Topic 6

Dynamic Programming

Dynamic Programming

Dynamic Programming is a general algorithm design technique for solving problems defined by or formulated as recurrences with overlapping subinstances

- Invented by American mathematician Richard Bellman in the 1950s to solve optimization problems and later assimilated by CS

- "Programming" here means "planning"

- 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 that table

Example: Fibonacci numbers

- **Recall definition of Fibonacci numbers:**

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

2^n

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

$F(n)$

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

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

...

Example: Fibonacci numbers (cont.)

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

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

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

Efficiency:
- time n ✓
- space n ✓

What if we solve it recursively? 2^n ✗

Examples of DP algorithms

1 • Computing a binomial coefficient

2 • Longest common subsequence

3 • Warshall's algorithm for transitive closure

4 • Floyd's algorithm for all-pairs shortest paths

5 • Constructing an optimal binary search tree

6 • Some instances of difficult discrete optimization problems:
  6.1 - traveling salesman
  6.2 - knapsack

Computing a binomial coefficient by DP

Binomial coefficients are coefficients of the binomial formula:

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

علاقة $a^n + nab + b^n$

Recurrence: $C(n,k) = \underline{C(n-1,k) + C(n-1,k-1)}$ for $n > k > 0$

$\underline{C(n,0) = 1}$, $\underline{C(n,n) = 1}$ for $n \geq 0$

Value of $C(n,k)$ can be computed by filling a table:

	0	1	2	. . .	k-1	$\boxed{k}$
0	1					
1	1	1	2			
2	1	$\boxed{2}$				
.		$\boxed{3}$				
.						
n-1					$C(n-1,k-1)$	$C(n-1,k)$
$\boxed{n}$						$C(n,k)$

$C(2,1)$

$n = 2$

$K = 1$

$C(2-1,1) +$

$C(2-1,1-1) =$

$C(1,1) + C(1,0)$

$= 1 + 1 = \boxed{2}$

Computing $C(n,k)$: pseudocode and analysis

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** n
  for $j \leftarrow 0$ **to** min(i, k) **do** k
    if $j = 0$ **or** $j = i$ nk
      $C[i, j] \leftarrow 1$
    else $C[i, j] \leftarrow C[i-1, j-1] + C[i-1, j]$
return $C[n, k]$

Time efficiency: $\Theta(nk)$ ✓

Space efficiency: $\Theta(nk)$ ✓

Knapsack Problem by DP

$V(3,10)$

$V(2,10)$ | $20 + V(2,7)$

Given n items of

- integer weights: $w_1 \quad w_2 \quad \overset{3}{\ldots} \quad w_n$
- values: $v_1 \quad v_2 \quad \overset{20}{\ldots} \quad v_n$
- a knapsack of integer capacity W

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

Consider instance defined by first i items and capacity j ($j \leq W$).

Let $V[i,j]$ be optimal value of such an instance. Then

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

المكسب + العناصر الباقية

أو

في حالة الوزن كان كبير

1 2

Initial conditions: $V[0,j] = 0$ and $V[i,0] = 0$

Knapsack Problem by DP

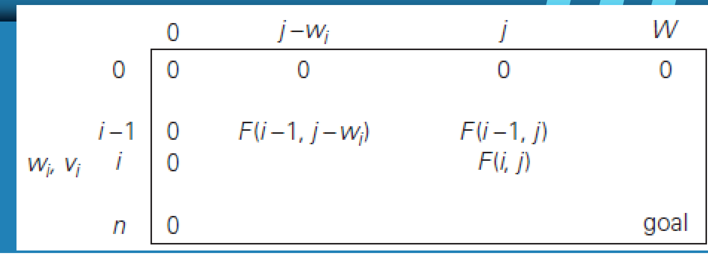
Solution for i-1 items and capacity j is known — $V[i-1, j]$

Consider picking item i

- It is too big to fit in, i.e., $j < w(i)$ — $V[i, j] = V[i-1, j]$
- It can fit in, i.e., $j >= w(i)$
  - Leave it: $V[i, j] = V[i-1, j]$
  - Pick it: $V[i, j] = v(i) + V[i-1, j-w(i)]$

Go with the better option.

Knapsack Problem by DP (example)



Example: Knapsack capacity $W = 5$

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

$$V[i,j] = \begin{cases} \max\left(V[i-1,j],\ v_i + V[i-1,j-w_i]\right) & \text{if } j \ge w_i \\ V[i-1,j] & \text{if } j < w_i \end{cases}$$

$V(4,5)$

$V(3,5)$ $\quad$ $\boxed{15} + V(3,3)$

$\boxed{32}$ $\quad$ $\boxed{22}$

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 ?

$(22+15)$

Backtracing finds the actual optimal subset, i.e. solution.

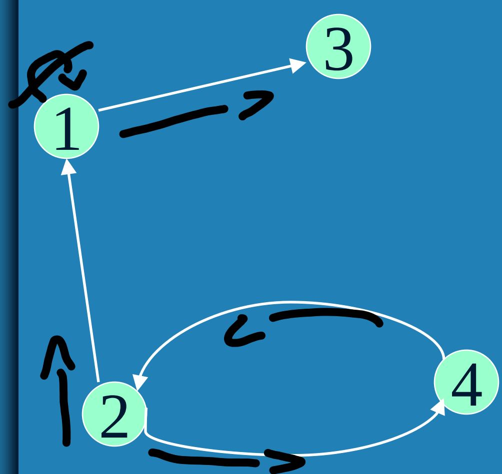
Knapsack Problem by DP (pseudocode)

```
Algorithm DPKnapsack(w[1..n], v[1..n], W)
    var V[0..n,0..W], P[1..n,1..W]: int
    for j := 0 to W do
        V[0,j] := 0
    for i := 0 to n do
        V[i,0] := 0
    for i := 1 to n do
        for j := 1 to W do
            if w[i] ≤ j and v[i] + V[i-1,j-w[i]] > V[i-1,j] then
                V[i,j] := v[i] + V[i-1,j-w[i]]; P[i,j] := j-w[i]
            else
                V[i,j] := V[i-1,j]; P[i,j] := j
    return V[n,W] and the optimal subset by backtracing
```

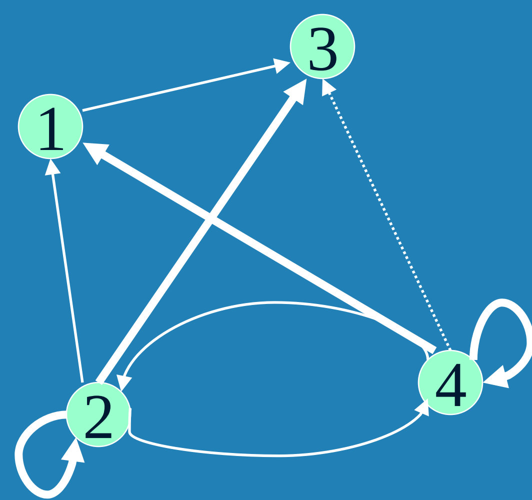
Running time and space:
$\boxed{O(nW)}$

Warshall's Algorithm: Transitive Closure

- Computes the transitive closure of a relation
- Alternatively: existence of all nontrivial paths in a digraph
- Example of transitive closure:



	1	2	3	4
1	0	0	1	0
2	1	0	0	1
3	0	0	0	0
4	0	1	0	0

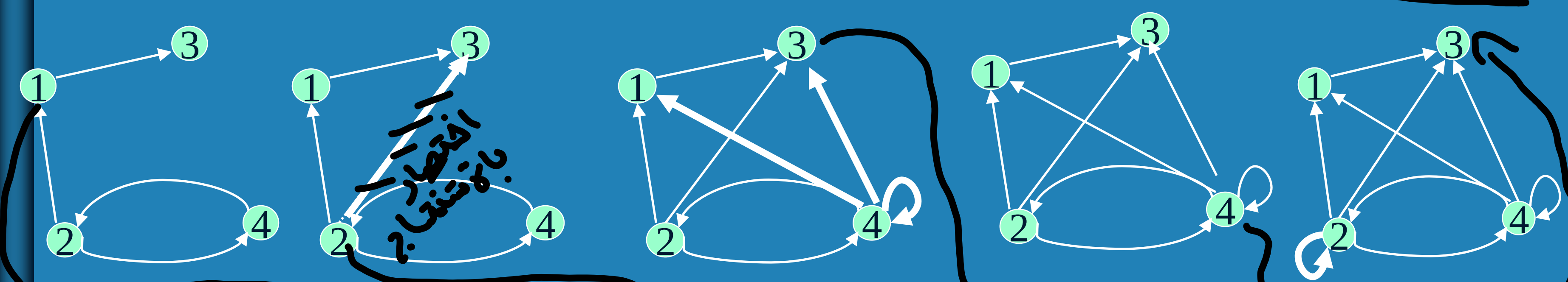


0	0	1	0
1	**1**	**1**	1
0	0	0	0
1	1	**1**	**1**

Warshall's Algorithm

Constructs transitive closure T as the last matrix in the sequence of n-by-n matrices $R^{(0)}, \ldots, R^{(k)}, \ldots, R^{(n)}$ where $R^{(k)}[i,j] = 1$ iff there is nontrivial path from i to j with only the first k vertices allowed as intermediate

Note that $R^{(0)} = A$ (adjacency matrix), $R^{(n)} = T$ (transitive closure)



$$R^{(0)} = \begin{matrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{matrix} \qquad R^{(1)} = \begin{matrix} 0 & 0 & 1 & 0 \\ 1 & 0 & \mathbf{1} & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{matrix} \qquad R^{(2)} = \begin{matrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ \mathbf{1} & 1 & \mathbf{1} & \mathbf{1} \end{matrix} \qquad R^{(3)} = \begin{matrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{matrix} \qquad R^{(4)} = \begin{matrix} 0 & 0 & 1 & 0 \\ 1 & \mathbf{1} & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{matrix}$$

Warshall's Algorithm (recurrence)

On the k-th iteration, the algorithm determines for every pair of vertices i, j if a path exists from i and j with just vertices $1,\ldots,k$ allowed as intermediate

R^0 $\begin{bmatrix} 0 & 0 & \boxed{1} \\ \boxed{1} & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$

R^3 R^2

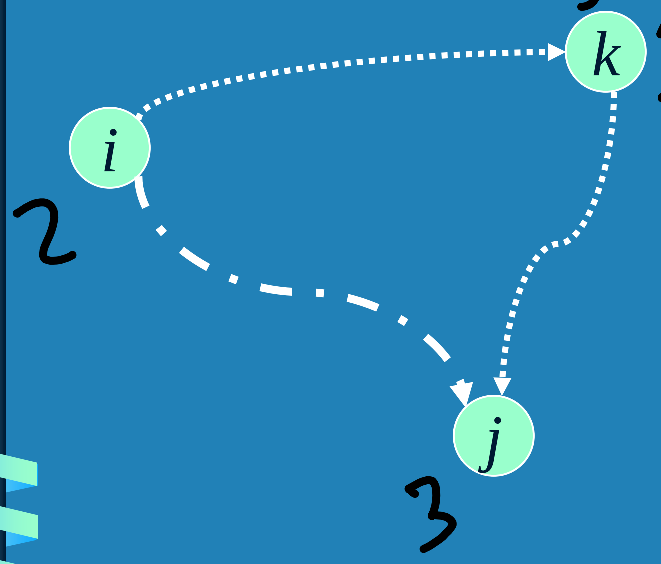
$$R^{(k)}[i,j] = \begin{cases} R^{(k-1)}[i,j] & \textbf{(path using just } 1,\ldots,k\textbf{-1)} \\ \text{or} & \\ R^{(k-1)}[i,k] \text{ and } R^{(k-1)}[k,j] & \textbf{(path from } i \textbf{ to } k \textbf{ and from } k \textbf{ to } j \textbf{ using just } 1,\ldots,k\textbf{-1)} \end{cases}$$

AND=1

1 $)=1$

2,1 1,3

1 R^1



Initial condition?

k=2
i=2
j=3

Warshall's Algorithm (matrix generation)

Recurrence relating elements $R^{(k)}$ to elements of $R^{(k-1)}$ is:

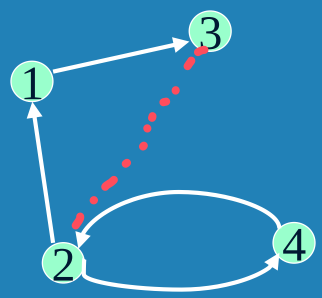
$$R^{(k)}[i,j] = R^{(k-1)}[i,j] \text{ or } (R^{(k-1)}[i,k] \text{ and } R^{(k-1)}[k,j])$$

It implies the following rules for generating $R^{(k)}$ from $R^{(k-1)}$:

Rule 1 If an element in row i and column j is 1 in $R^{(k-1)}$, it remains 1 in $R^{(k)}$

Rule 2 If an element in row i and column j is 0 in $R^{(k-1)}$, it has to be changed to 1 in $R^{(k)}$ if and only if the element in its row i and column k and the element in its column j and row k are both 1's in $R^{(k-1)}$

Warshall’s Algorithm (example)



$$R^{(0)} = \begin{array}{cc} & \begin{matrix} 1 & & & \end{matrix} \\ 1 & \begin{matrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{matrix} \end{array}$$

$$R^{(1)} = \begin{array}{cc} & \begin{matrix} & 2 & & \end{matrix} \\ \begin{matrix} \\ 2 \\ \\ \\ \end{matrix} & \begin{matrix} 0 & 0 & 1 & 0 \\ 1 & 0 & \mathbf{1} & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{matrix} \end{array}$$

$$R^{(2)} = \begin{matrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ \mathbf{1} & 1 & \mathbf{1} & \mathbf{1} \end{matrix}$$

non

$$R^{(3)} = \begin{matrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{matrix}$$

$$R^{(4)} = \begin{matrix} 0 & 0 & 1 & 0 \\ 1 & \mathbf{1} & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{matrix}$$

end

transitive closure

Warshall's Algorithm (pseudocode and analysis)

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

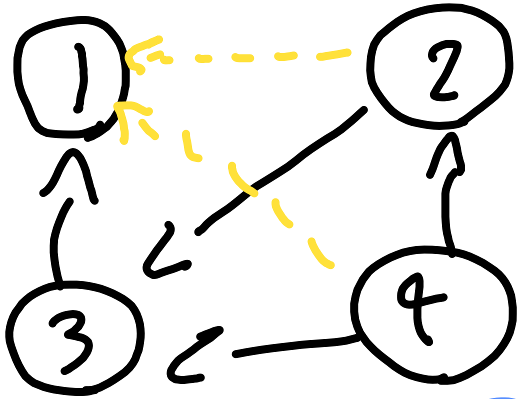
//Implements Warshall's algorithm for computing the transitive closure
//Input: The adjacency matrix A of a digraph with n vertices
//Output: The transitive closure of the digraph

$R^{(0)} \leftarrow A$
for $k \leftarrow 1$ **to** n **do** n
  for $i \leftarrow 1$ **to** n **do** n
    for $j \leftarrow 1$ **to** n **do** n
      $R^{(k)}[i, j] \leftarrow R^{(k-1)}[i, j]$ **or** $(R^{(k-1)}[i, k]$ $\boxed{\textbf{and}}$ $R^{(k-1)}[k, j])$
return $R^{(n)}$

$\Theta(n^3)$

Time efficiency: $\Theta(n^3)$

Space efficiency: Matrices can be written over their predecessors (with some care), so it's $\Theta(n\text{^}2)$.



$$R^0 \quad \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & \\ 2 & \\ 3 & \\ 4 & \end{array} \hspace{-4.5em} \begin{array}{c} \\ \end{array}$$

$$R^0 = \begin{array}{cc} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

$$R^1 = \begin{array}{cc} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

$$R^2 = \begin{array}{cc} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

$$R^3 = \begin{array}{cc} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

$$R^4 = \begin{array}{cc} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

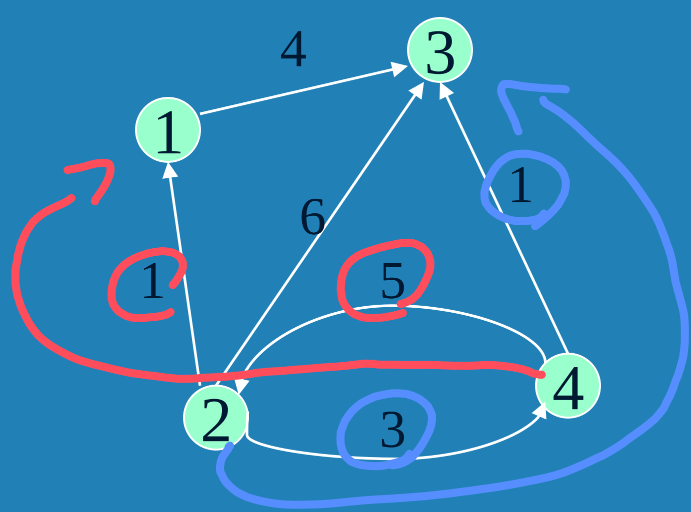
transitive closure $\longleftarrow$

Floyd's Algorithm: All pairs shortest paths

Problem: In a weighted (di)graph, find shortest paths between every pair of vertices

Same idea: construct solution through series of matrices $D^{(0)}, \ldots, D^{(n)}$ using increasing subsets of the vertices allowed as intermediate

Example:

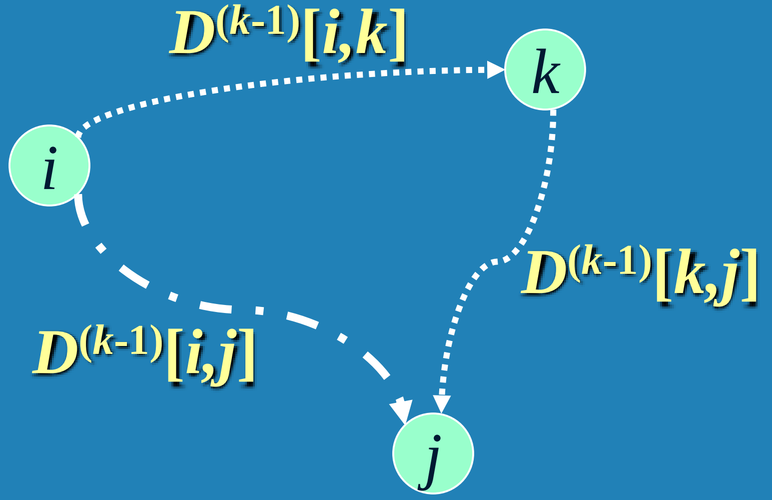


	1	2	3	4
1	0	∞	4	∞
2	1	0	4	3
3	∞	∞	0	∞
4	6	5	1	0

Floyd's Algorithm (matrix generation)

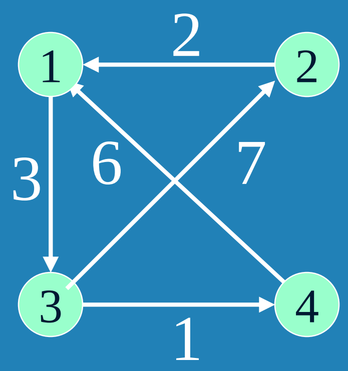
On the k-th iteration, the algorithm determines shortest paths between every pair of vertices i, j that use only vertices among $1,\ldots,k$ as intermediate

$$D^{(k)}[i,j] = \min \{D^{(k-1)}[i,j],\ D^{(k-1)}[i,k] + D^{(k-1)}[k,j]\}$$



Initial condition?

Floyd’s Algorithm (example)



$$D^{(0)} = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & \infty & 3 & \infty \\ 2 & 2 & 0 & \infty & \infty \\ 3 & \infty & 7 & 0 & 1 \\ 4 & 6 & \infty & \infty & 0 \end{array}$$

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

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

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

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

$6+1=7<9$

(ناخذ الأصغر)

Floyd's Algorithm (pseudocode and analysis)

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

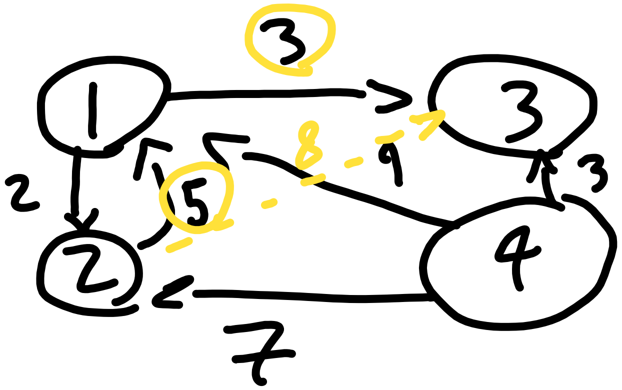
```
//Implements Floyd's algorithm for the all-pairs shortest-paths problem
//Input: The weight matrix W of a graph with no negative-length cycle
//Output: The distance matrix of the shortest paths' lengths
D ← W //is not necessary if W can be overwritten
for k ← 1 to n do                                   n
    for i ← 1 to n do                                 n        Θ(n^3)
        for j ← 1 to n do                               n
            D[i, j] ← min{D[i, j], D[i, k] + D[k, j]}
return D    ~~If D[i,k] + D[k,j] < D[i,j] then P[i,j] ← k~~
```

Time efficiency: $\Theta(n^3)$

Space efficiency: Matrices can be written over their predecessors $\Theta(n^2)$

Since the superscripts *k* or *k-1* make no difference to $D[i,k]$ and $D[k,j]$.

Note: Works on graphs with negative edges but without negative cycles.
Shortest paths themselves can be found, too. How?



$$D^0 \quad \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 2 & 3 & \infty \\ 2 & 5 & 0 & \boxed{\infty} & \infty \\ 3 & \infty & \infty & 0 & \infty \\ 4 & 9 & 7 & 3 & 0 \end{array} \longrightarrow$$

$$D^1 \quad \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 2 & 3 & \infty \\ 2 & 5 & 0 & 8 & \infty \\ 3 & \infty & \infty & 0 & \infty \\ 4 & 9 & 7 & 3 & 0 \end{array} \longrightarrow$$

$$D^2 \quad \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 2 & 3 & \infty \\ 2 & 5 & 0 & 8 & \infty \\ 3 & \infty & \infty & 0 & \infty \\ 4 & 9 & 7 & 3 & 0 \end{array}$$

$$D^3 \quad \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 2 & 3 & \infty \\ 2 & 5 & 0 & 8 & \infty \\ 3 & \infty & \infty & 0 & \infty \\ 4 & 9 & 7 & 3 & 0 \end{array} \longrightarrow$$

$$D^4 \quad \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 2 & 3 & \infty \\ 2 & 5 & 0 & 8 & \infty \\ 3 & \infty & \infty & 0 & \infty \\ 4 & 9 & 7 & 3 & 0 \end{array}$$