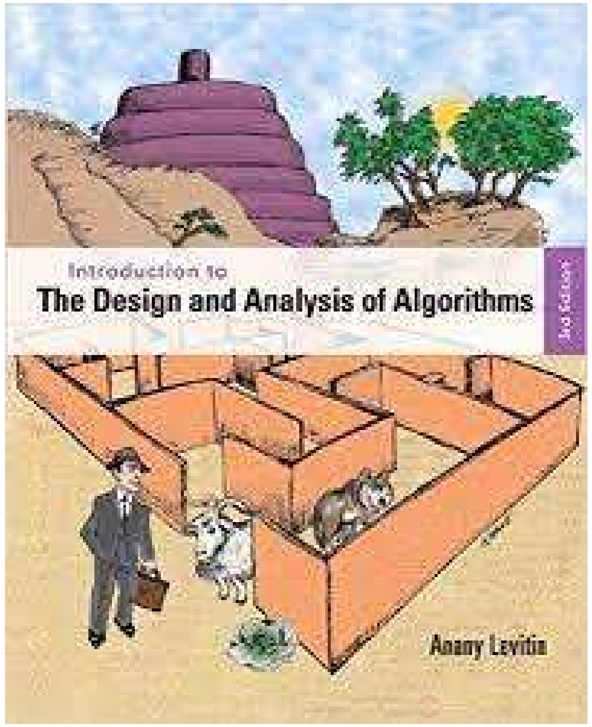
8-Dynamic Programming

A. Levitin "Introduction to the Design & Analysis of Algorithms," 3rd ed., Ch. 1



Dynamic Programming

Dynamic Programming is a general algorithm design technique for solving problems defined by recurrences with overlapping subproblems

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

- "Programming" here means "planning"

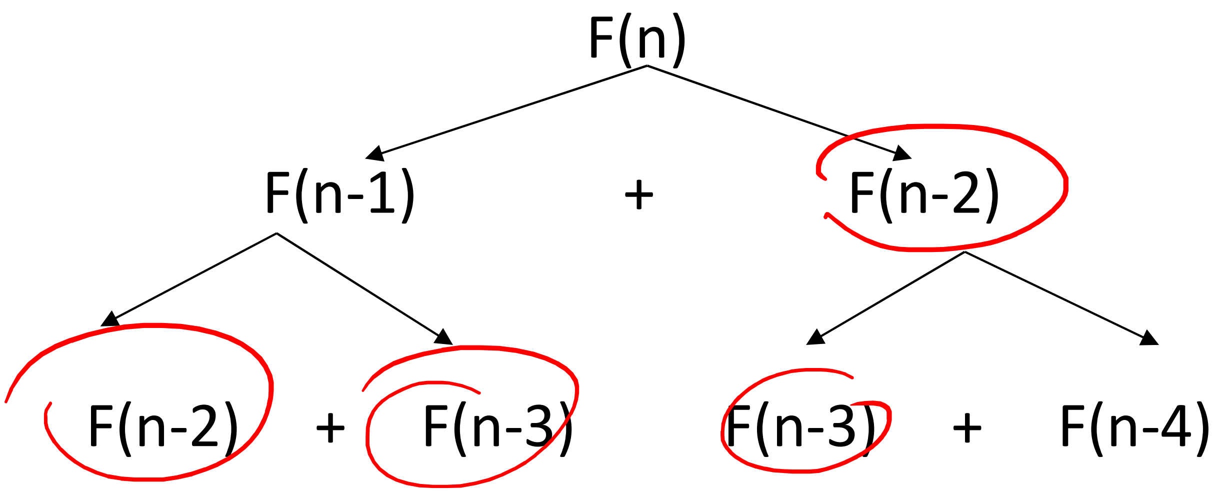
- Main idea:
  - set up a recurrence relating a solution to a larger instance to solutions of some smaller instances
  - solve smaller instances once
  - record solutions in a table
  - extract solution to the initial instance from that table

Example 1: Fibonacci numbers

- Recall definition of Fibonacci numbers:

$$F(n) = F(n-1) + F(n-2)$$
$$F(0) = 0$$
$$F(1) = 1$$

- Computing the nth Fibonacci number recursively (top-down):



Example 1: Fibonacci numbers

- Computing the nth Fibonacci number using bottom-up iteration and recording results:

$F(0) = 0$

$F(1) = 1$

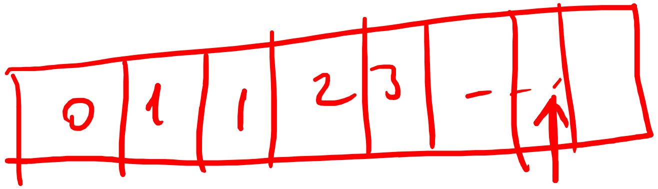
$F(2) = 1+0 = 1$

…

$F(n-2) =$

$F(n-1) =$

$F(n) = F(n-1) + F(n-2)$



$F(n)$

Efficiency:

- time $\Theta(n)$
- space $\Theta(n)$

Example 2: Coin-row problem

There is a row of n coins whose values are some positive integers $c_1, c_2, \dots, c_n$, not necessarily distinct. The goal is to pick up the maximum amount of money subject to the constraint that no two coins adjacent in the initial row can be picked up.

c_1 c_2 c_3 c_4 c_5 c_6

E.g.: 5, 1, 2, 10, 6, 2. What is the best selection?

recurrence relation.

Brute-force

$\{5\} \to 5$

~~{5,1}~~

~~{5,1,2}~~

$\boxed{\{5, 10, 2\} \Rightarrow 17}$

(*) $F(i)$ = max amount that can be collected using the first i coins.

$$\overbrace{c_1\ c_2\ c_3 - }^{F(n-2)}\ \cancel{c_{n-1}}\ \boxed{c_n}$$

$$\underbrace{c_1\ c_2 c_3 - - - c_{n-1}}_{F(n-1)}\ \cancel{c_n}$$

$F(0)=0$

$F(1)=c_1$

$$\begin{cases} F(n) = c_n + F(n-2) & \text{if I select } c_n \\ F(n) = F(n-1) & \text{if I don't select } c_n \end{cases}$$

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

$F(0) = 0 \qquad F(1) = c_1$

DP solution to the coin-row problem

Let $F(n)$ be the maximum amount that can be picked up from the row of n coins. To derive a recurrence for $F(n)$, we partition all the allowed coin selections into two groups:

those without last coin – the max amount is ?
those with the last coin -- the max amount is ?

Thus, we have the following recurrence

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

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

DP solution to the coin-row problem

$$
\begin{aligned}
F(n) &= \max\{c_n + F(n-2), F(n-1)\} \; for\, n > 1, \\
F(0) &= 0, \; F(1) = c_1
\end{aligned}
$$

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

		c_1	c_2	c_3	c_4	c_5	c_6
index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15	15	17

$F(2) = \max\{\underbrace{c_2 + F(0)}_{1+0}, \underbrace{F(1)}_{5}\} = 5$

$F(3) = \max\{\underbrace{c_3 + F(1)}_{2+5}, \underbrace{F(2)}_{5}\} = 7$

$F(4) = \max\{\underbrace{c_4 + F(2)}_{10+5}, \underbrace{F(3)}_{7}\} = 15$

$F(5) = \max\{\underbrace{c_5 + F(3)}_{6+7}, \underbrace{F(4)}_{15}\} = 15$

$F(6) = \max\{\underbrace{c_6 + F(4)}_{2+15}, \underbrace{F(5)}_{15}\} = 17$

Max amount: 17

Coins of optimal solution: c_6, c_4, c_1
(backward)

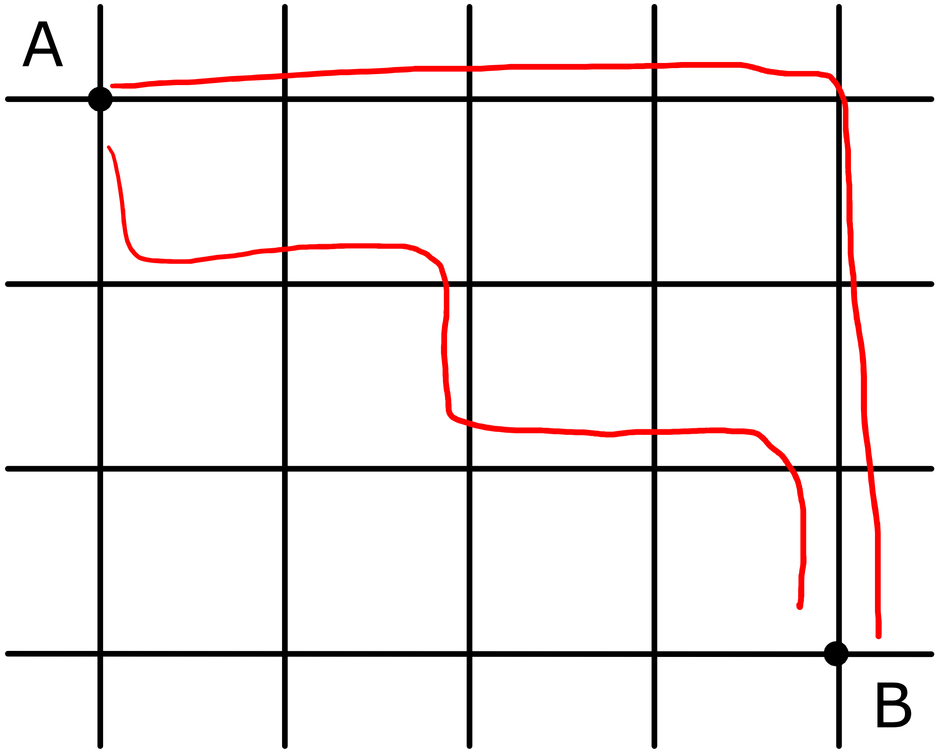
Time efficiency: $\Theta(n)$

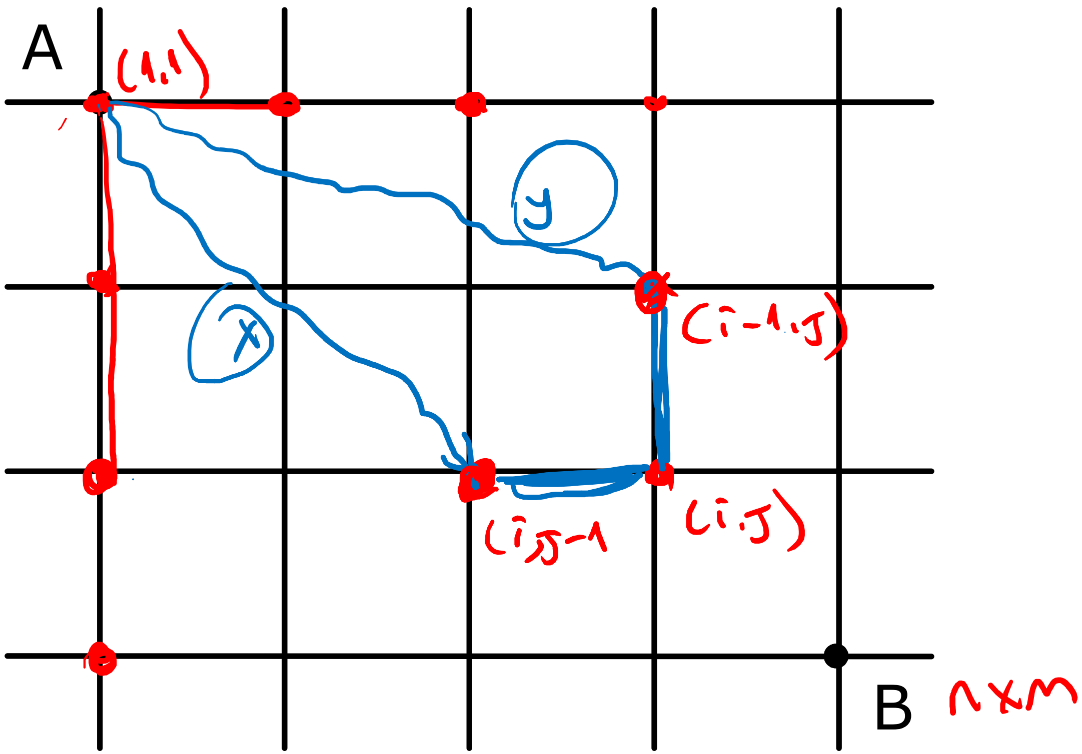
Space efficiency: $\Theta(n)$

Note: All smaller instances were solved

Example 3: Path counting

Consider the problem of counting the number of shortest paths from point A to point B in a city with perfectly horizontal streets and vertical avenues

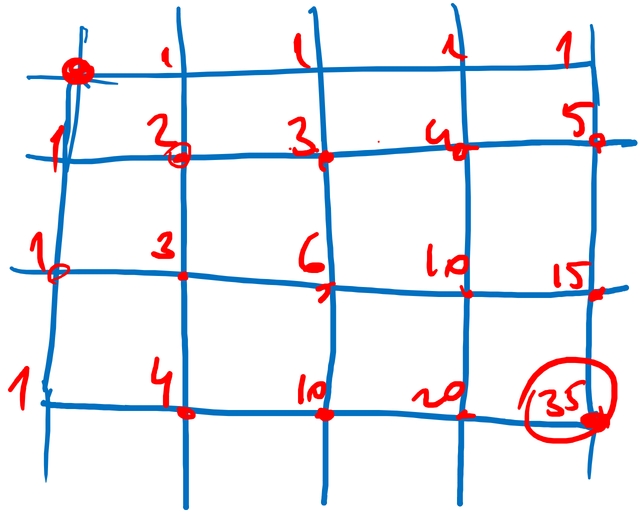




$P(i,j)$: # of paths from $(1,1)$ to (i,j)

$P(i,1) = 1$

$P(1,j) = 1$



$$P(i,j) = P(i-1,j) + P(i,j-1)$$

time complexity: $\Theta(mn)$

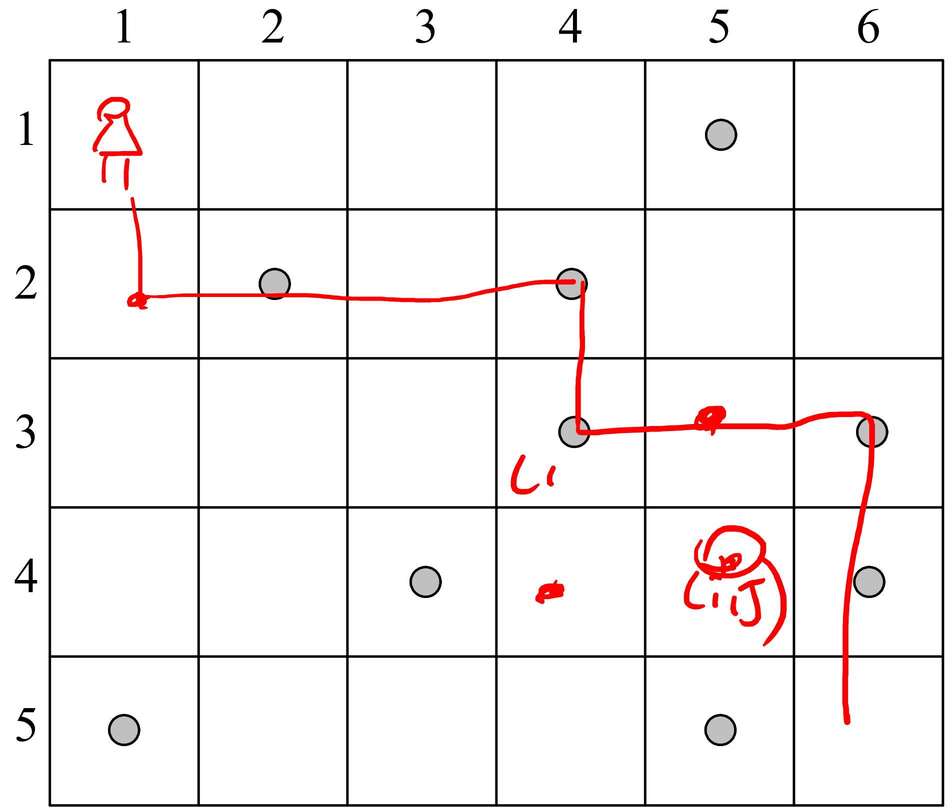
space complexity: $\Theta(mn)$

Example 4: Coin-collecting by robot

Several coins are placed in cells of an $n \times m$ board.

A robot, located in the upper left cell of the board, needs to collect as many of the coins as possible and bring them to the bottom right cell.

On each step, the robot can move either one cell to the right or one cell down from its current location.



Solution to the coin-collecting problem

Let $F(i, j)$ be the largest number of coins the robot can collect and bring to cell (i, j) in the ith row and jth column.

$c_{ij} \in \{0,1\}$

The largest number of coins that can be brought to cell (i, j):

from the left neighbor ?

from the neighbor above?

The recurrence:

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

where $c_{ij} = 1$ if there is a coin in cell (i, j), and $c_{ij} = 0$ otherwise

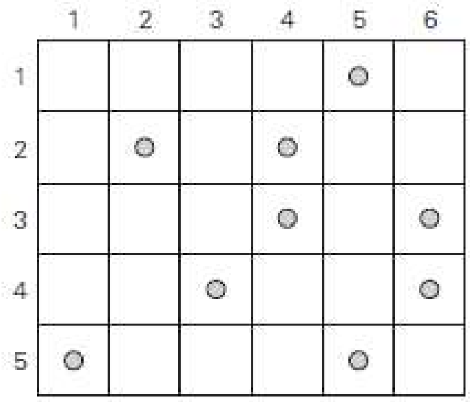
$$F(0,j) = 0 \; for\ 1 \le j \le m \text{ and } F(i,0) = 0 \; for\ 1 \le i \le n.$$

Solution to the coin-collecting problem

(a) Coins to collect.

(b) Dynamic programming algorithm results.

(c) Two paths to collect 5 coins, maximum number of coins possible.

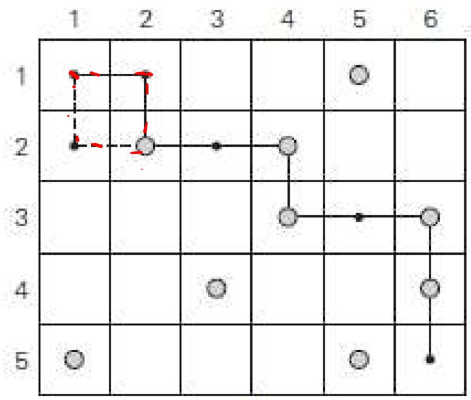


(a)

	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)

n x m



(c)

backtracing

$$\boxed{\begin{array}{l} F(i, j) = \max\{F(i-1, j), F(i, j-1)\} + c_{ij} \quad \text{for } 1 \le i \le n,\ 1 \le j \le m \\ F(0, j) = 0 \ \text{ for } 1 \le j \le m \quad \text{and} \quad F(i, 0) = 0 \ \text{ for } 1 \le i \le n, \end{array}}$$

time complexity $\Theta(mn)$

space complexity $\Theta(mn)$

Other examples of DP algorithms

- Computing a binomial coefficient (# 9, Exercises 8.1)
- General case of the change making problem (Sec. 8.1)
- Some difficult discrete optimization problems:
  - knapsack (Sec. 8.2)
  - traveling salesman
- Constructing an optimal binary search tree (Sec. 8.3)
- Warshall's algorithm for transitive closure (Sec. 8.4)
- Floyd's algorithm for all-pairs shortest paths (Sec. 8.4)

Knapsack Problem by DP

Given n items of

integer weights: $w_1 \quad w_2 \quad \underline{w_i} \ldots \quad w_n$

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

$F[i,J]$: the first i items that fit the knapsack capacity of J.
max profit.

(i) If ith item does not fit into the knapsack.

$$\boxed{F[i,J] = F[i-1, J] \qquad \boxed{\text{if } J < w_i}}$$

Otherwise, $\boxed{\text{if } J \geq w_i}$

(a) If we use item i

$$F[i,J] = F[i-1, J-w_i] + V_i$$

(b) If we don't use item i

$$F[i,J] = F[i-1, J]$$

$$F[i,J] = \max\{F[i-1, J], \; F[i-1, J-w_i] + V_i\}$$

max

Knapsack Problem by DP

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

Let $F(i,j)$ be the value of an optimal solution to this instance, i.e., the value of the most valuable subset of the first i items that fit into the knapsack of capacity j.

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

Initial conditions:

$$F(0,\ j) = 0 \ \text{ for } j \geq 0 \quad \text{and} \quad F(i,\ 0) = 0 \ \text{ for } i \geq 0.$$

		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

Knapsack Problem by DP (example)

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

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

capacity $W = 5$.

$$F(0, j) = 0 \text{ for } j \geq 0 \quad \text{and} \quad F(i, 0) = 0 \text{ for } i \geq 0.$$

F[i,j]

capacity j

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

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

capacity $W = 5$.

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

$F[1,1] = F[0,1] = 0$ $\qquad j < w_1$

$F[1,2] = \max\{\underbrace{F[0,2]}_{0}, \underbrace{v_1}_{12} + F[0,0]\}$ $\qquad j=2 \quad w_1 = 2 \qquad j - w_1 = 0$

$= 12$

$F[1,3] = \max\{F[0,3], \underbrace{v_1}_{12} + F[0,1]\}$ $\qquad j=3 \quad w_1 = 2 \quad j - w_1 = 1$

$= 12$

$F[3,1] = F[2,1]$ $\qquad j = 1 \quad w_3 = 3$

$F[3,2] = F[2,2]$ $\qquad j = 2 \quad w_3 = 3$

$F[3,3] = \max\{\underbrace{F[2,3]}_{22}, \underbrace{v_3}_{20} + F[2,0]\}$ $\qquad j = 3 \quad w_3 = 3$

$= 22$

n item, W capacity
time complexity
$\Theta(nW)$

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**

0 W

max = 37 profit
item 1
item 2
item 4

max profit.

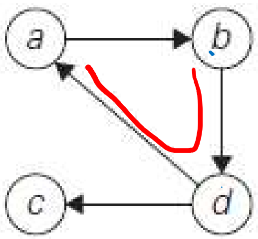
$F(4,5) > F(3,5)$, Item 4 has to be included
$5-2=3$ remaining capacity.
$F(3,3) = F(2,3)$, item 3 not need to be in optimal solution
$F(2,3) > F(1,3)$, Item 2 has to be included.
$3-1=2$ remaining capacity,
$F(1,2) > F(0,2)$ item 1 has to be included.

Warshall’s and Floyd’s Algorithms

- Warshall’s algorithm for computing the transitive closure of a directed graph
- Floyd’s algorithm for the all-pairs shortest-paths problem.
- These algorithms are based on essentially the same idea:
  - exploit a relationship between a problem and its simpler rather than smaller version

Warshall's Algorithm: Transitive Closure

DEFINITION The ***transitive closure*** of a directed graph with n vertices can be defined as the $n \times n$ boolean matrix $T = \{t_{ij}\}$, in which the element in the ith row and the jth column is 1 if there exists a nontrivial path (i.e., directed path of a positive length) from the ith vertex to the jth vertex; otherwise, t_{ij} is 0.



(a)

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

(b)

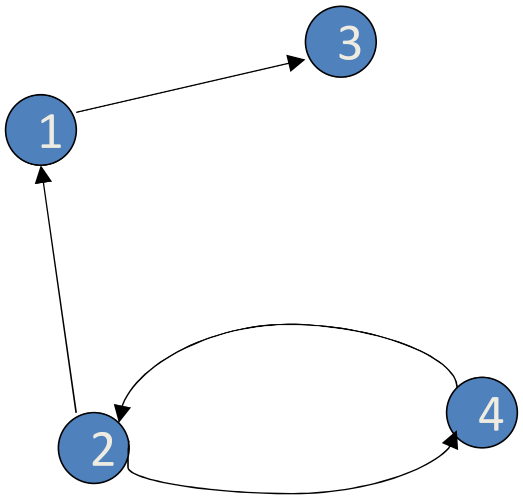
$$T = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

(c)

(a) Digraph. (b) Its adjacency matrix. (c) Its transitive closure.

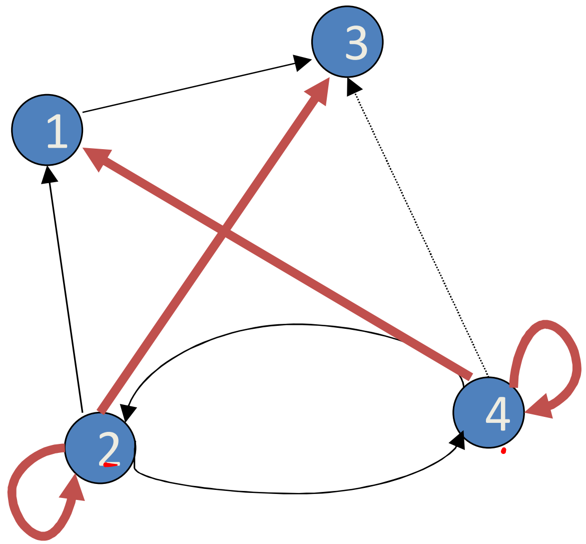
Warshall's Algorithm: Transitive Closure

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



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

adjacency matrix



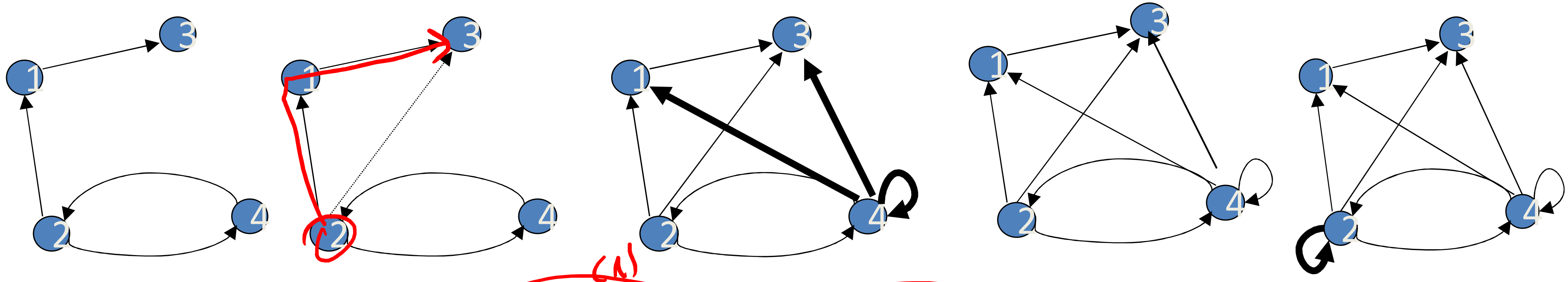
$$\begin{matrix} 0 & 0 & 1 & 0 \\ 1 & \mathbf{1} & \mathbf{1} & 1 \\ 0 & 0 & 0 & 0 \\ \mathbf{1} & 1 & \mathbf{1} & \mathbf{1} \end{matrix}$$

Warshall's Algorithm: Transitive Closure

- 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 $r_{ij}^{(k)} = 1$ iff there is nontrivial path from i to j with only first k vertices allowed as intermediate
- Note that $R^{(0)} = A$(adjacency matrix), $R^{(n)} = T$ (transitive closure)

in $R^{(1)}$ only node 1 can be used as intermediate node.

Warshall's Algorithm: Transitive Closure



(1)

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

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

$R^{(2)}$
$$\begin{array}{c} \\ \\ \\ 4 \end{array} \begin{matrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ \mathbf{1} & 1 & \mathbf{1} & \mathbf{1} \end{matrix}$$

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

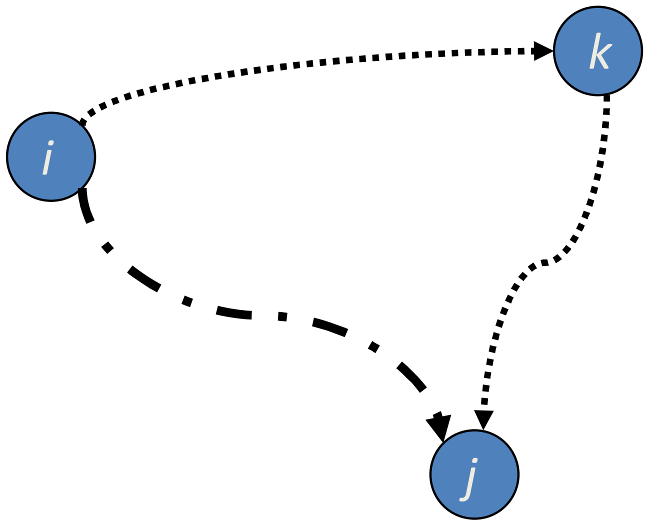
$R^{(4)}$
$$\begin{matrix} 0 & 0 & 1 & 0 \\ 1 & \mathbf{1} & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{matrix}$$

adjacency 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,…,k allowed as intermediate.

We have just proved is that if $r_{ij}^{(k)} = 1$, then either $r_{ij}^{(k-1)} = 1$ or both $r_{ik}^{(k-1)} = 1$ and $r_{kj}^{(k-1)} = 1$



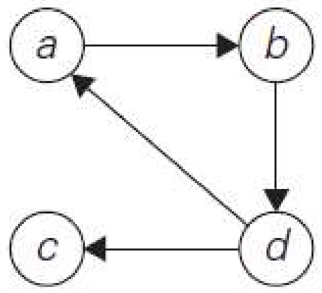
Recurrence relating elements $R^{(k)}$ to elements of $R^{(k-1)}$ is:

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

Warshall's Algorithm (matrix generation)

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

- If an element r_{ij} is 1 in $R^{(k-1)}$, it remains 1 in $R^{(k)}$.
- If an element r_{ij} 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 $R^{(k-1)}$.



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

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 $R^{(3)}$.

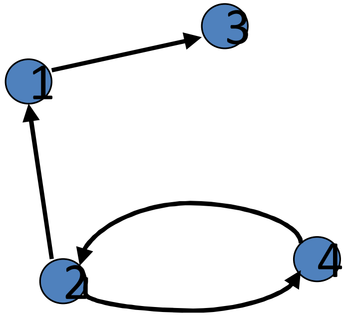
$$R^{(3)} = \begin{array}{cc} & \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \end{array} & \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

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).

Warshall's Algorithm (example)



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

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

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

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

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

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**
    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 $R^{(n)}$

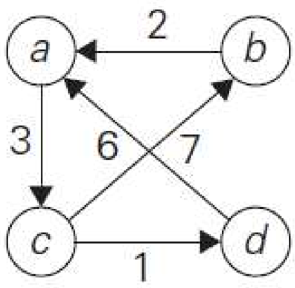
Time efficiency: $\Theta(n^3)$

Space efficiency: Matrices can be written over their predecessors

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 $D^{(0)}, \dots, D^{(n)}$ using increasing subsets of the vertices allowed as intermediate



(a)

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

(b)

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

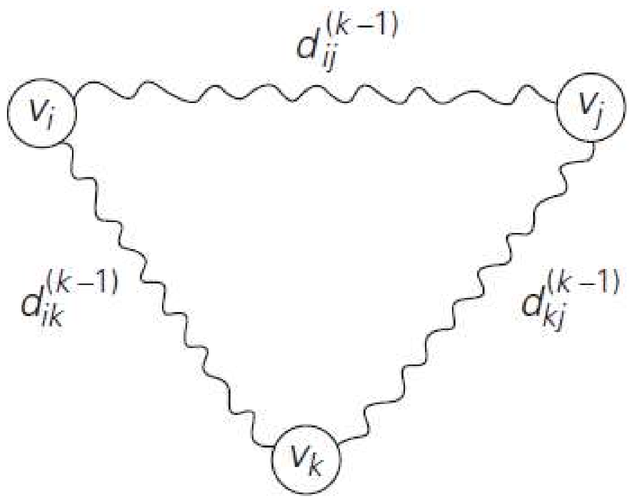
(c)

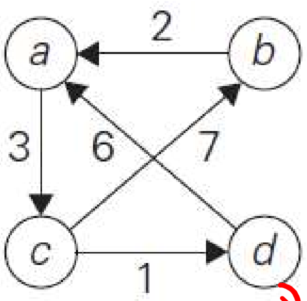
(a) Digraph. (b) Its weight matrix. (c) Its distance matrix.

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,…,k as intermediate

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





min{∞,5}

min{∞,9}

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

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

{9,∞}

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

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*).

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

4

,6?

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*).

{7,9}

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

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*).

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

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*).

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
    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: $\Theta(n^3)$

Space efficiency: Matrices can be written over their predecessors

Note: Shortest paths themselves can be found, too (Problem 10)