non-negative:

running the Dijkstra's algorithm n-times, once for each vertex as the source

  - *running time:* O(nm lg n)

    (using binary-heap)

- negative-weight edges:

  Bellman-Ford algorithm

  - *running time:* O(n^2 m)

All-Pairs Shortest Paths

data structure

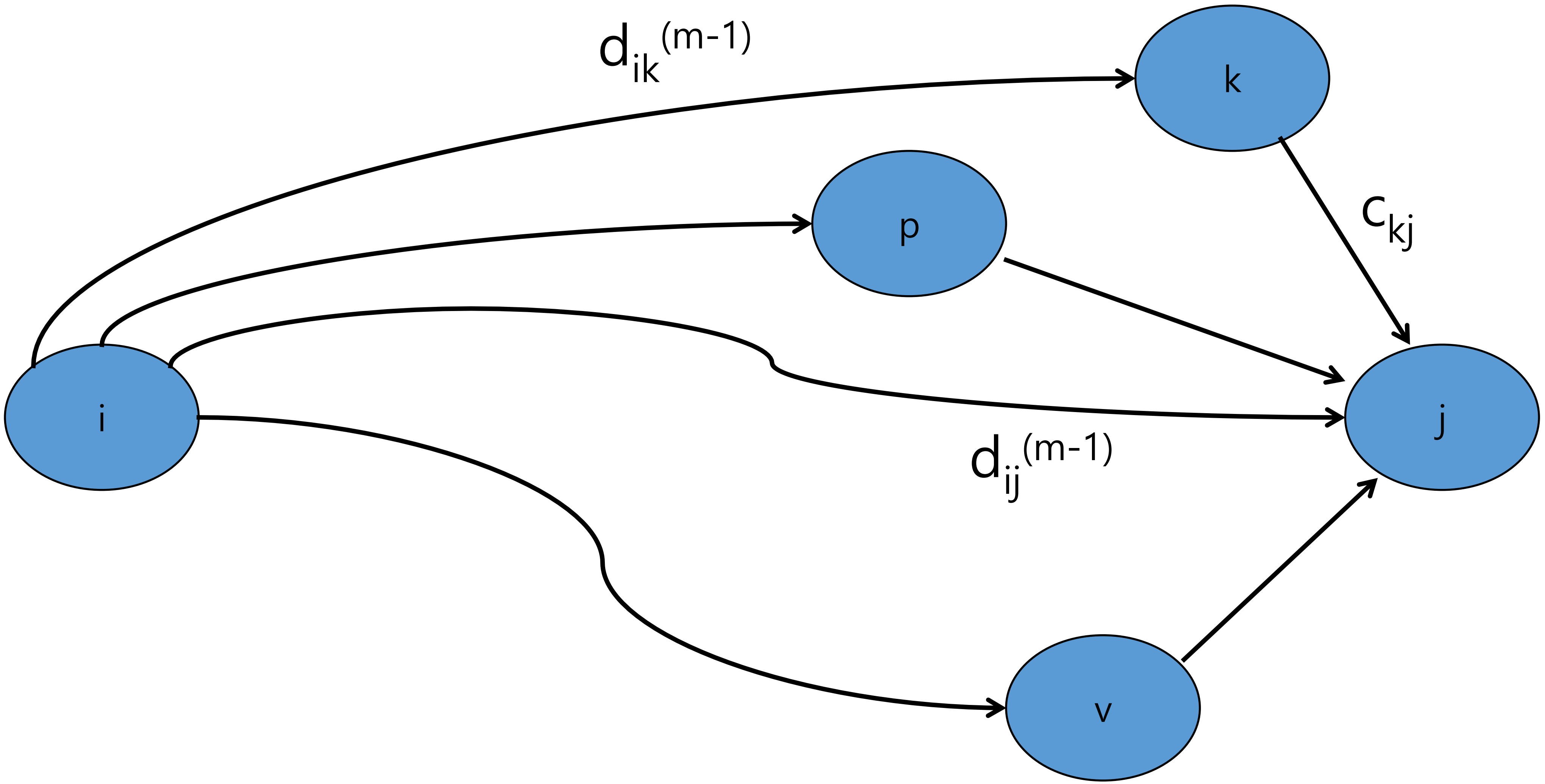
- adjacency matrix
- c: E $\rightarrow \Re$ as n x n matrix C

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

matrix multiplication (idea)

- $d_{ij}^{(m)}$: minimum weight of any path
  from i to j
  that contains at most m edges

matrix multiplication (idea)



matrix multiplication (idea)

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

↳ look at all possible predecessors k of j and compare

matrix multiplication (structure)

$$d_{ij}^{(1)} = c_{ij}$$

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

matrix multiplication (structure)

- Compute a series of matrices

$$D^{(1)}, \ D^{(2)}, \ \ldots, \ D^{(n-1)}$$

$$D^{(m)} = D^{(m-1)} \cdot C$$

final matrix $D^{(n-1)}$ contains the final shortest-path weights

matrix multiplication (pseudo-code)

EXTEND-SHORTEST-PATHS (D, C)

$n \leftarrow rows[D]$

let $D' = (d'_{ij})$ be an n x n matrix

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

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

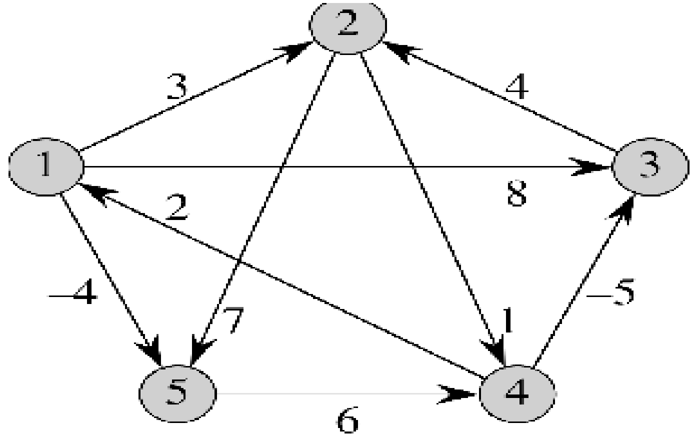
    do $d'_{ij} \leftarrow \infty$

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

      do $d'_{ij} \leftarrow \min(d'_{ij}, d_{ik} + c_{kj})$

return D'

matrix multiplication (example)



$$d_{14}^{(2)} = (0\ \ 3\ \ 8\ \ \infty\ -4) * \begin{pmatrix} \infty \\ 1 \\ \infty \\ 0 \\ 6 \end{pmatrix}$$

$$= \min(\infty, 4, \infty, \infty, 2)$$

$$= 2$$

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

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

Figure 25.1, p690

matrix multiplication (example)

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

$$d_{14}^{(2)} = (0 \;\; 3 \;\; 8 \;\; \infty \;\; -4) * \begin{pmatrix} \infty \\ 1 \\ \infty \\ 0 \\ 6 \end{pmatrix}$$

$$= \min(\infty, 4, \infty, \infty, 2)$$

$$= 2$$

relation to matrix multiplication

$C = A \cdot B$

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

compare:

$D^{(m)} = D^{(m-1)} \cdot C$

$\Rightarrow d_{ij}^{(m)} = \min_{1 \leq k \leq n} \{d_{ik}^{(m-1)} + c_{kj}\}$

matrix multiplication

- compute the sequence of n-1 matrices:

$$D^{(1)} = D^{(0)} \cdot C = C,$$

$$D^{(2)} = D^{(1)} \cdot C = C^2,$$

$$D^{(3)} = D^{(2)} \cdot C = C^3,$$

…

$$D^{(n-1)} = D^{(n-2)} \cdot C = C^{n-1}$$

matrix multiplication (pseudo-code)

ALL-PAIRS-SHORTEST-PATHS (C)

$n \leftarrow rows[C]$

$D^{(1)} \leftarrow C$

for $m \leftarrow 2$ **to** n **-** 1

do $D^{(m)} \leftarrow$ EXTEND-SHORTEST-PATHS ($D^{(m-1)}, C$)

return $D^{(n-1)}$

- *negative cycles:* $d_{ii} < 0$

matrix multiplication (running time)

Improving the running time:

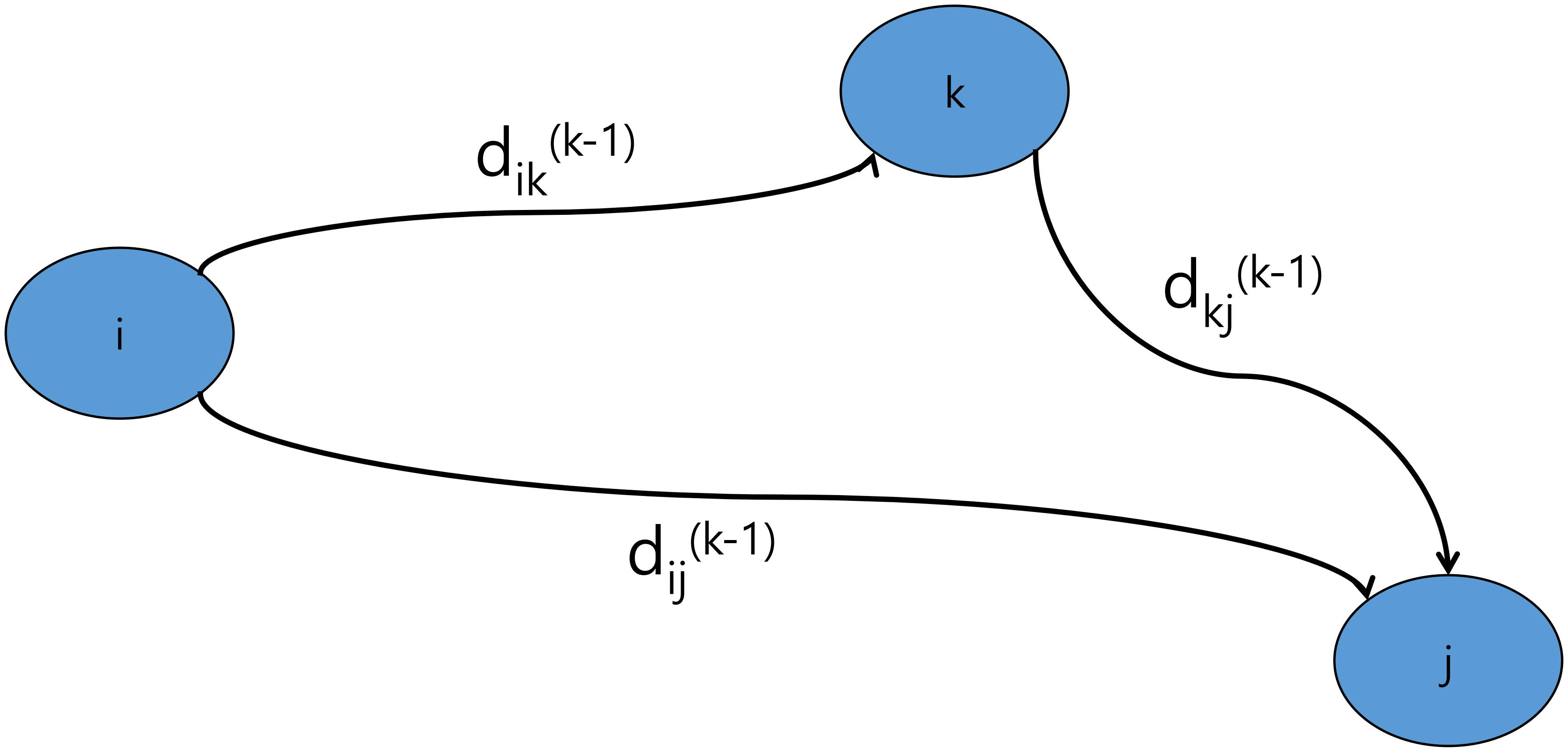
- compute not all $D^{(m)}$ matrices

  interested only in $D^{(n-1)}$, which is $D^{(m)}$ for all integers $m \geq n-1$

The Floyd-Warshall Algorithm

- Graph analysis algorithm for finding shortest paths in a weighted, directed graph
- Graph can have negative weights, but not negative weight cycles
- Example of dynamic programming, a method of solving problems where one needs to find the best decision one after another

Floyd-Warshall algorithm (cont‘d)



Floyd-Warshall algorithm (cont‘d)

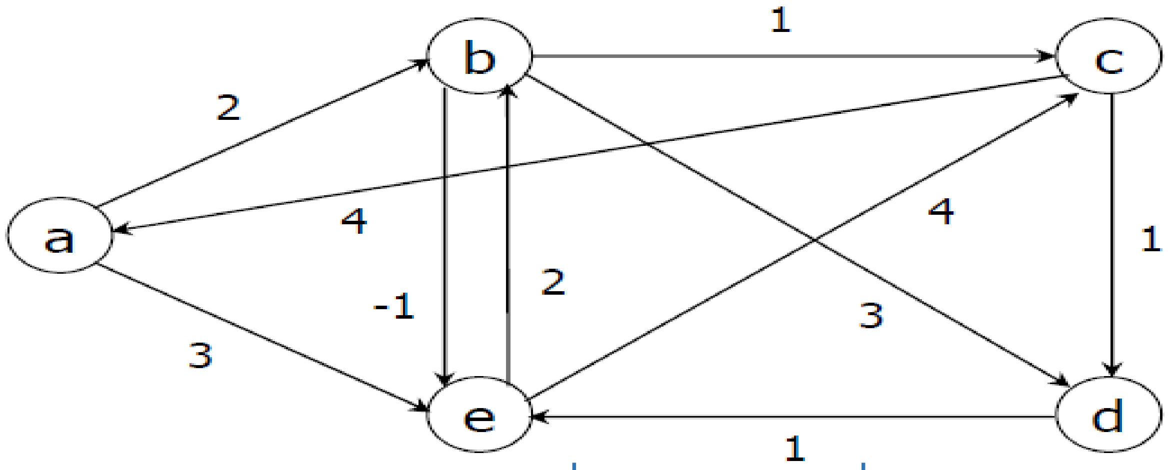
- Consider a graph *G* with vertices *V*, each numbered 1 through *N*.
  - Further consider a function shortestPath(i, j, k) that returns the shortest possible path from i to j using vertices only from the set $\{1,2,...,k\}$ as intermediate points along the way.
  - Now, given this function, our goal is to find the shortest path from each i to each j using only vertices 1 to $k + 1$.

Floyd-Warshall algorithm (cont'd)

- There are two candidates for each of these paths: either the true shortest path only uses vertices in the set $\{1, \ldots, k\}$; or there exists some path that goes from i to $k + 1$, then from $k + 1$ to j that is better.
  - We know that the best path from i to j that only uses vertices 1 through k is defined by shortest path(i, j, k), and it is clear that if there were a better path from i to j, then the length of this path would be the concatenation of the shortest path from i to $k + 1$ (using vertices in $\{1, \ldots, k\}$) and the shortest path from $k + 1$ to j (also using vertices in $\{1, \ldots, k\}$).

Floyd-Warshall algorithm (cont‘d)

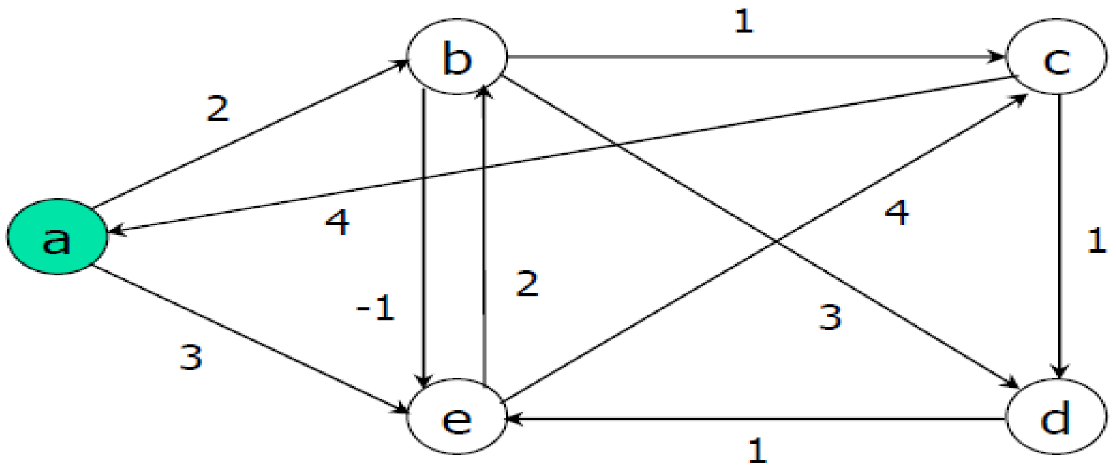
Example : step(1/6), K=0



$$D^{(0)} = \begin{array}{c c} & \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \\ e \end{array} & \begin{bmatrix} 0 & 2 & \infty & \infty & 3 \\ \infty & 0 & 1 & 3 & -1 \\ 4 & \infty & 0 & 1 & \infty \\ \infty & \infty & \infty & 0 & 1 \\ \infty & 2 & 4 & \infty & 0 \end{bmatrix} \end{array}$$

Floyd-Warshall algorithm (cont'd)

Example : step(2/6), k=1

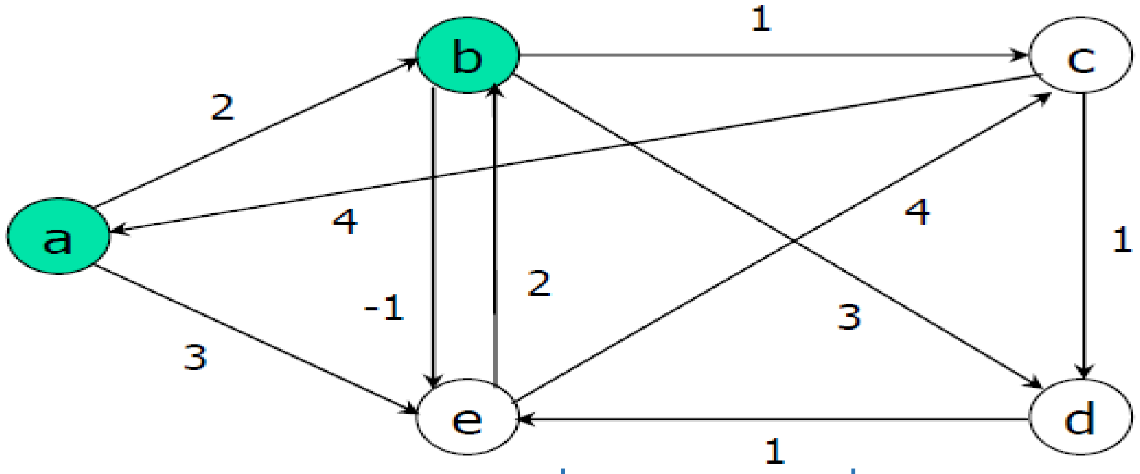


$$
D^{(1)} = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array}
\begin{array}{c}
\begin{array}{ccccc} a & b & c & d & e \end{array} \\
\begin{bmatrix}
0 & 2 & \infty & \infty & 3 \\
\infty & 0 & 1 & 3 & -1 \\
4 & \underline{6} & 0 & 1 & \underline{7} \\
\infty & \infty & \infty & 0 & 1 \\
\infty & 2 & 4 & \infty & 0
\end{bmatrix}
\end{array}
$$

$$
\begin{aligned}
d_{35}^{(1)} &= \min(d_{35}^{(0)}, d_{31}^{(0)}+d_{15}^{(0)}) \\
&= \min(\infty, 4+3) \\
&= 7
\end{aligned}
$$

Floyd-Warshall algorithm (cont‘d)

Example : step(3/6), k=2

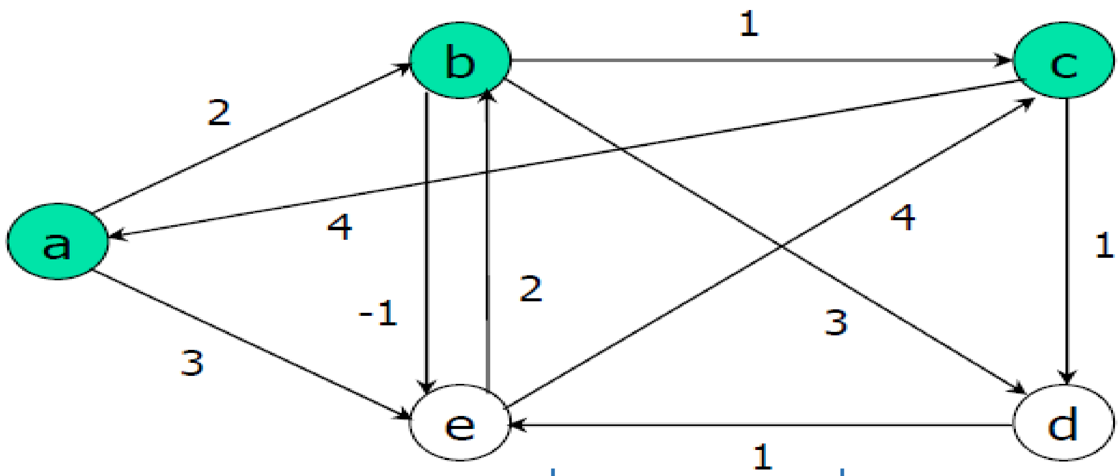


$$
D^{(2)} = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array}
\begin{array}{c}
\begin{array}{ccccc} a & b & c & d & e \end{array} \\
\begin{bmatrix}
0 & 2 & \underline{3} & \underline{5} & \underline{1} \\
\infty & 0 & 1 & 3 & -1 \\
4 & 6 & 0 & 1 & \underline{5} \\
\infty & \infty & \infty & 0 & 1 \\
\infty & 2 & 3 & \underline{5} & 0
\end{bmatrix}
\end{array}
$$

$d_{35}^{(2)} = \min(d_{35}^{(1)}, d_{32}^{(1)}+d_{25}^{(1)})$
$= \min(7, 6\text{-}1)$
$= 5$

Floyd-Warshall algorithm (cont‘d)

Example : step(4/6), k=3

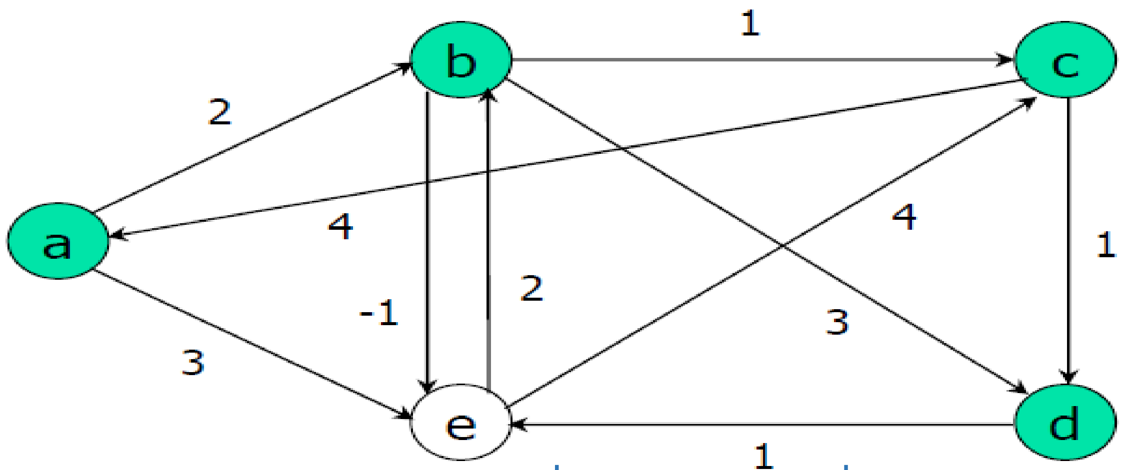


$$D^{(3)} = \begin{array}{cc} & \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \\ e \end{array} & \begin{bmatrix} 0 & 2 & 3 & \underline{4} & 1 \\ \underline{5} & 0 & 1 & \underline{2} & -1 \\ 4 & 6 & 0 & 1 & 5 \\ \infty & \infty & \infty & 0 & 1 \\ \underline{7} & 2 & 3 & \underline{4} & 0 \end{bmatrix} \end{array}$$

$$d_{35}^{(3)} = \min(d_{35}^{(2)}, d_{33}^{(2)}+d_{35}^{(2)}) = \min(5, 0+5) = 5$$

Floyd-Warshall algorithm (cont`d)

Example : step(5/6), k=4

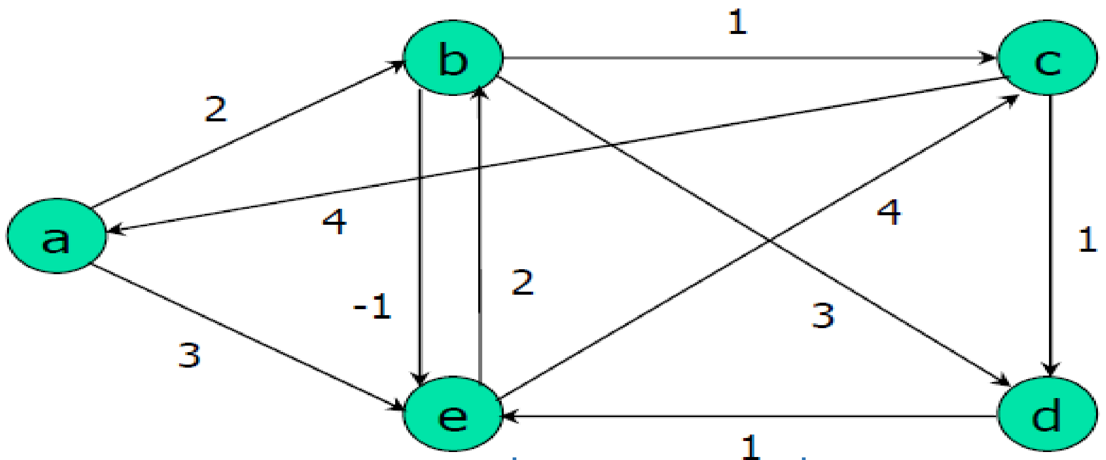


$$D^{(4)} = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{bmatrix} 0 & 2 & 3 & 4 & 1 \\ 5 & 0 & 1 & 2 & -1 \\ 4 & 6 & 0 & 1 & \underline{2} \\ \infty & \infty & \infty & 0 & 1 \\ 7 & 2 & 3 & 4 & 0 \end{bmatrix} \end{array}$$

$d_{35}^{(4)} = \min(d_{35}^{(3)}, d_{34}^{(3)}+d_{45}^{(3)})$
$= \min(5, 1+1)$
$= 2$

Floyd-Warshall algorithm (cont‘d)

Example : step(6/6) , k=5



$$D^{(5)} = \begin{array}{c|ccccc} & a & b & c & d & e \\ \hline a & 0 & 2 & 3 & 4 & 1 \\ b & 5 & 0 & 1 & 2 & -1 \\ c & 4 & \underline{4} & 0 & 1 & 2 \\ d & \underline{8} & \underline{3} & \underline{4} & 0 & 1 \\ e & 7 & 2 & 3 & 4 & 0 \end{array}$$

$$\begin{aligned} d_{35}^{(5)} &= \min(d_{35}^{(4)}, d_{35}^{(4)}+d_{55}^{(4)}) \\ &= \min(2, 2+0) \\ &= 2 \end{aligned}$$

Floyd-Warshall algorithm (cont‘d)

Compute bottom-up

Compute in increasing order of k:

```
FLOYD-WARSHALL(W, n)
D^(0) = W
for k = 1 to n
    let D^(k) = (d_ij^(k)) be a new n × n matrix
    for i = 1 to n
        for j = 1 to n
            d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
return D^(n)
```

$\text{FLOYD-WARSHALL}(W, n)$

$D^{(0)} = W$

for $k = 1$ **to** n

  let $D^{(k)} = (d_{ij}^{(k)})$ 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 $D^{(n)}$

running time: $O(n^3)$

Floyd-Warshall algorithm (cont‘d)

```
Floyd(A, n) {
/* input: adjacency matrix A[1..n][1..n], n=|V|,
   output: distance matrix D */

        int  i,  j,  k;
        for (i = 1;  i <= n;  i++)
            for (j = 1;  j <= n;  j++)
                D[i][j] = A[i][j];
        for (k = 1;  k <= n;  k++)
            for (i = 1;  i <= n;  i++)
                for (j = 1; j <= n;  j++)
                    if (D[i][j] > D[i][k] + D[k][j])
                        D[i][j] = D[i][k] + D[k][j];
  }
```

Implementation

Implementation of Floyd-Warshall algorithm

Compute bottom-up

Compute in increasing order of k:

FLOYD-WARSHALL(W, n)

$D^{(0)} = W$

for $k = 1$ **to** n

  let $D^{(k)} = (d_{ij}^{(k)})$ 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 $D^{(n)}$

```
import sys
N = 5 # matrix size NxN
M = 5 # the number of edges

distance = [[sys.maxsize for j in range(N)] for i in range(N)]

# inital edges
for i in range(0, N):
  distance[i][i] = 0

distance[0][1] = 2
distance[0][4] = 3
distance[1][2] = 1
distance[1][3] = 3
distance[1][4] = -1
distance[2][0] = 4
distance[2][3] = 1
distance[3][4] = 1
distance[4][1] = 2
distance[4][2] = 3

# core algorithm
for k in range(0, N):
  for i in range(0, N):
    for j in range(0, N):
      if distance[i][j] > distance[i][k] + distance[k][j]:
        distance[i][j] = distance[i][k] + distance[k][j]

for i in range(0, N):
  for j in range(0, N):
    if distance[i][j] == sys.maxsize:
      print('&', end='  ')
    else:
      print(distance[i][j], end='  ')
  print()
```

```
0  2  3  4  1
5  0  1  2  -1
4  4  0  1  2
8  3  4  0  1
7  2  3  4  0
```

Example code test

- Code test: https://www.acmicpc.net/problem/11403
- Solving the problem using Floyd-Warshall algorithm
- Example result of submission

THANK YOU