Introduction to Algorithm Design

Lecture Notes 8

ROAD MAP

- **Dynamic Programming**
  - The Knapsack Problem
  - All Pairs Shortest Paths
  - Optimal Binary Search Tree
  - String Editing
  - Matrix Chain Product

Dynamic Programming

Definition :

- Dynamic programming is an interesting algorithm design technique for *optimizing multistage decision problems*
- Programming in the name of this technique stands for *planning*
  - Does not refer to computer programming
- It is a technique for solving problems with overlapping subproblems
  - Typically these subproblems arise from a recurrence relations
  - Suggests solving each of the smaller subproblems only once and recording the results in a table

Dynamic Programming

Main idea:

- set up a recurrence
  - relating a solution to a larger instance to solutions of some smaller instances
- solve smaller instances once
- record solutions in a table
- extract solution to the initial instance from the table

Example 1: Fibonacci numbers

- Recall definition of Fibonacci numbers:

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

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

$$F(n)$$

$$F(n\text{-}1) \quad + \quad F(n\text{-}2)$$

$$F(n\text{-}2) \quad + \quad F(n\text{-}3) \qquad F(n\text{-}3) \quad + \quad F(n\text{-}4)$$

•••

Example 1: Fibonacci numbers

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

0	**1**	**1**	**. . .**	***F*(*n*-2)**	***F*(*n*-1)**	***F*(*n*)**

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

Efficiency:
- time
- space

Example 2: Binomial Coefficients

- Definition :
  - Binomial coefficient is the number of combinations (subsets) of *k* elements from an *n* element set *(0≤k≤n)*

$$C(n,k) \quad \text{or} \quad \binom{n}{k}$$

  - Binomial coefficient comes from the participation of these numbers in binomial formula

$$(a+b)^n = C(n,0)a^n + \ldots + C(n,i)a^{n-i}b^i + \ldots + C(n,n)b^n$$

  - Recursive definition of Binomial coefficients

$$C(n,k) = C(n-1,k-1) + C(n-1,k) \qquad \text{for} \qquad n > k > 0$$

$$C(n,0) = C(n,n) = 1$$

Example 2: Binomial Coefficients

- Use a table with *n+1* rows and *k+1* columns

	0	1	2	...	$k-1$	k
0	1					
1	1	1				
2	1	2	1			
$\vdots$						
k	1					1
$\vdots$						
$n-1$	1				$C(n-1, k-1)$	$C(n-1, k)$
n	1					$C(n, k)$

Example 2: Binomial Coefficients

ALGORITHM *Binomial*(n, k)
//Computes $C(n, k)$ by the dynamic programming algorithm
//Input: A pair of nonnegative integers $n \geq k \geq 0$
//Output: The value of $C(n, k)$
for $i \leftarrow 0$ **to** n **do**
    for $j \leftarrow 0$ **to** $\min(i, k)$ **do**
        if $j = 0$ **or** $j = i$
            $C[i, j] \leftarrow 1$
        else $C[i, j] \leftarrow C[i-1, j-1] + C[i-1, j]$
return $C[n, k]$

Example 2: Binomial Coefficients

- Analysis :
  - Basic operation in algorithm is *addition*
  - Computing each entry requires one addition
  - First k+1 rows of the table from a triangle while the remaining n-k rows from a rectangle
  - So we split the sum expression into two parts

$$A(n,k) = \sum_{i=1}^{k} \sum_{j=1}^{i-1} 1 + \sum_{i=k+1}^{n} \sum_{j=1}^{k} 1 = \sum_{i=1}^{k} (i-1) + \sum_{i=k+1}^{n} k$$

$$= \frac{k(k-1)}{2} + k(n-k) \in \Theta(nk)$$

Dynamic Programming

Main idea:

- set up a recurrence
  - relating a solution to a larger instance to solutions of some smaller instances
- solve smaller instances once
- record solutions in a table
- extract solution to the initial instance from the table

- Dynamic programming usually used for optimization problems
  - How do we get the recurrence relation?

ROAD MAP

- **Dynamic Programming**
  - **The Knapsack Problem**
  - All Pairs Shortest Paths
  - Optimal Binary Search Tree
  - String Editing
  - Matrix Chain Product

0/1 Knapsack Problem

Definition:

Given n items of

integer weights: $w_1 \quad w_2 \quad \ldots \quad w_n$

values: $v_1 \quad v_2 \quad \ldots \quad v_n$

a knapsack of integer capacity W

find most valuable subset of the items that fit into the knapsack

- *Assume w_1, w_2, …, w_n and W are intergers*
- How can we design a dynamic programming algorithm ?

Dynamic Programming

1. Sequence of decisions

Ex : 0/1 Knapsack problem

- decide values (0 or 1) of x_i ($1 \le i \le n$) one by one

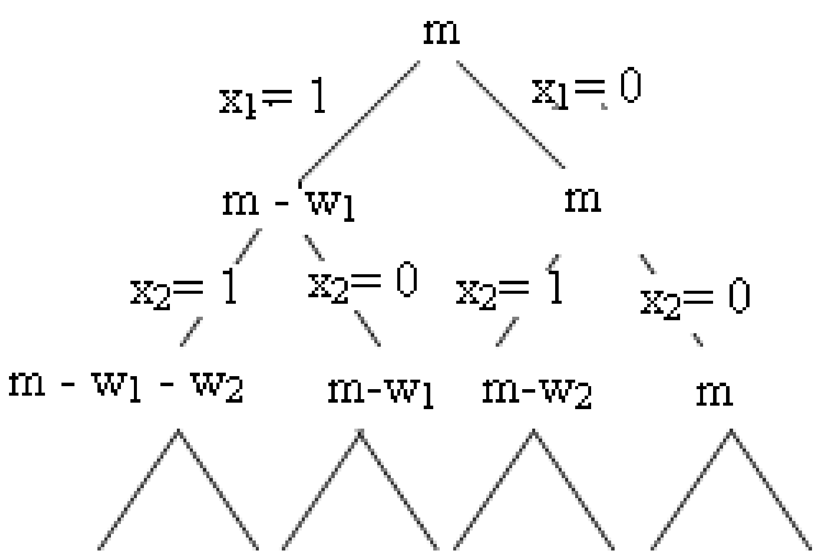
$\sum v_i x_i$ is maximized

- To get the optimal solution
  - No error in decisions
  - Try all possible decisions

Dynamic Programming

2. Try all decision sequences

Ex : 0/1 Knapsack problem



Dynamic programming

- enumerate the decisions (problems arise after the decision) that can lead to optimal solution and reuse them

Dynamic Programming

3. <u>Principle of Optimality</u>

   Assume an optimal sequence of decisions. Whatever the initial state and first decision the remaining sequence of decisions is an optimal sequence from the state after the first decision

   Ex : 0/1 Knapsack Problem

   $\mathbf{x_1, x_2, \ldots, x_n}$ is optimal for $\mathbf{KNAP(1,n,W)}$

   if $\mathbf{x_n = 0}$

   $\mathbf{x_1, \ldots, x_{n-1}}$ is optimal for $\mathbf{KNAP(1,n-1,W)}$

   if $\mathbf{x_1 = 1}$

   $\mathbf{x_1, \ldots, x_{n-1}}$ is optimal for $\mathbf{KNAP(1,n-1,W-w_1)}$

Dynamic Programming

Ex : Shortest Path Problem

in a directed graph if

$i, i_1, i_2, \ldots, i_k, j$ is shortest path from i to j

then

$i_1, i_2, \ldots, i_k, j$ is shortest path from i_1 to j

OR

$$\underbrace{i, i_1, i_2, \ldots k,}_{\substack{\text{shortest path} \\ \text{from } i \text{ to } k}} \underbrace{p_1, p_2, \ldots, j}_{\substack{\text{shortest path} \\ \text{from } k \text{ to } j}}$$

Dynamic Programming

4. Write a recurrence relation for the optimal solution

$S_0 \rightarrow$ initial state

$d_i \rightarrow$ decisions to be made $1<i<n$

$D_1 \rightarrow \{r_1\ r_2\ \ldots\ r_k\}$ possible decisions for d_1

$S_i \rightarrow$ state after $d_1 = r_i$

$T_i \rightarrow$ optimal sequence of decisions after S_i

By principal of optimality:

optimal sequence from S_0 is

the best of $r_i T_i \quad 1<i<k$

Dynamic Programming

Ex : Shortest Path Problem

$$i, \underbrace{k, p_1, \ldots, p_l, j}_{\text{optimal}}$$

$$P_{ij} = \min_{k \in A_1} \{ c_{ik} + P_{kj} \}$$

A_1 = set of vertices from i

Dynamic Programming

Ex : 0/1 Knapsack problem

- Consider an instance defined by the first *i* items $1 \le i \le n$
  - with weights $w_1, \ldots, w_i$
  - values $v_1, \ldots, v_i$
  - capacity j $1 \le j \le W$
- V(i,j) be the value of an optimal solution to this instance

V(n,W) → optimal value for **`KNAP(1,n,W)`**
V(i,j) → optimal value for **`KNAP(1,i,j)`**

Knapsack Problem

Approach :

We can divide all subsets of the first *i* items that fit the knapsack of capacity *j* into two categories

1. Among the subsets that *do not* include the i^{th} item,
   - the value of an optimal subset is, $V(i-1, j)$
2. Among the subsets that *do* include the i^{th} item,
   - an optimal subset is made up of
     - this item and
     - an optimal subset of the first $i-1$ items that fit into the knapsack of capacity $j-w_i$ ($j-w_i \geq 0$)
   - The value of such an optimal subset is $v_i + V(i-1, j-w_i)$

$$V(n,W) \rightarrow \max \{V(n-1,W),\ V(n-1,W-w_n)+v_n\}$$

$$V(i,j) \rightarrow \max \{V(i-1,j),\ V(i-1,j-w_i)+v_i\}$$

Knapsack Problem

- So, the following recurrence

$$V[i,j] = \begin{cases} \max\{V[i-1,j], v_i + V[i-1,j-w_i]\} & \text{if } j - w_i \geq 0 \\ V[i-1,j] & \text{if } j - w_i < 0 \end{cases}$$

- Initial conditions

$$V[0,j] = 0 \quad \text{for} \quad j \geq 0$$

$$V[i,0] = 0 \quad \text{for} \quad i \geq 0$$

How to solve this recurrence??

Knapsack Problem

		0	$j-w_i$	j	W
	0	0	0	0	0
	$i-1$	0	$V[i-1, j-w_i]$	$V[i-1, j]$	
w_i, v_i	i	0		$V[i, j]$	
	n	0			goal

Table for solving the knapsack problem by dynamic programming

Knapsack Problem by DP (example)

Example: Knapsack of capacity $W = 5$

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

capacity j

	i	0	1	2	3	4	5
	0						
$w_1 = 2, v_1= 12$	1						
$w_2 = 1, v_2= 10$	2						
$w_3 = 3, v_3= 20$	3						
$w_4 = 2, v_4= 15$	4						?

- **Example :**

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

capacity $W = 5$

The given instance

	capacity j						
	i	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

Maximal value is ***V[4, 5] = 37***

Dynamic programming table

Knapsack Problem

- Analysis :
  - Time efficiency and space effciency of this algorithm is $\Theta(nW)$
  - The time needed to find the composition of an optimal solution is in $O(n+W)$

ROAD MAP

- **Dynamic Programming**
  - Computing a Binomial Coefficient
  - The Knapsack Problem
  - **All Pairs Shortest Paths**
    - **Floyd's Algorithm**
  - Optimal Binary Search Tree
  - String Editing
  - Matrix Chain Product

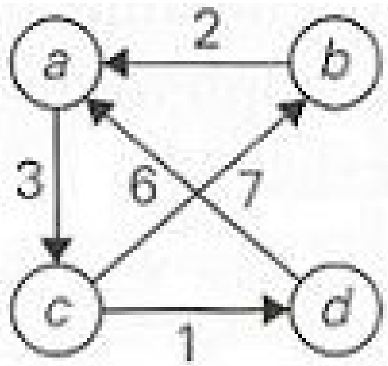
All-Pair Shortest Path Problem

- Definition:
  - Given a weighted graph G,
    - G has no cycle with negative length
  - Compute the distances (***the length of the shortest paths)*** between every pair of vertices in a graph *G*

Specifically:

- Find D = Distance matrix
  where d_{ij} = length of the shortest path from i to j

All-Pair Shortest Path Problem



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

Digraph **weight matrix** **distance matrix**

All-Pair Shortest Path Problem

- What is the sequence of decisions?
- What are possible choices at each decision point?
- What about principle of optimality…
- How to write the recurrence relation?

All-Pair Shortest Path Problem

Idea:

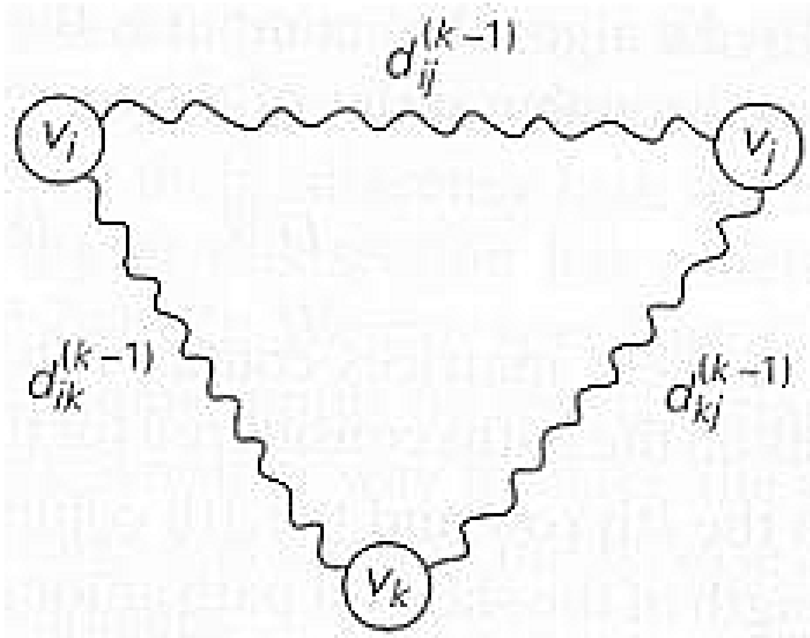
- Compute D through a series of n-by-n matrices

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

- where $d_{ij}^{(k)}$ = the length of the shortest path from i^{th} vertex to j^{th} vertex that use only vertices among 1,…,k as intermediate
  - each intermediate vertex, if any, numbered not higher than k
  - k is the largest index on the path
- Optimal path from i to j contains no cycle
  - Vertex k appears only once on the path
- Because of the principle of optimality

$$\underbrace{i \quad \ldots \quad}_{\text{sp}} k \underbrace{\quad \ldots \quad j}_{\text{sp}}$$

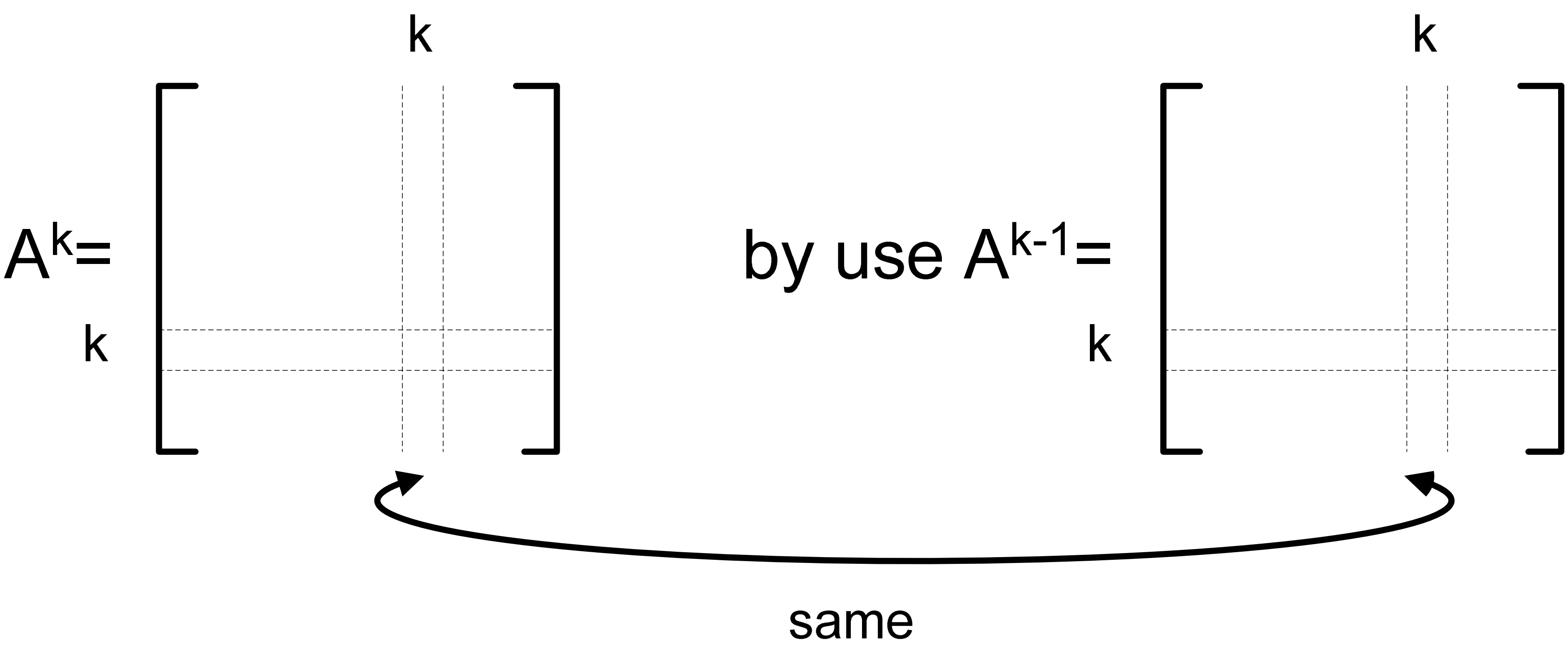
Floyd's Algorithm

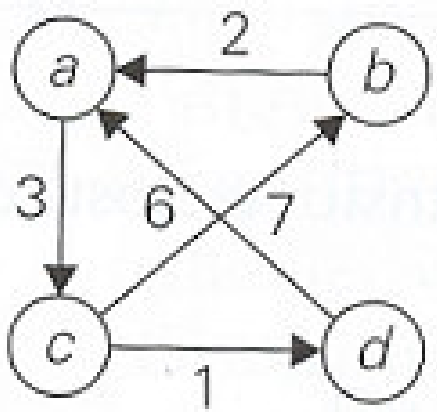


$$d_{ij}^{(k)} = \min\left\{ d_{ij}^{(k-1)} ,\quad \mathrm{d}_{ik}^{(k-1)} + \mathrm{d}_{kj}^{(k-1)} \right\} \quad \text{for} \quad k \geq 1,$$

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

All-Pair Shortest Path Problem





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

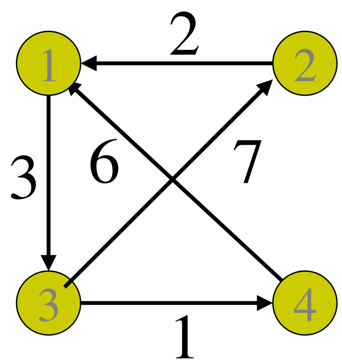
$$D^{(1)} = \begin{array}{cc} & \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \end{array} & \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \mathbf{5} & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \mathbf{9} & 0 \end{bmatrix} \end{array}$$

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

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

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

Floyd's Algorithm (example)



$$D^{(0)} = \begin{matrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{matrix}$$

$$D^{(1)} = \begin{matrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \mathbf{5} & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \mathbf{9} & 0 \end{matrix}$$

$$D^{(2)} = \begin{matrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ \mathbf{9} & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{matrix}$$

$$D^{(3)} = \begin{matrix} 0 & \mathbf{10} & 3 & \mathbf{4} \\ 2 & 0 & 5 & \mathbf{6} \\ 9 & 7 & 0 & 1 \\ 6 & \mathbf{16} & 9 & 0 \end{matrix}$$

$$D^{(4)} = \begin{matrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ \mathbf{7} & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{matrix}$$

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 ← W //is not necessary if W can be overwritten
for k ← 1 to n do
    for i ← 1 to n do
        for j ← 1 to n do
            D[i, j] ← min{D[i, j], D[i, k] + D[k, j]}
return D
```

Time efficiency is cubic

All-Pair Shortest Path Problem

- Floyd's algorithm finds the lengths of shortest paths.
  - What is the complexity?
    - Time
    - Space
- How to find the actual paths?
  - One of them
  - All of them
  - What is the complexity?
    - Time
    - Space