Announcement

Programming assignment # 3 has been posted in both Blackboard and course website

Due at: 11:59pm EST, Sunday, April 10

Announcement

We will have a quiz on Tuesday, April 5. The quiz is about the question of using Floyd's algorithm for all-pairs shortest paths

Chapter 8: Dynamic Programming

Dynamic Programming is a general algorithm design technique

Invented by American mathematician Richard Bellman in the 1950s to solve optimization problems

“Programming” here means “planning”

Main idea:

- solve several smaller (overlapping) subproblems

  Compared to

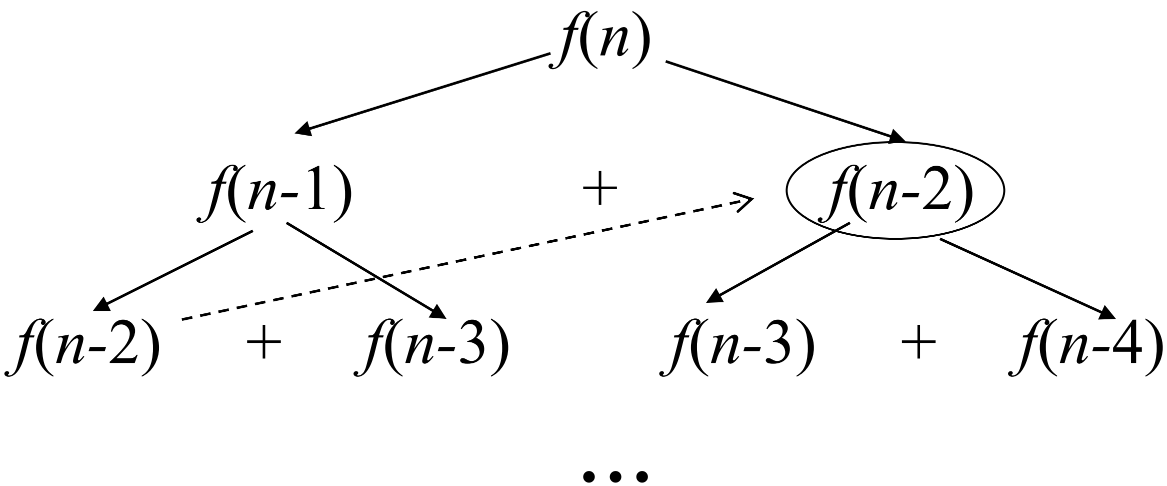
  - Divide-and-conquer: two non-overlapping subproblems
  - Decrease-and-conquer: one non-overlapping subproblem
- record solutions in a table so that each subproblem is only solved once
- final state of the table will be (or contain) the solution

Example: Fibonacci numbers

Recall definition of Fibonacci numbers:

$$f(0) = 0$$
$$f(1) = 1$$
$$f(n) = f(n\text{-}1) + f(n\text{-}2)$$

Computing the n^{th} Fibonacci number recursively (top-down):



Example: Fibonacci numbers

Computing the n^{th} Fibonacci number using bottom-up iteration:

$f(0) = 0$

$f(1) = 1$

$f(2) = 0+1 = 1$

$f(3) = 1+1 = 2$

$f(4) = 1+2 = 3$

$f(5) = 2+3 = 5$

- ***A table stores the history***
- ***Record only the previous two results***

…

$f(n\text{-}2) =$

$f(n\text{-}1) =$

$f(n) = f(n\text{-}1) + f(n\text{-}2)$

Examples of Dynamic Programming Algorithms

Warshall's algorithm for transitive closure

Floyd's algorithms for all-pairs shortest paths

Some instances of difficult discrete optimization problems:

- travelling salesman
- knapsack

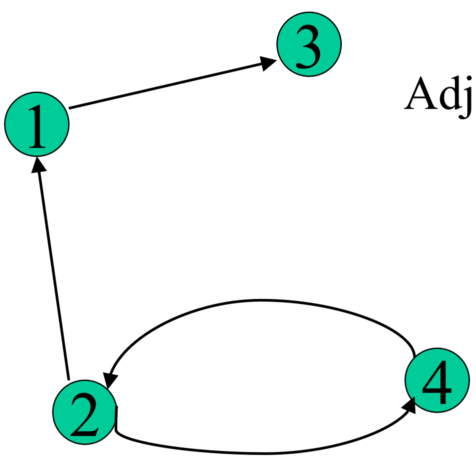
Transitive Closure

In a **directed graph** with ***n*** vertices:

- A ***transitive closure*** is an nxn boolean matrix $T = \{t_{ij}\}$

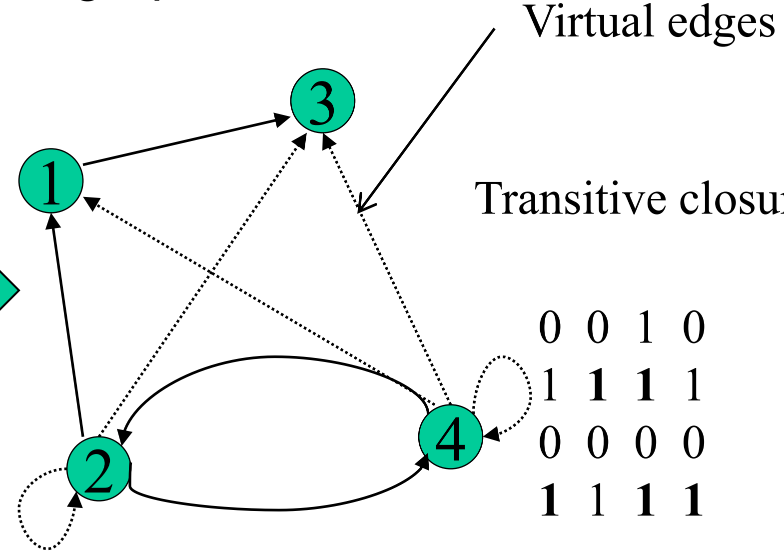
$$\boldsymbol{R}_{ij} = \begin{cases} 1 & \text{There is a directed } \textbf{path} \text{ from vertex i to vertex j} \\ 0 & \text{otherwise} \end{cases}$$

- Problem: Computes the transitive closure of a graph
- Solution: find all paths in a directed graph



Adjacency matrix

0	0	1	0
1	0	0	1
0	0	0	0
0	1	0	0



Transitive closure

0	0	1	0
1	**1**	**1**	1
0	0	0	0
1	1	**1**	**1**

Warshall's Algorithm

Main idea: a path exists between two vertices i, j, iff

- there is an edge from i to j; or
- there is a path from i to j going through vertex 1; or
- there is a path from i to j going through vertex 1 and/or 2; or
- there is a path from i to j going through vertex 1, 2, and/or 3; or
- ...
- there is a path from i to j going through any of the other vertices

Construct the transitive closure in a sequence of steps

Warshall's Algorithm

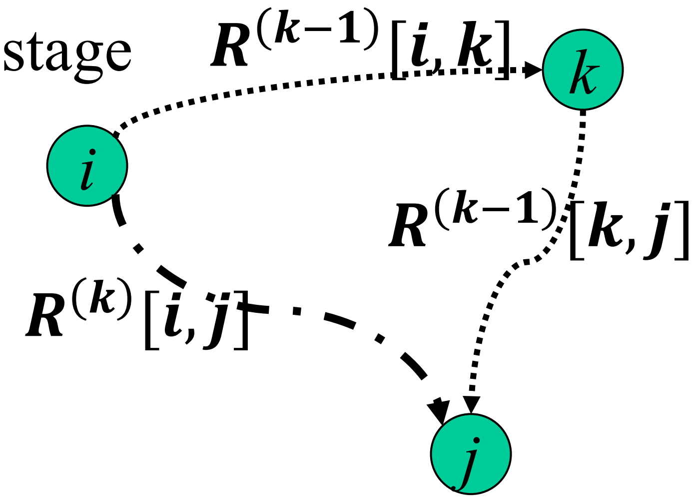
In the k^{th} stage determine if a path exists between two vertices *i, j* using intermediate vertices numbered within 1,…,*k*

- **$R^{(0)}$ does not allow any intermediate vertex and is equal to adjacency matrix**

$$R^{(k)}[i,j] = \begin{cases} R^{(k-1)}[i,j] & \text{(path using just } 1,\ldots,k\text{-}1\text{)} \\ \quad \textbf{or} & \\ (R^{(k-1)}[i,k] \ \textbf{and}\ R^{(k-1)}[k,j]) & \text{(path from } i \text{ to } k \text{ and from } k \text{ to } j \text{ using just } 1,\ldots,k\text{-}1\text{)} \end{cases}$$

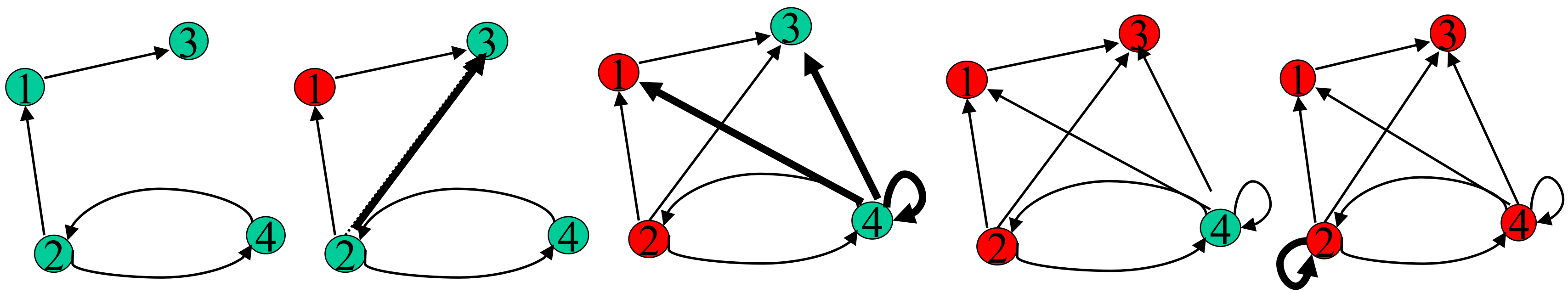
Logic operation

k^{th} stage



K is the new intermediate node

Warshall's Algorithm



R_0

0	0	1	0
1	0	0	1
0	0	0	0
0	1	0	0

R_1

0	0	1	0
1	0	**1**	1
0	0	0	0
0	1	0	0

R_2

0	0	1	0
1	0	1	1
0	0	0	0
1	1	**1**	**1**

R_3

0	0	1	0
1	0	1	1
0	0	0	0
1	1	1	1

R_4

0	0	1	0
1	**1**	1	1
0	0	0	0
1	1	1	1

Pseudocode of Warshall's Algorithm

ALGORITHM $Warshall(A[1..n, 1..n])$

$R^{(0)} \leftarrow A$

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

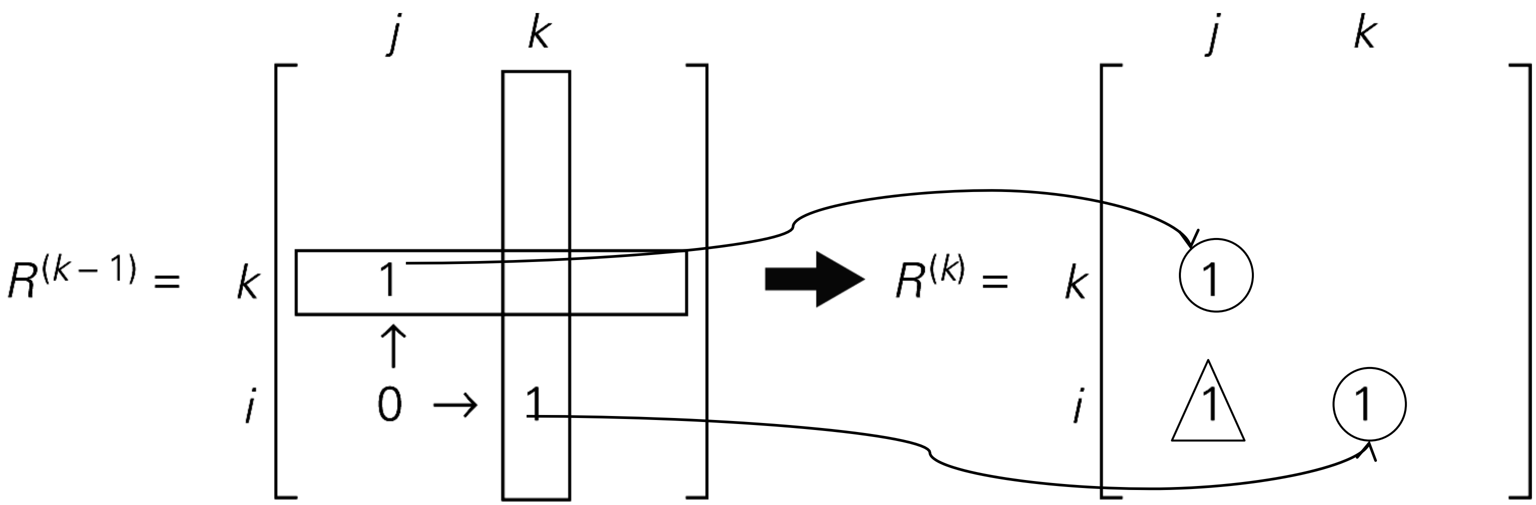
  for $i \leftarrow 1$ **to** n **do**

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

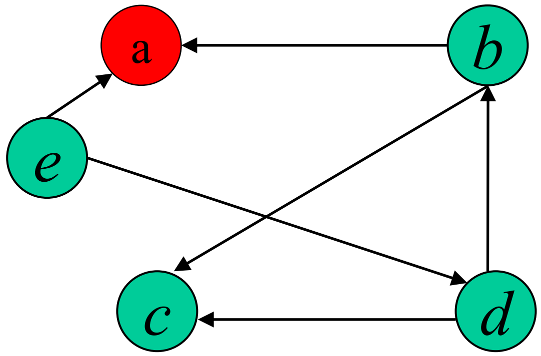
      $R^{(k)}[i, j] \leftarrow R^{(k-1)}[i, j]$ **or** $(R^{(k-1)}[i, k]$ **and** $R^{(k-1)}[k, j])$

return $R^{(n)}$

Warshall's Algorithm



First Example



Start from the adjacency matrix

$R^{(0)} =$

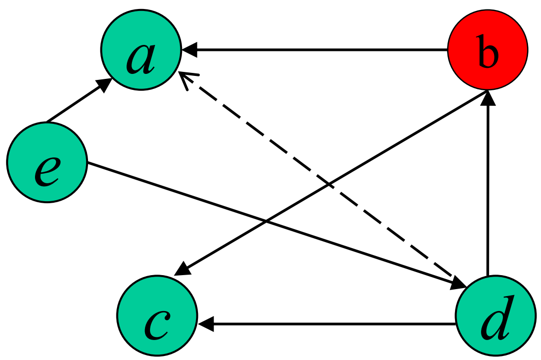
	a	b	c	d	e
a	0	0	0	0	0
b	1	0	1	0	0
c	0	0	0	0	0
d	0	1	1	0	0
e	1	0	0	1	0

No new path through a

$R^{(1)} =$

	a	b	c	d	e
a	0	0	0	0	0
b	1	0	1	0	0
c	0	0	0	0	0
d	0	1	1	0	0
e	1	0	0	1	0

First Example



$R^{(1)} =$

	a	b	c	d	e
a	0	0	0	0	0
b	1	0	1	0	0
c	0	0	0	0	0
d	0	1	1	0	0
e	1	0	0	1	0

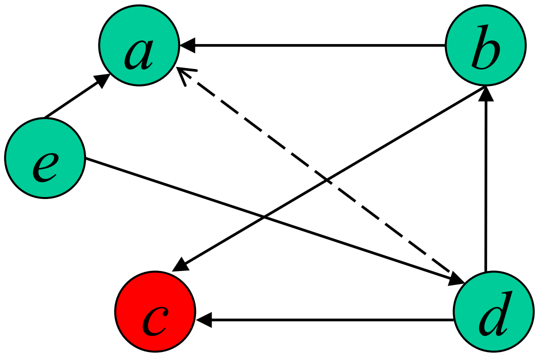
d-b and b-a → d-a

d-b and b-c → d-c exists

$R^{(2)} =$

	a	b	c	d	e
a	0	0	0	0	0
b	1	0	1	0	0
c	0	0	0	0	0
d	1	1	1	0	0
e	1	0	0	1	0

First Example



$R^{(2)} =$

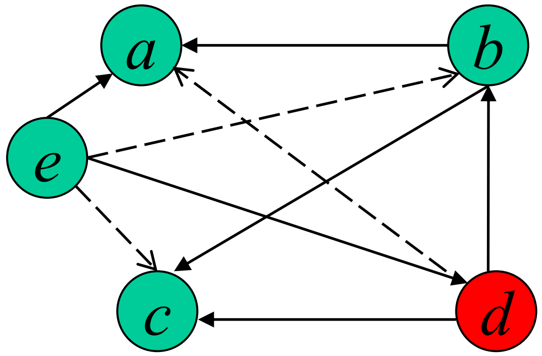
	a	b	c	d	e
a	0	0	0	0	0
b	1	0	1	0	0
c	0	0	0	0	0
d	1	1	1	0	0
e	1	0	0	1	0

No new path through c

$R^{(3)} =$

	a	b	c	d	e
a	0	0	0	0	0
b	1	0	1	0	0
c	0	0	0	0	0
d	1	1	1	0	0
e	1	0	0	1	0

First Example



$R^{(3)} =$

	a	b	c	d	e
a	0	0	0	0	0
b	1	0	1	0	0
c	0	0	0	0	0
d	1	1	1	0	0
e	1	0	0	1	0

e-d and d-a → e-a exists

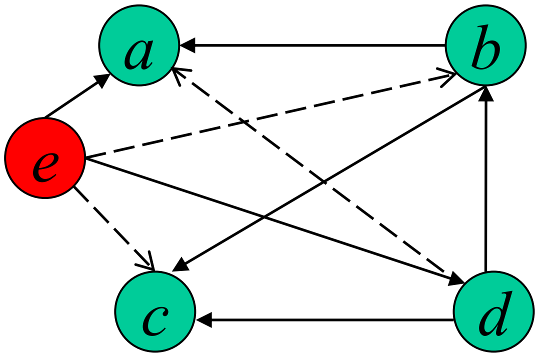
e-d and d-b → e-b

e-d and d-c → e-c

$R^{(4)} =$

	a	b	c	d	e
a	0	0	0	0	0
b	1	0	1	0	0
c	0	0	0	0	0
d	1	1	1	0	0
e	1	1	1	1	0

First Example



$$R^{(4)} =$$

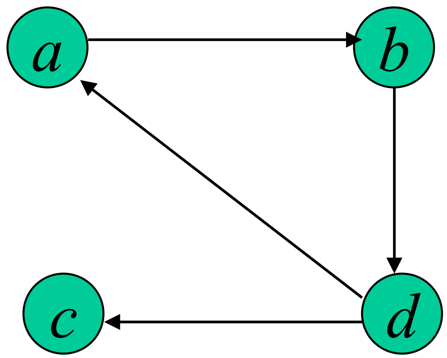
	a	b	c	d	e
a	0	0	0	0	0
b	1	0	1	0	0
c	0	0	0	0	0
d	1	1	1	0	0
e	1	1	1	1	0

No new path through e

$$R=R^{(5)} =$$

	a	b	c	d	e
a	0	0	0	0	0
b	1	0	1	0	0
c	0	0	0	0	0
d	1	1	1	0	0
e	1	1	1	1	0

Second Example



$$R^{(0)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$$R^{(1)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & \mathbf{1} & 1 & 0 \end{bmatrix} \end{array}$$

$$R^{(2)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & \mathbf{1} \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & \mathbf{1} \end{bmatrix} \end{array}$$

$$R^{(3)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

$$R^{(4)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} \mathbf{1} & 1 & \mathbf{1} & 1 \\ \mathbf{1} & \mathbf{1} & \mathbf{1} & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

Time efficiency? ***Best case?*** $\Theta(n^3)$

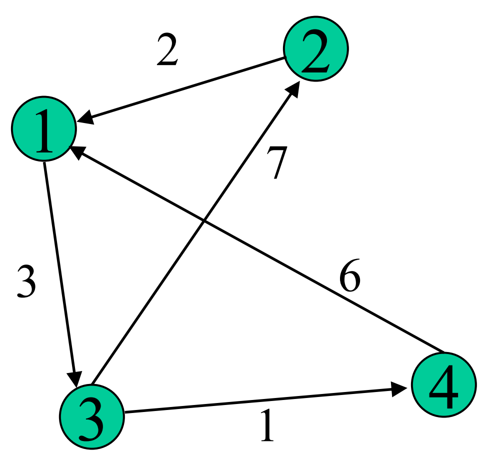
Average case? $\Theta(n^3)$

Worst case? $\Theta(n^3)$

Floyd's Algorithm: All pairs shortest paths

In a weighted graph, find shortest paths between every pair of vertices

Same idea: construct solution through series of matrices $D^{(0)}$, $D^{(1)}$, … using an initial subset of the vertices as intermediaries.



	1	2	3	4
1	0	∞	3	∞
2	2	0	∞	∞
3	∞	7	0	1
4	6	∞	∞	0

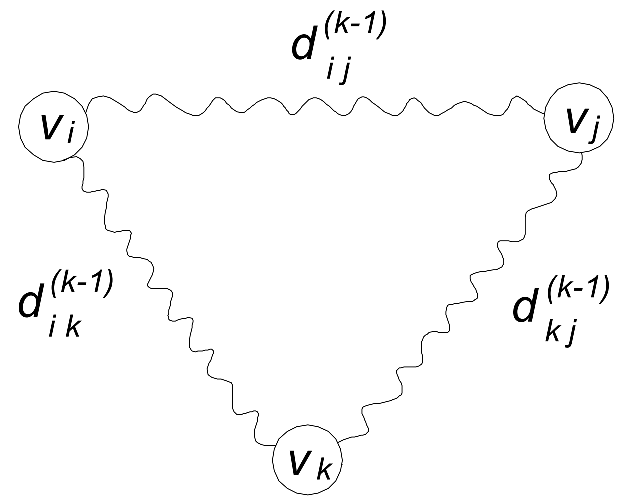
Weight matrix

	1	2	3	4
1	0	10	3	4
2	2	0	5	6
3	7	7	0	1
4	6	16	9	0

Distance matrix

Similar to Warshall's Algorithm

$d_{ij}^{(k)}$ in $D^{(k)}$ **is equal to the length of shortest path among all paths from the *i*th vertex to *j*th vertex with each intermediate vertex, if any, numbered not higher than *k***



$$d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\} \text{ for } k \geq 1, d_{ij}^{(0)} = w_{ij}$$

Pseudocode of Floyd's Algorithm

The next matrix in sequence can be written over its predecessor

ALGORITHM $Floyd(W[1..n, 1..n])$

$D \leftarrow W$

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

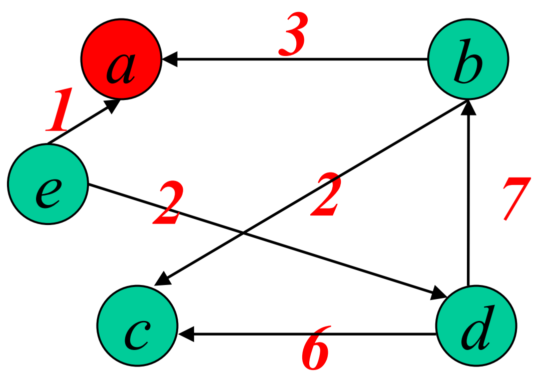
  for $i \leftarrow 1$ **to** n **do**

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

      $D[i, j] \leftarrow \min\{D[i, j], D[i,k] + D[k, j]\}$

return D

First Example



Start from the weight matrix

$D^{(0)} =$

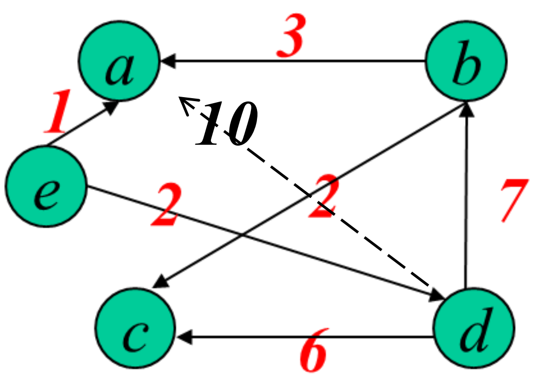
	a	b	c	d	e
a	0	∞	∞	∞	∞
b	3	0	2	∞	∞
c	∞	∞	0	∞	∞
d	∞	7	6	0	∞
e	1	∞	∞	2	0

No new path through a

$D^{(1)} =$

	a	b	c	d	e
a	0	∞	∞	∞	∞
b	3	0	2	∞	∞
c	∞	∞	0	∞	∞
d	∞	7	6	0	∞
e	1	∞	∞	2	0

First Example



$$D^{(1)} =$$

	a	b	c	d	e
a	0	∞	∞	∞	∞
b	3	0	2	∞	∞
c	∞	∞	0	∞	∞
d	∞	7	6	0	∞
e	1	∞	∞	2	0

$D_{d,b} + D_{b,a} = 7 + 3 = 10 < D_{d,a}$

Reset $D_{d,a}$

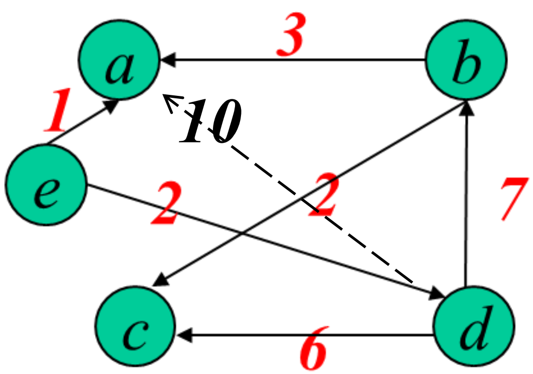
$D_{d,b} + D_{b,c} = 7 + 2 = 9 > D_{d,c}$

Keep it

$$D^{(2)} =$$

	a	b	c	d	e
a	0	∞	∞	∞	∞
b	3	0	2	∞	∞
c	∞	∞	0	∞	∞
d	10	7	6	0	∞
e	1	∞	∞	2	0

First Example



$D^{(2)} =$

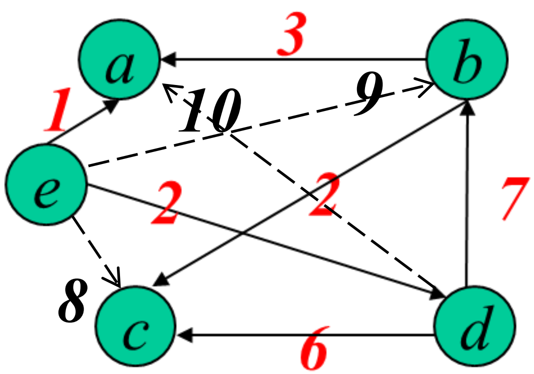
	a	b	c	d	e
a	0	∞	∞	∞	∞
b	3	0	2	∞	∞
c	∞	∞	0	∞	∞
d	10	7	6	0	∞
e	1	∞	∞	2	0

No new path through c

$D^{(3)} =$

	a	b	c	d	e
a	0	∞	∞	∞	∞
b	3	0	2	∞	∞
c	∞	∞	0	∞	∞
d	10	7	6	0	∞
e	1	∞	∞	2	0

First Example



$$D^{(3)} =$$

	a	b	c	d	e
a	0	∞	∞	∞	∞
b	3	0	2	∞	∞
c	∞	∞	0	∞	∞
d	10	7	6	0	∞
e	1	∞	∞	2	0

$$D^{(4)} =$$

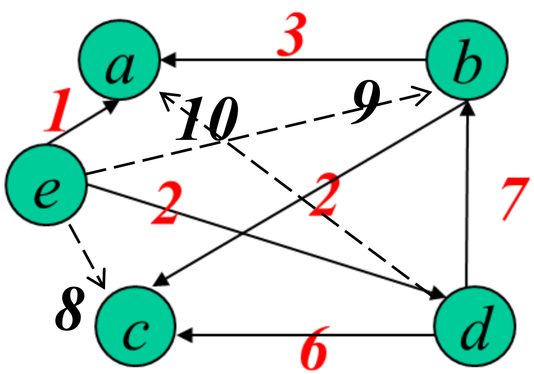
	a	b	c	d	e
a	0	∞	∞	∞	∞
b	3	0	2	∞	∞
c	∞	∞	0	∞	∞
d	10	7	6	0	∞
e	1	9	8	2	0

$$D_{e,d} + D_{d,a} = 2 + 10 = 12 > D_{e,a}$$

$$D_{e,d} + D_{d,b} = 2 + 7 = 9 < D_{e,b}$$

$$D_{e,d} + D_{d,c} = 2 + 6 = 8 < D_{e,b}$$

First Example



$D^{(4)} =$

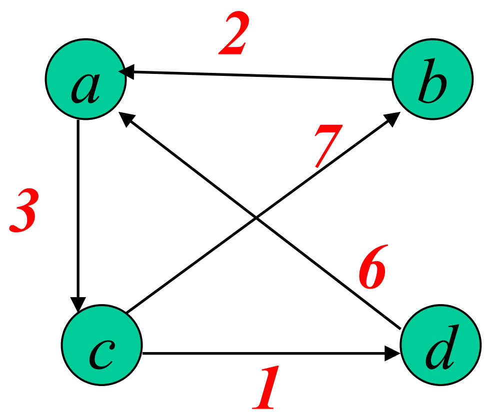
	a	b	c	d	e
a	0	∞	∞	∞	∞
b	3	0	2	∞	∞
c	∞	∞	0	∞	∞
d	10	7	6	0	∞
e	1	9	8	2	0

No new path through e

$D = D^{(5)} =$

	a	b	c	d	e
a	0	∞	∞	∞	∞
b	3	0	2	∞	∞
c	∞	∞	0	∞	∞
d	10	7	6	0	∞
e	1	9	8	2	0

Second Example



$D^{(0)} =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	∞	∞
c	∞	7	0	1
d	6	∞	∞	0

$D^{(1)} =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	**5**	∞
c	∞	7	0	1
d	6	∞	**9**	0

$D^{(2)} =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	5	∞
c	**9**	7	0	1
d	6	∞	9	0

$D^{(3)} =$

	a	b	c	d
a	0	**10**	3	**4**
b	2	0	5	**6**
c	9	7	0	1
d	6	**16**	9	0

$D^{(4)} =$

	a	b	c	d
a	0	10	3	4
b	2	0	5	6
c	**7**	7	0	1
d	6	16	9	0

Time efficiency? $\Theta(n^3)$ **for all cases**

if using adjacency matrix in implementation