# Dynamic Programming

Natasha D.

CPE 212 Algorithm Design (1/2018)

# Topics

- Basic examples
- Knapsack problem
- Warshall's and Floyd's algorithms

# Dynamic Programming

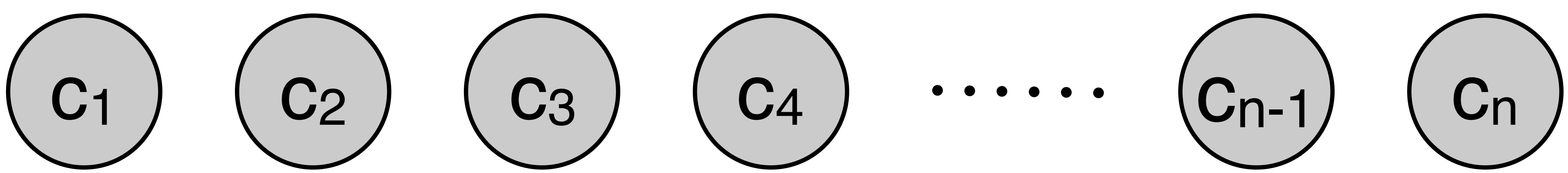
- Dynamic programming was invented by a prominent U.S. mathematician, Richard Bellman, in the 1950s as a general method for ***optimizing multistage decision processes***.
- Thus, the word "*programming*" in the name of this technique stands for "*planning*" and does not refer to computer programming.

# Dynamic Programming

- Dynamic programming is a technique for solving problems with overlapping subproblems.
- Typically, these subproblems arise from a recurrence relating a given problem's solution to solutions of its smaller subproblems.
- ***Rather than solving overlapping subproblems again and again***, dynamic programming suggests solving each of the smaller subproblems only once and recording the results in a table from which a solution to the original problem can then be obtained.

# Coin-Row Problem

- Row of 6 coins whose values are 5, 1, 2, 10, 6, 2
  - Pick up the maximum amount of money
  - No two adjacent coins can be picked up.




- In general, values are $c_1$, $c_2$, $c_3$, …, $c_n$ not necessarily distinct.

- F(n) = Max amount that can be picked up from the row of n coins.
- Partition all the allowed coin selections into two groups
  - Those including the last coin G1
  - Those without the last coin G2
- In G1, the max amount is $c_n$ + F(n-2)
- In G2, the max amount is F(n-1)
- Therefore

$$F(n) = \max\{c_n + F(n-2), F(n-1)\}, \quad n > 1$$

$$F(0) = 0, \quad F(1) = c_1$$

- We can compute F(n) by filling the one-row table left to right.
- Consider the coin row 5, 1, 2, 10, 6, 2.

$$F(n) = \max\{c_n + F(n-2), F(n-1)\}, \quad n > 1$$

$$F(0) = 0, \quad F(1) = c_1$$

$F[0] = 0$, $F[1] = c_1 = 5$

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| $C$ | | 5 | 1 | 2 | 10 | 6 | 2 |
| $F$ | 0 | 5 | | | | | |

$F[2] = \max\{1 + 0, 5\} = 5$

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| $C$ | | 5 | 1 | 2 | 10 | 6 | 2 |
| $F$ | 0 | 5 | 5 | | | | |

$F[3] = \max\{2 + 5, 5\} = 7$

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| $C$ | | 5 | 1 | 2 | 10 | 6 | 2 |
| $F$ | 0 | 5 | 5 | 7 | | | |

$$F(n) = \max\{c_n + F(n-2), F(n-1)\}, \quad n > 1$$

$$F(0) = 0, \quad F(1) = c_1$$

$F[4] = \max\{10 + 5, 7\} = 15$

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| *C* | | 5 | 1 | 2 | 10 | 6 | 2 |
| *F* | 0 | 5 | 5 | 7 | 15 | | |

$F[5] = \max\{6 + 7, 15\} = 15$

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| *C* | | 5 | 1 | 2 | 10 | 6 | 2 |
| *F* | 0 | 5 | 5 | 7 | 15 | 15 | |

$F[6] = \max\{2 + 15, 15\} = 17$

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| *C* | | 5 | 1 | 2 | 10 | 6 | 2 |
| *F* | 0 | 5 | 5 | 7 | 15 | 15 | **17** |

**ALGORITHM** *CoinRow*$(C[1..n])$

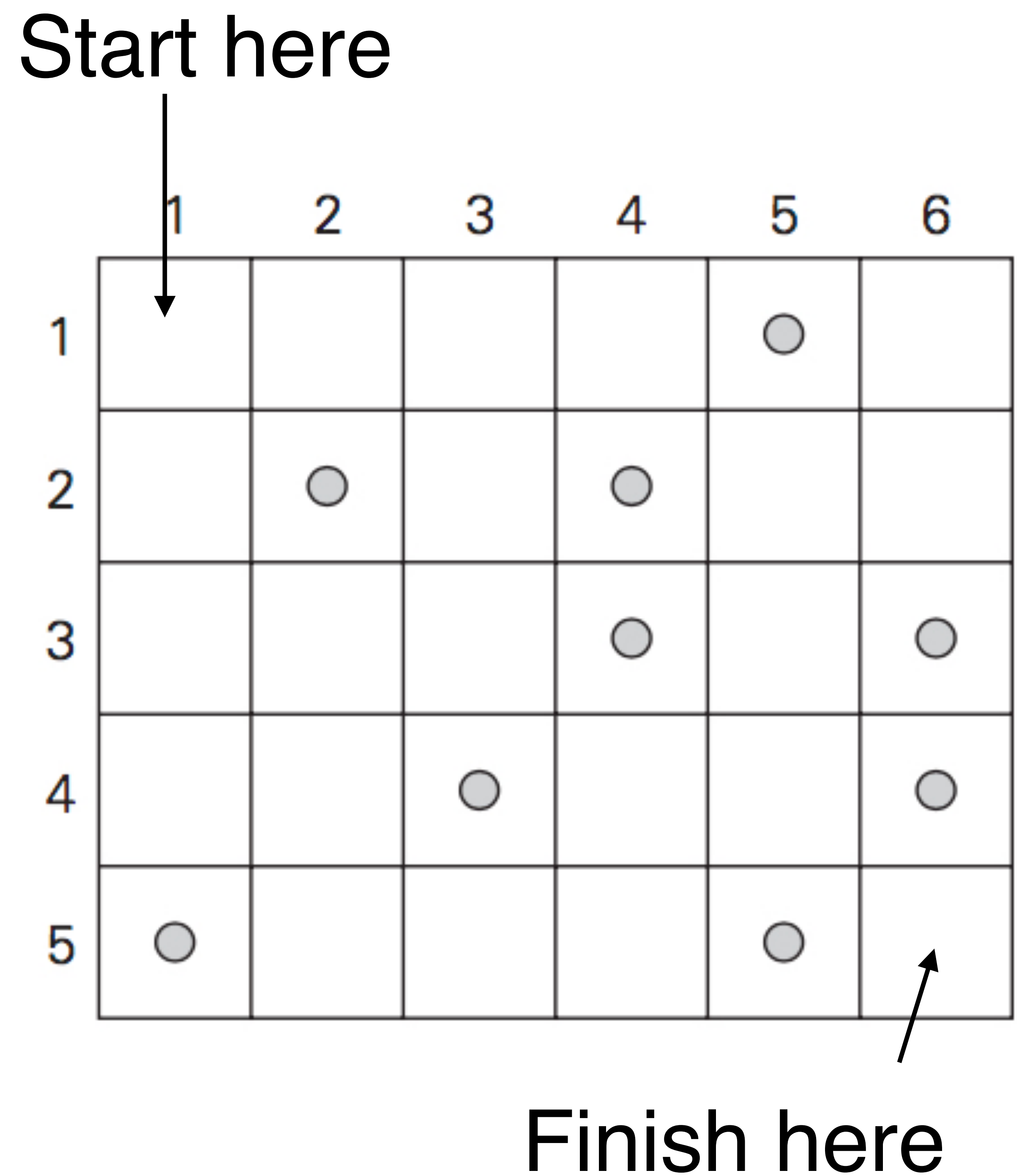
//Applies formula (8.3) bottom up to find the maximum amount of money
//that can be picked up from a coin row without picking two adjacent coins
//Input: Array $C[1..n]$ of positive integers indicating the coin values
//Output: The maximum amount of money that can be picked up
$F[0] \leftarrow 0$; $F[1] \leftarrow C[1]$
**for** $i \leftarrow 2$ **to** $n$ **do**
    $F[i] \leftarrow \max(C[i] + F[i-2], F[i-1])$
**return** $F[n]$

Also keep track here of which coins give higher value

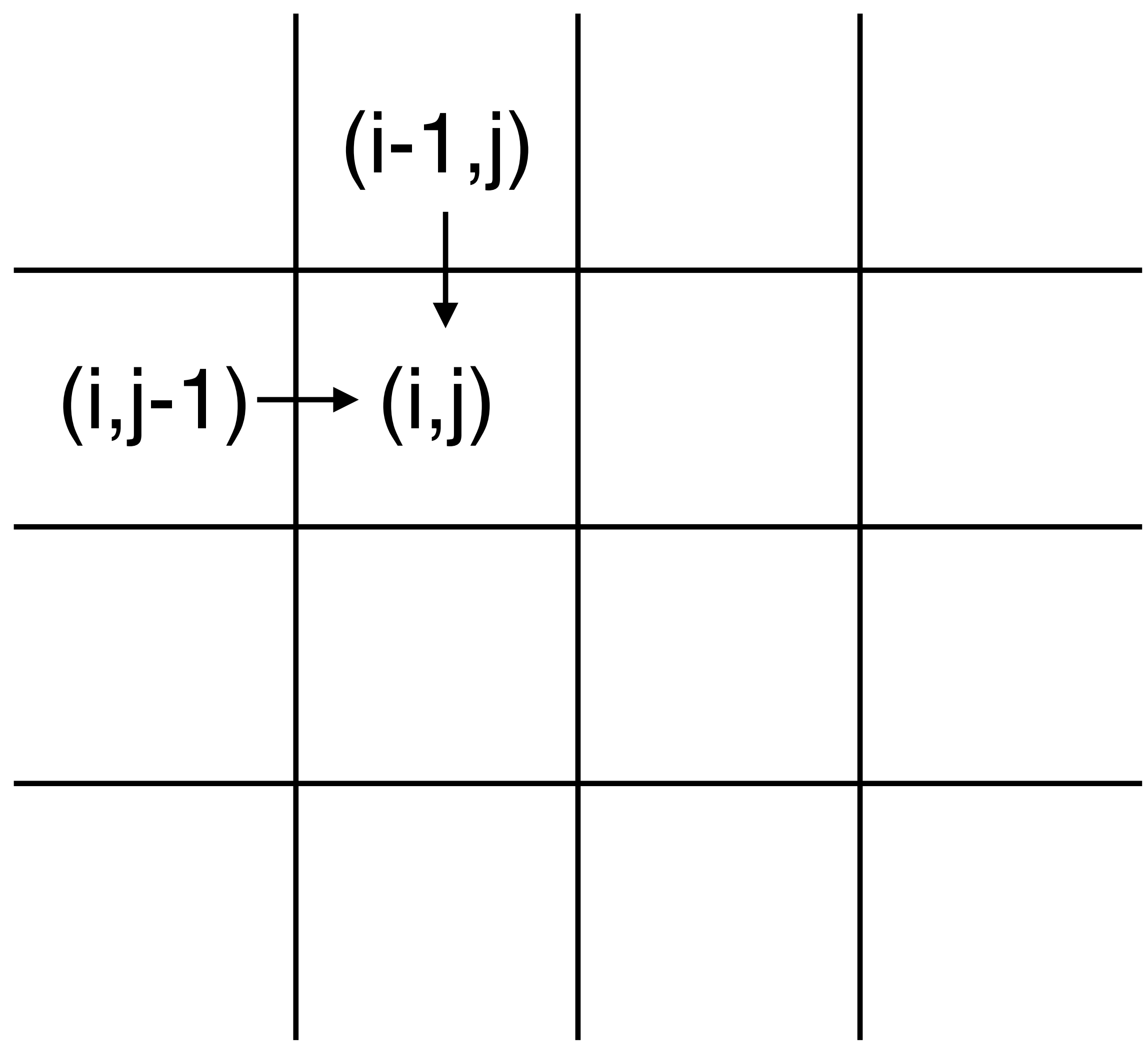
Time complexity Θ(n)
Space complexity Θ(n)

# Coin Collecting Problem

- Coins in cells of an n x m board.
- From an upper left cell, Robot is to collect as many coins as possible to bottom right.
  - Movement is either one cell down or right.
  - What is the path to follow?



- Let F(i,j) be the largest amount of coins that robot can collect and bring to cell (i,j)
  - Can be reached from adjacent cell on the left or above.
  - Largest # coins brought to these cells : F(i,j-1) and F(i-1,j)



- Therefore, F(i,j) satisfies the following formula:

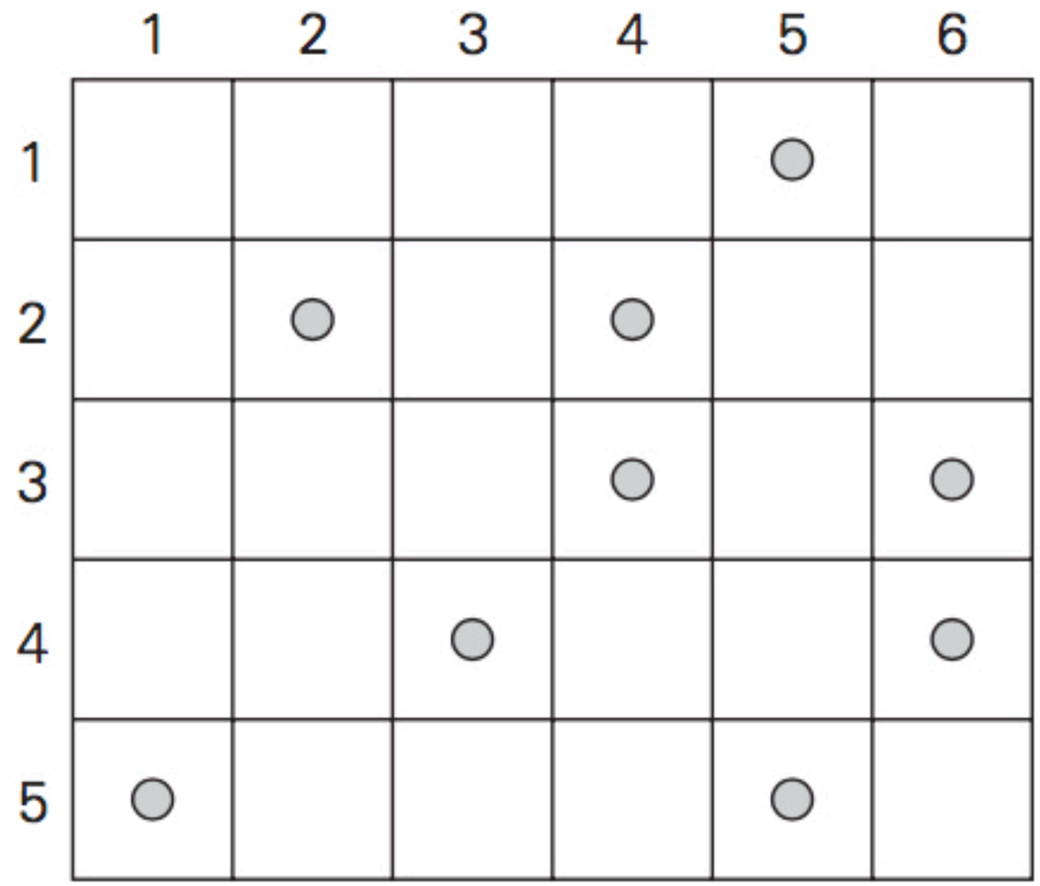
$$F(i,j) = \max\{F(i-1,j), F(i,j-1)\} + c_{ij}, \quad 1 \le i \le n,\ 1 \le j \le m$$

$$F(0,j) = 0, \quad 1 \le j \le m$$

$$F(i,0) = 0, \quad 1 \le i \le n$$

$$c_{ij} = \begin{cases} 1 & \text{coin in cell } (i,j) \\ 0 & \text{no coin in cell } (i,j) \end{cases}$$

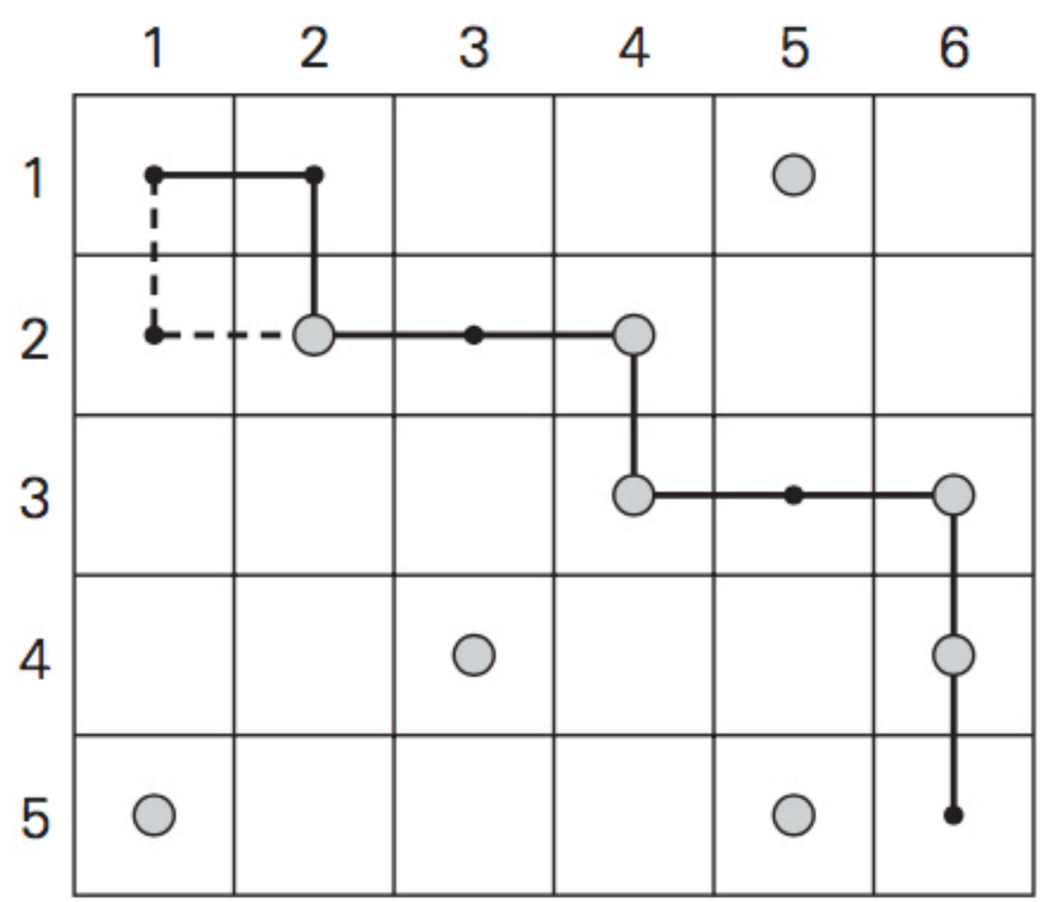
- Fill in n by m table of F(i,j) values either row-by-row or column-by-column.




(a)

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 2 | 0 | 1 | 1 | 2 | 2 | 2 |
| 3 | 0 | 1 | 1 | 3 | 3 | 4 |
| 4 | 0 | 1 | 2 | 3 | 3 | 5 |
| 5 | 1 | 1 | 2 | 3 | 4 | **5** |

(b)




(c)

# General Concepts of Dynamic Programming

- Programming = Planning
- Technique to solve problems with overlapping subproblems.
  - Subproblems typically arise from recurrence relation of between the problem solution and subproblem solutions.
  - Subproblem could be solved only once and results recorded in a table similar to space-time trade-off design.

# Knapsack Problem

| item | weight | value |
|---|---|---|
| 1 | 2 | \$12 |
| 2 | 1 | \$10 |
| 3 | 3 | \$20 |
| 4 | 2 | \$15 |

capacity $W = 5$.

- Select items whose sum of weight does not exceed W and sum of values is maximized.
- n items with known weights $w_1, w_2, \ldots, w_n$, values $v_1, v_2, \ldots, v_n$, and knapsack capicity W

- Let $F(i,j)$ be the ***optimal value*** obtained from a subset of first i items that fit into the knapsack of capacity j.
- All subsets of first i items can be divided into two cases
  - those not including $i^{th}$ item
  - those including $i^{th}$ item
- For those not including $i^{th}$ item, $F(i,j) = F(i-1,j)$
- For those including $i^{th}$ item ($j - w_i \geq 0$)
  - Optimal subset made up of $i^{th}$ item and optimal subset of first $i - 1$ items in knapsack capacity $j - w_i$
  - Optimal value is therefore $v_i + F(i-1, j-w_i)$

**ALGORITHM** *MFKnapsack*$(i, j)$

```
//Implements the memory function method for the knapsack problem
//Input: A nonnegative integer i indicating the number of the first
//       items being considered and a nonnegative integer j indicating
//       the knapsack capacity
//Output: The value of an optimal feasible subset of the first i items
//Note: Uses as global variables input arrays Weights[1..n], Values[1..n],
//and table F[0..n, 0..W] whose entries are initialized with −1's except for
//row 0 and column 0 initialized with 0's
if F[i, j] < 0
    if j < Weights[i]
        value ← MFKnapsack(i − 1, j)
    else
        value ← max(MFKnapsack(i − 1, j),
                    Values[i] + MFKnapsack(i − 1, j − Weights[i]))
    F[i, j] ← value
return F[i, j]
```

- Therefore, among all feasible subsets of the first i items,

$$F(i, j) = \begin{cases} \max\{F(i-1, j), v_i + F(i-1, j-w_i)\} & \text{if } j - w_i \geq 0 \\ F(i-1, j) & \text{if } j - w_i < 0 \end{cases}$$

$$F(0, j) = 0, \quad j \geq 0$$

$$F(i, 0) = 0, \quad i \geq 0$$

- Our goal is to find F(n, W)

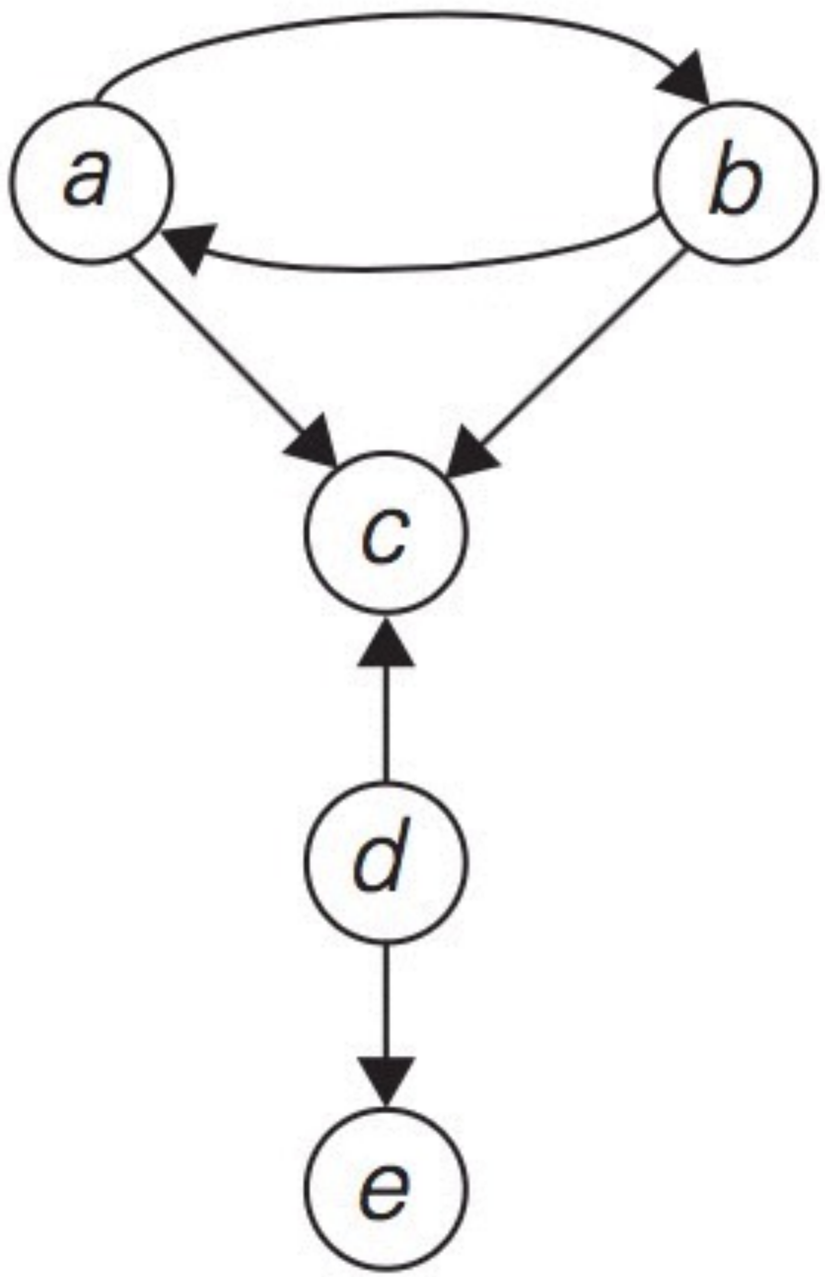
| | | $0$ | $j-w_i$ | $j$ | $W$ |
|---|---|---|---|---|---|
| | $0$ | $0$ | $0$ | $0$ | $0$ |
| | $i-1$ | $0$ | $F(i-1, j-w_i)$ | $F(i-1, j)$ | |
| $w_i, v_i$ | $i$ | $0$ | | $F(i, j)$ | |
| | $n$ | $0$ | | | goal |

| **item** | **weight** | **value** |
| --- | --- | --- |
| 1 | 2 | $12 |
| 2 | 1 | $10 |
| 3 | 3 | $20 |
| 4 | 2 | $15 |

capacity $W = 5$.

|  | $i$ | | | | **capacity** $j$ | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 0 | 1 | 2 | 3 | 4 | 5 |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $w_1 = 2,\ v_1 = 12$ | 1 | 0 | 0 | 12 | 12 | 12 | 12 |
| $w_2 = 1,\ v_2 = 10$ | 2 | 0 | 10 | 12 | 22 | 22 | 22 |
| $w_3 = 3,\ v_3 = 20$ | 3 | 0 | 10 | 12 | 22 | 30 | 32 |
| $w_4 = 2,\ v_4 = 15$ | 4 | 0 | 10 | 15 | 25 | 30 | **37** |

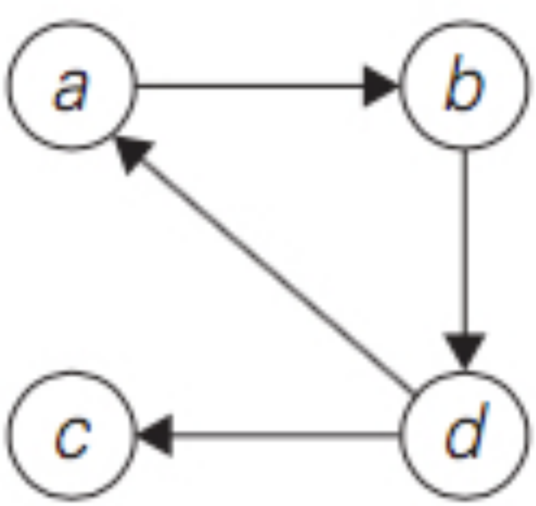
# Digraph and Adjacency Matrix




What is the adjacency matrix ?

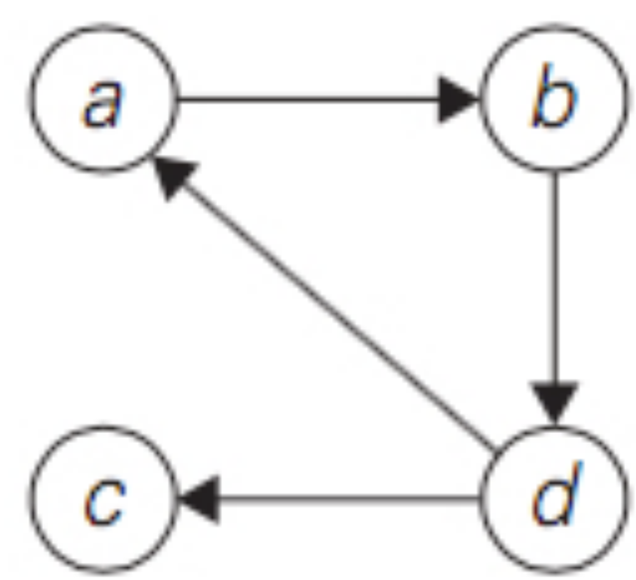
# Transitive Closure of a Directed

- A matrix that tells (in a constant time) if $j^{th}$ vertex is reachable from $i^{th}$ vertex (there exists a directed path from i to j)
    - Represent by a boolean matrix $T = \{t_{ij}\}$




$$A = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array} \qquad T = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

- Application examples
  - Dependency of spreadsheet cells
  - Data flow in software design
- Depth-first-search or Breadth-first-search can be used to generate a transitive closure of a digraph.
  - Perform a traversal at $i^{th}$ vertex and fill in columns in the $i^{th}$ row.
  - Ex: Try DFS starting at vertex a.
- What wrong with DFS or BFS?



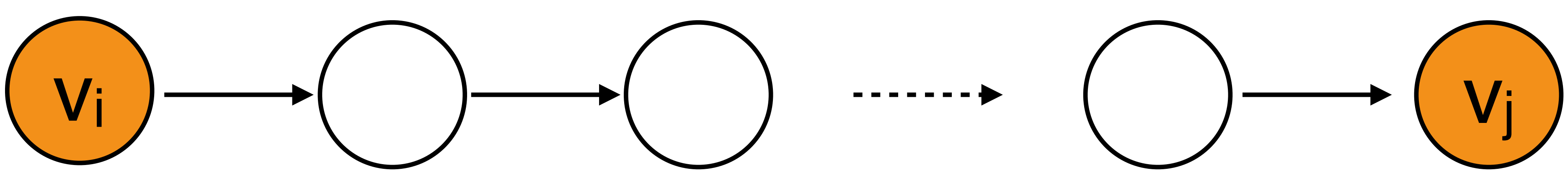
# Warshall's Algorithm

- Label vertices from 1 to n
- Define matrix $R^{(k)}$ whose elements are

$$r_{ij}^{(k)} = \begin{cases} 1 & \text{there exists a directed path from } i \text{ to } j \text{ with} \\ & \text{intermediate vertices (if any) not higher than } k \\ 0 & \text{otherwise} \end{cases}$$

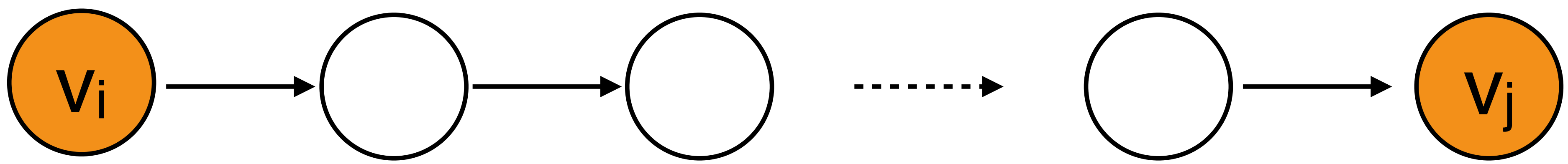
- Transitve closure is constructed through a series of n x n boolean matrices $R^{(0)}$, $R^{(1)}$, …, $R^{(k-1)}$, $R^{(k)}$, …, $R^{(n)}$.

- Examples
  - $R^{(0)}$ is just the adjacency matrix.
  - $R^{(1)}$: Paths only contains the first vertex as intermediate
  - $R^{(2)}$: Paths only contains the first two vertices as intermediate
  - $R^{(n)}$: Paths contains any vertices as intermediate
- $R^{(k)}$ with entry $r_{ij}^{(k)} = 1$ means that there exists a path between $i^{th}$ vertex $v_i$ and $j^{th}$ vertex $v_j$ such that




intermediate vertices ≤ k

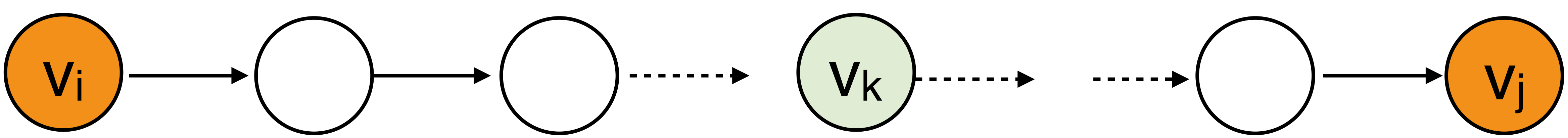
- Two situations regarding the path are possible.
  - list of intermediate vertices does not have k^th vertex.




intermediate vertices ≤ k-1

$$r_{ij}^{(k-1)} = 1$$

  - list of intermediate vertices have k^th vertex.




intermediate vertices ≤ k-1

intermediate vertices ≤ k-1

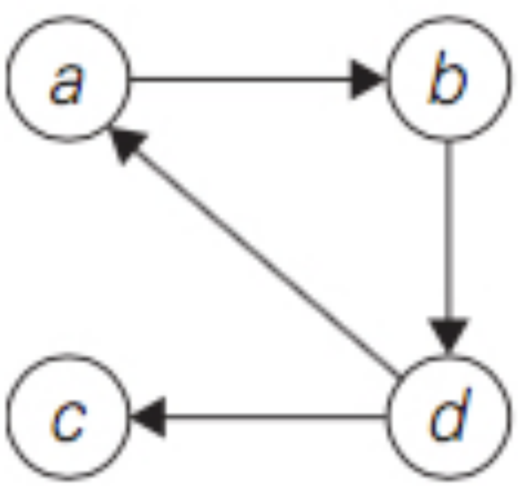
$$r_{ik}^{(k-1)} = r_{kj}^{(k-1)} = 1$$

- Therefore, if $r_{ij}^{(k)} = 1$, then either

$$r_{ij}^{(k)} = r_{ij}^{(k-1)} \quad \text{or} \quad \left(r_{ik}^{(k-1)} \text{ and } r_{kj}^{(k-1)}\right)$$

$$R^{(k-1)} = \begin{array}{cc} & \begin{array}{ccc} j & & k \end{array} \\ \begin{array}{c} \\ k \\ \\ i \\ \\ \end{array} & \begin{bmatrix} & & & \\ 1 & & \boxed{\phantom{1}} & \\ \uparrow & & & \\ 0 & \rightarrow & 1 & \\ & & & \end{bmatrix} \end{array} \Longrightarrow R^{(k)} = \begin{array}{cc} & \begin{array}{cc} j & k \end{array} \\ \begin{array}{c} \\ k \\ \\ i \\ \\ \end{array} & \begin{bmatrix} & & \\ 1 & & \\ & & \\ 1 & 1 & \\ & & \end{bmatrix} \end{array}$$

- Rules for applying Warshall's algorithm by hand:
  - If $r_{ij}$ is 1 in $R^{(k-1)}$, it remains 1 in $R^{(k)}$
  - If $r_{ij}$ is 0 in $R^{(k-1)}$, it has to be changed to 1 in $R^{(k)}$ if and only if $r_{ik}$ and $r_{kj}$ in $R^{(k-1)}$ are both 1.




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

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

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

Work out $R^{(3)}$ and $R^{(4)}$

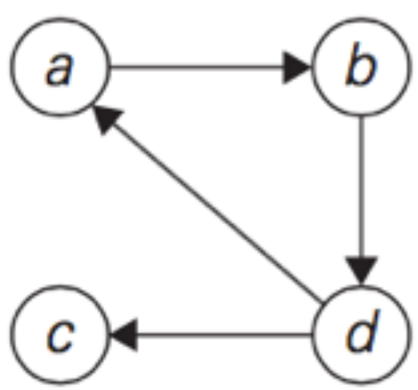
**ALGORITHM** *Warshall*$(A[1..n, 1..n])$

//Implements Warshall's algorithm for computing the transitive closure
//Input: The adjacency matrix $A$ of a digraph with $n$ vertices
//Output: The transitive closure of the digraph

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

**1.** Apply Warshall's algorithm to find the transitive closure of the digraph defined by the following adjacency matrix:

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

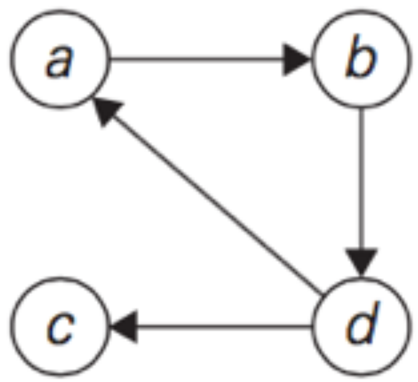



$$R^{(0)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

1's reflect the existence of paths with no intermediate vertices ($R^{(0)}$ is just the adjacency matrix); boxed row and column are used for getting $R^{(1)}$.

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

1's reflect the existence of paths with intermediate vertices numbered not higher than 1, i.e., just vertex *a* (note a new path from *d* to *b*); boxed row and column are used for getting $R^{(2)}$.




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

1's reflect the existence of paths with no intermediate vertices ($R^{(0)}$ is just the adjacency matrix); boxed row and column are used for getting $R^{(1)}$.

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

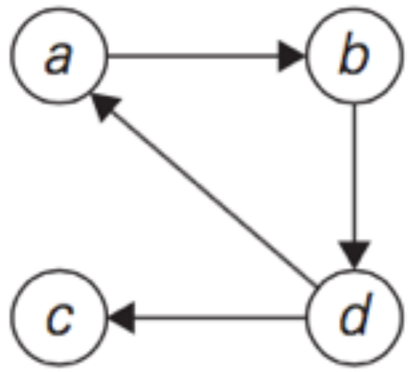
1's reflect the existence of paths with intermediate vertices numbered not higher than 1, i.e., just vertex *a* (note a new path from *d* to *b*); boxed row and column are used for getting $R^{(2)}$.

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

1's reflect the existence of paths with intermediate vertices numbered not higher than 2, i.e., *a* and *b* (note two new paths); boxed row and column are used for getting $R^{(3)}$.

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

1's reflect the existence of paths with intermediate vertices numbered not higher than 3, i.e., *a*, *b*, and *c* (no new paths); boxed row and column are used for getting $R^{(4)}$.




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

1's reflect the existence of paths with no intermediate vertices ($R^{(0)}$ is just the adjacency matrix); boxed row and column are used for getting $R^{(1)}$.

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

1's reflect the existence of paths with intermediate vertices numbered not higher than 1, i.e., just vertex *a* (note a new path from *d* to *b*); boxed row and column are used for getting $R^{(2)}$.

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

1's reflect the existence of paths with intermediate vertices numbered not higher than 2, i.e., *a* and *b* (note two new paths); boxed row and column are used for getting $R^{(3)}$.

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

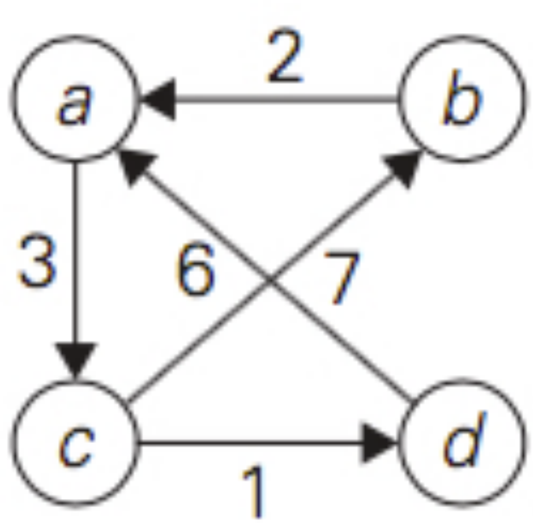
1's reflect the existence of paths with intermediate vertices numbered not higher than 3, i.e., *a, b,* and *c* (no new paths); boxed row and column are used for getting $R^{(4)}$.

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

1's reflect the existence of paths with intermediate vertices numbered not higher than 4, i.e., *a, b, c,* and *d* (note five new paths).

# All-Pairs Shortest-Paths Problem

- Consider a positive weighted connected graph W.
- Find the shortest paths from each vertex to all the others.
- Distance matrix D contains the shortest path length from $i^{th}$ vertex to $j^{th}$ vertex.




$$W = \begin{array}{c|cccc} & a & b & c & d \\ a & 0 & \infty & 3 & \infty \\ b & 2 & 0 & \infty & \infty \\ c & \infty & 7 & 0 & 1 \\ d & 6 & \infty & \infty & 0 \end{array}$$

$$D = \begin{array}{c|cccc} & a & b & c & d \\ a & 0 & 10 & 3 & 4 \\ b & 2 & 0 & 5 & 6 \\ c & 7 & 7 & 0 & 1 \\ d & 6 & 16 & 9 & 0 \end{array}$$

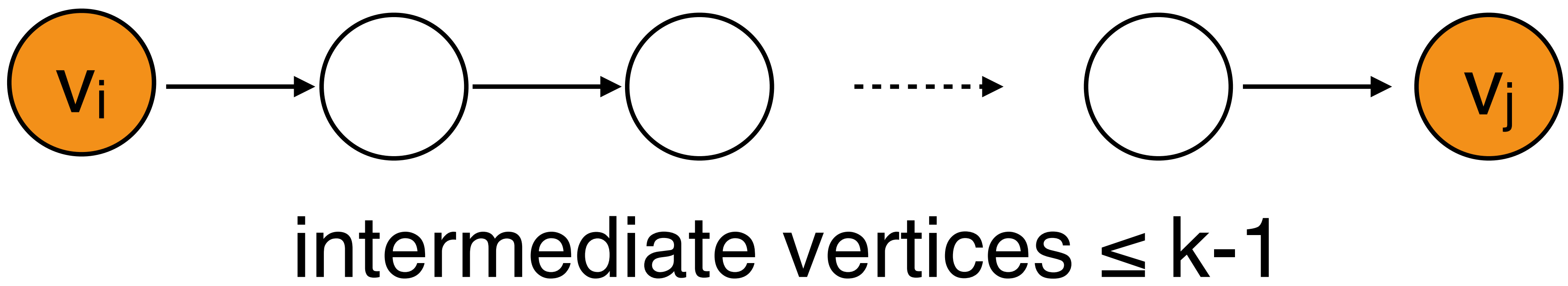
# Floyd's Algorithm

- Compute the distance matrix of a weighted graph with n vertices through a series of n x n matrices:

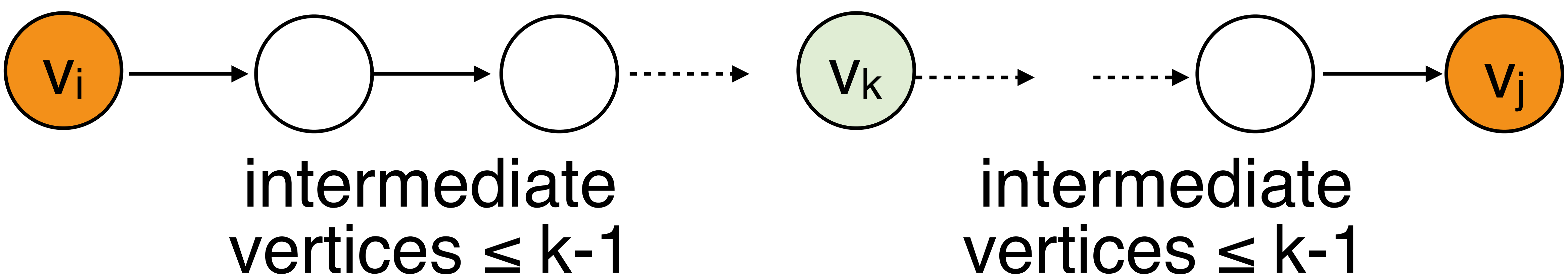
$$D^{(0)}, D^{(1)}, \ldots., D^{(k-1)}, D^{(k)}, \ldots, D^{(n)}$$

- $d_{ij}$ in $D^{(k)}$ is the length of shortest path from i to j with intermediate vertices (if any) not higher than k.
  - $D^{(0)}$ is simply the weight matrix.
  - $D^{(k)}$: Length of shortest paths with only first k vertices as intermediates

- Two situations regarding the path for $d_{ij}^{(k)}$ are possible.
  - I: List of intermediate vertices does not have $k^{th}$ vertex.




  - II: List of intermediate vertices have $k^{th}$ vertex.



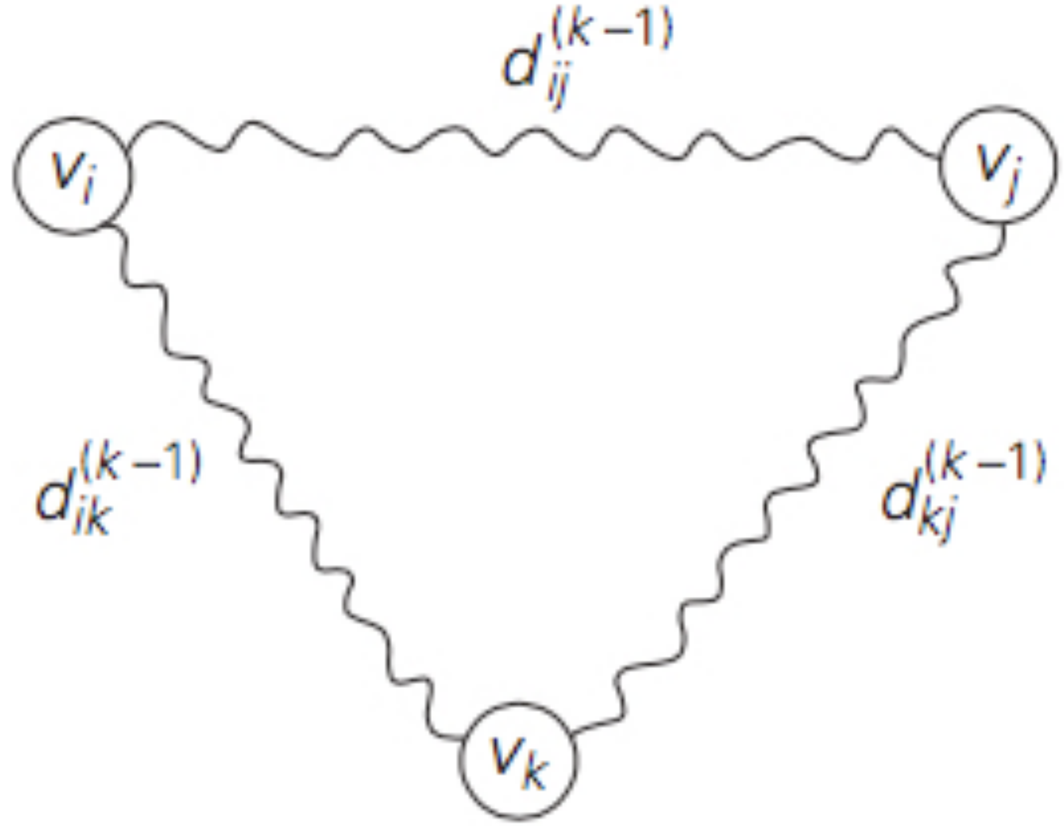
# Floyd's Algorithm

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

//Implements Floyd's algorithm for the all-pairs shortest-paths problem
//Input: The weight matrix $W$ of a graph with no negative-length cycle
//Output: The distance matrix of the shortest paths' lengths

$D \leftarrow W$ //is not necessary if $W$ can be overwritten
**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** $D$

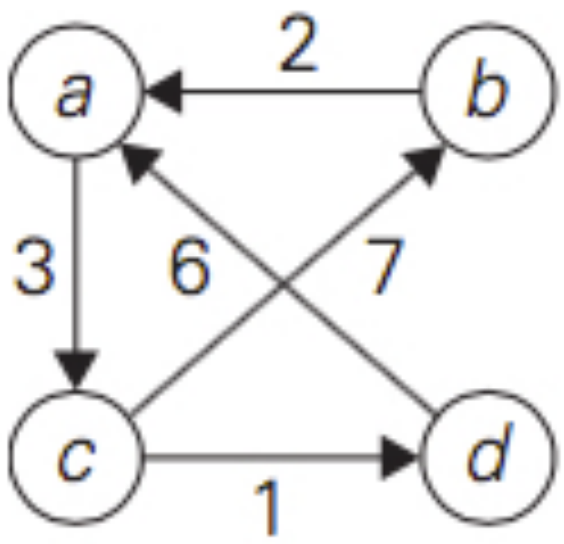
- For subset II, $d_{ij}^{(k)}$ will be different from $d_{ij}^{(k-1)}$




- Both subsets lead to the following recurrence:

$$d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\}, \quad k \geq 1$$

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



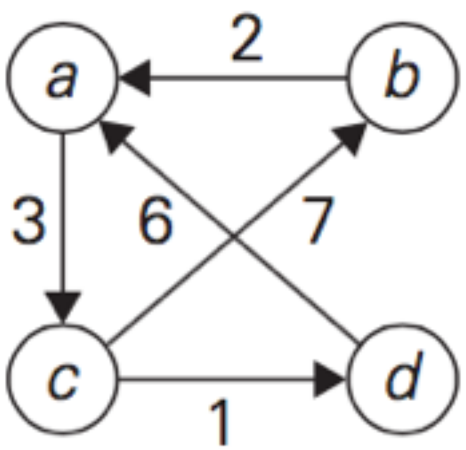

$$D^{(0)} = \begin{array}{cc} & \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \end{array} & \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

$$D^{(1)} = \begin{array}{cc} & \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \end{array} & \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \mathbf{5} & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \mathbf{9} & 0 \end{bmatrix} \end{array}$$

$$D^{(2)} = \begin{array}{cc} & \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \end{array} & \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ \mathbf{9} & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix} \end{array}$$

Work out $D^{(3)}$ and $D^{(4)}$

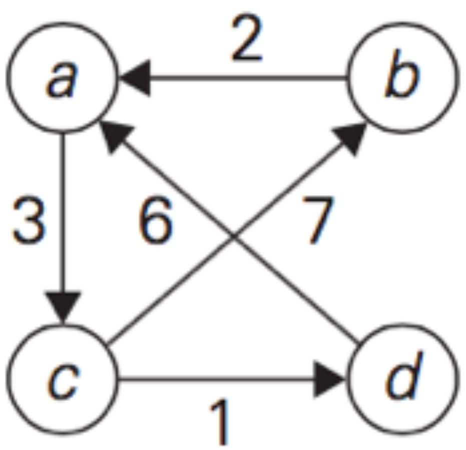
# Floyd's Algorithm




$$D^{(0)} = \begin{array}{cc} & \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \end{array} & \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

Lengths of the shortest paths with no intermediate vertices ($D^{(0)}$ is simply the weight matrix).

# Floyd's Algorithm



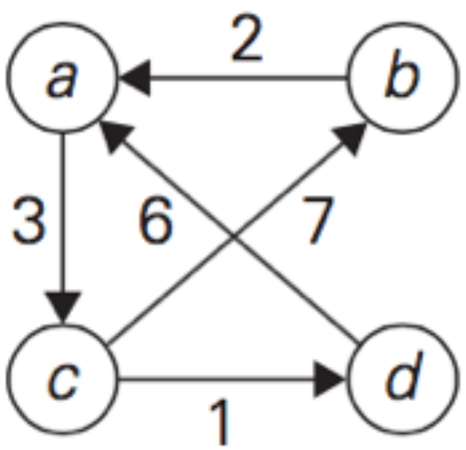

$$D^{(0)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

Lengths of the shortest paths with no intermediate vertices ($D^{(0)}$ is simply the weight matrix).

$$D^{(1)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \mathbf{5} & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \mathbf{9} & 0 \end{bmatrix} \end{array}$$

Lengths of the shortest paths with intermediate vertices numbered not higher than 1, i.e., just *a* (note two new shortest paths from *b* to *c* and from *d* to *c* ).

# Floyd's Algorithm




$$D^{(0)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \left[\begin{array}{cccc} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{array}\right] \end{array}$$

Lengths of the shortest paths with no intermediate vertices ($D^{(0)}$ is simply the weight matrix).

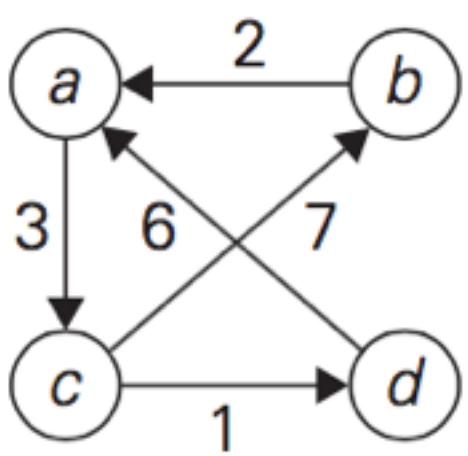
$$D^{(1)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \left[\begin{array}{cccc} 0 & \infty & 3 & \infty \\ 2 & 0 & \mathbf{5} & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \mathbf{9} & 0 \end{array}\right] \end{array}$$

Lengths of the shortest paths with intermediate vertices numbered not higher than 1, i.e., just *a* (note two new shortest paths from *b* to *c* and from *d* to *c* ).

$$D^{(2)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \left[\begin{array}{cccc} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ \mathbf{9} & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{array}\right] \end{array}$$

Lengths of the shortest paths with intermediate vertices numbered not higher than 2, i.e., a and *b* (note a new shortest path from *c* to *a*).

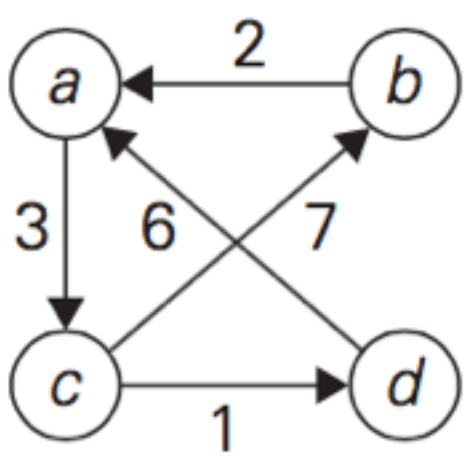
# Floyd's Algorithm




$$D^{(2)} = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & \infty & 3 & \infty \\ b & 2 & 0 & 5 & \infty \\ c & \mathbf{9} & 7 & 0 & 1 \\ d & 6 & \infty & 9 & 0 \end{array}$$

Lengths of the shortest paths with intermediate vertices numbered not higher than 2, i.e., a and *b* (note a new shortest path from *c* to *a*).

# Floyd's Algorithm



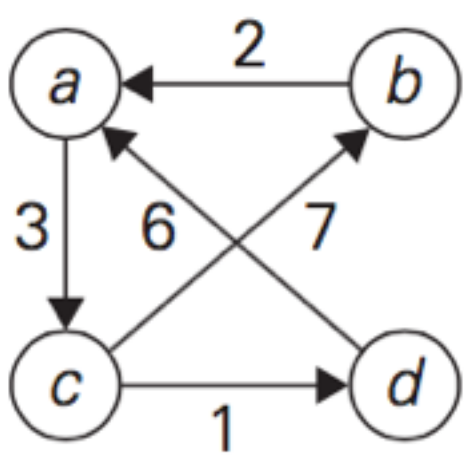

$$D^{(2)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \infty & \boxed{3} & \infty \\ 2 & 0 & \boxed{5} & \infty \\ \boxed{\mathbf{9}} & \boxed{7} & \boxed{0} & \boxed{1} \\ 6 & \infty & \boxed{9} & 0 \end{bmatrix} \end{array}$$

Lengths of the shortest paths with intermediate vertices numbered not higher than 2, i.e., a and *b* (note a new shortest path from *c* to *a*).

$$D^{(3)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \mathbf{10} & 3 & \boxed{\mathbf{4}} \\ 2 & 0 & 5 & \boxed{\mathbf{6}} \\ 9 & 7 & 0 & \boxed{1} \\ \boxed{6} & \boxed{\mathbf{16}} & \boxed{9} & \boxed{0} \end{bmatrix} \end{array}$$

Lengths of the shortest paths with intermediate vertices numbered not higher than 3, i.e., a, *b*, and *c* (note four new shortest paths from *a* to *b*, from a to *d*, from *b* to *d*, and from *d* to *b*).

# Floyd's Algorithm




$$D^{(2)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \infty & \boxed{3} & \infty \\ 2 & 0 & 5 & \infty \\ \mathbf{9} & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix} \end{array}$$

Lengths of the shortest paths with intermediate vertices numbered not higher than 2, i.e., a and *b* (note a new shortest path from *c* to *a*).

$$D^{(3)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \mathbf{10} & 3 & \mathbf{4} \\ 2 & 0 & 5 & \mathbf{6} \\ 9 & 7 & 0 & 1 \\ 6 & \mathbf{16} & 9 & 0 \end{bmatrix} \end{array}$$

Lengths of the shortest paths with intermediate vertices numbered not higher than 3, i.e., a, *b*, and *c* (note four new shortest paths from *a* to *b*, from a to *d*, from *b* to *d*, and from *d* to *b*).

$$D^{(4)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ \mathbf{7} & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{bmatrix} \end{array}$$

Lengths of the shortest paths with intermediate vertices numbered not higher than 4, i.e., a, *b*, *c*, and *d* (note a new shortest path from *c* to *a*).