Dynamic Programming … Cont'd

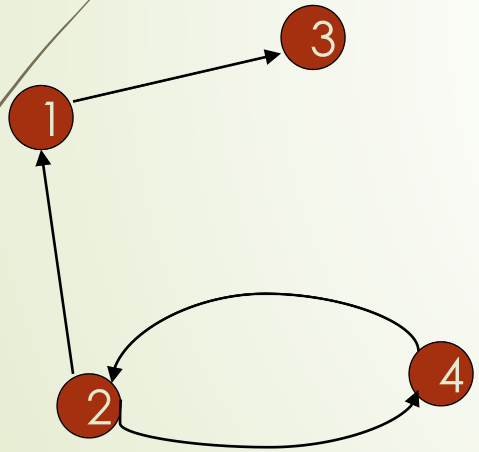
1. Warshall's Algorithm - for computing the *transitive closure of a graph* – also known as the **Reachability problem**

Warshall's algorithm for computing the *transitive closure of a graph*

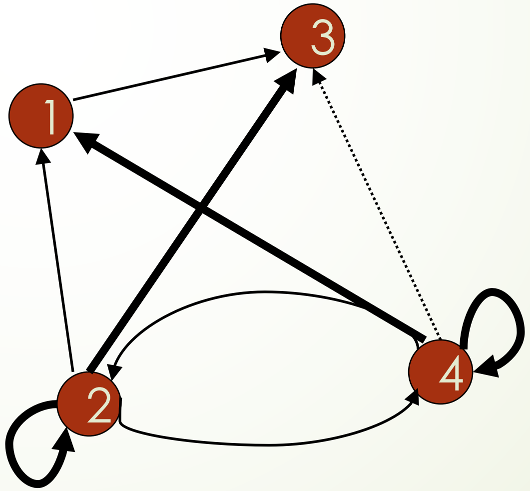
Given a directed graph **G**, determine if a vertex ***j*** is reachable from another vertex ***i*** for all vertex pairs ***(i, j)*** in **G.** Reachable means that there ***is a path from vertex i to j***. The reachability matrix, $\boldsymbol{R^{(n)}}$, is called the **transitive closure of G**

Warshall's Algorithm: Transitive Closure

- Computes the transitive closure of a relation (also known as the Reachability problem). **Find out all nodes reachable from every node to every other node**
- Alternatively: existence of all nontrivial paths in a digraph
- Example of transitive closure:



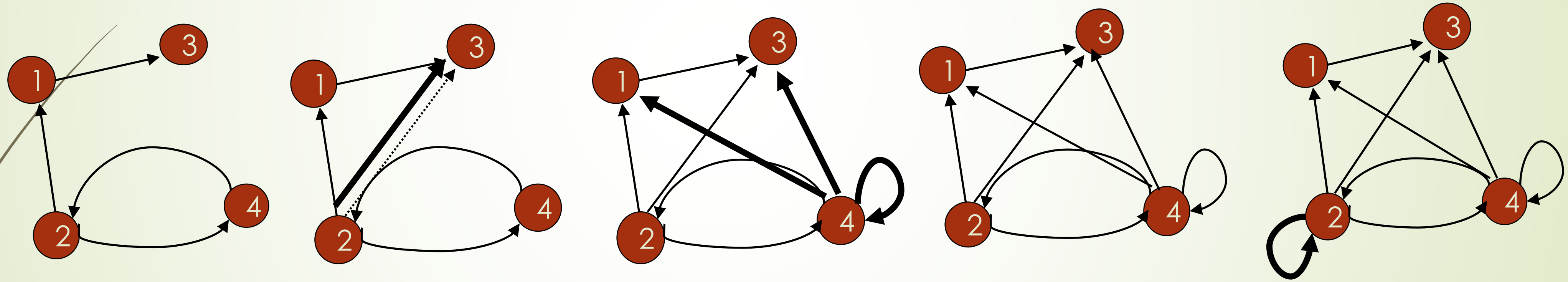
$$\begin{matrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{matrix}$$



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

Warshall's Algorithm

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 first k vertices allowed as intermediate
Note that $R^{(0)} = A$ (adjacency matrix), $R^{(n)} = T$ (transitive closure)

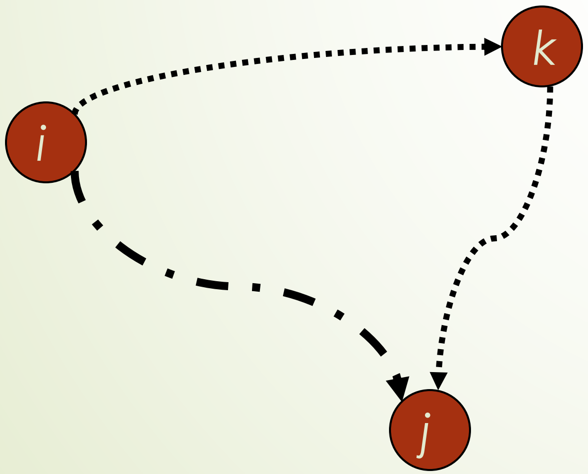


$$R^{(0)} \quad \begin{matrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{matrix} \qquad R^{(1)} \quad \begin{matrix} 0 & 0 & 1 & 0 \\ 1 & 0 & \mathbf{1} & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{matrix} \qquad R^{(2)} \quad \begin{matrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\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

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,…,k allowed as intermediate

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



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 $R^{(k-1)}$:

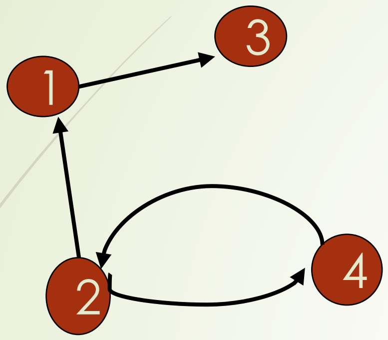
$$\begin{matrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{matrix}$$

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 $R^{(k-1)}$

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

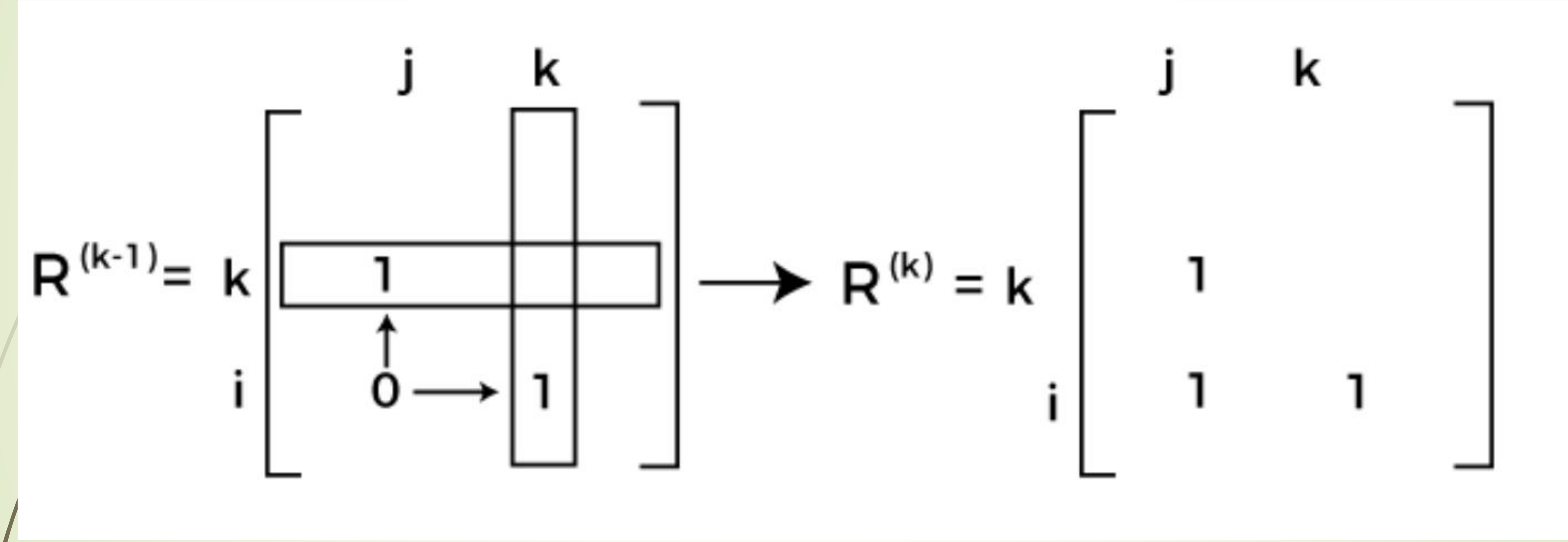
Warshall's Algorithm (example)



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

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

Rule for calculating $R^{(k)}$ from $R^{(k-1)}$ using Warshall's Algorithm



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, hence no extra space is required

2. Floyd’s algorithm for all-pairs shortest paths

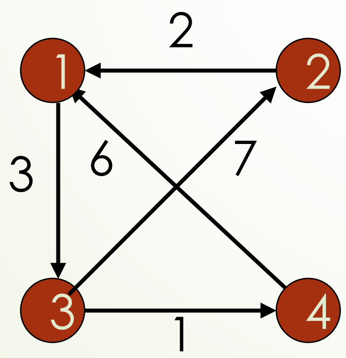
Using Dynamic Programming

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

Example:



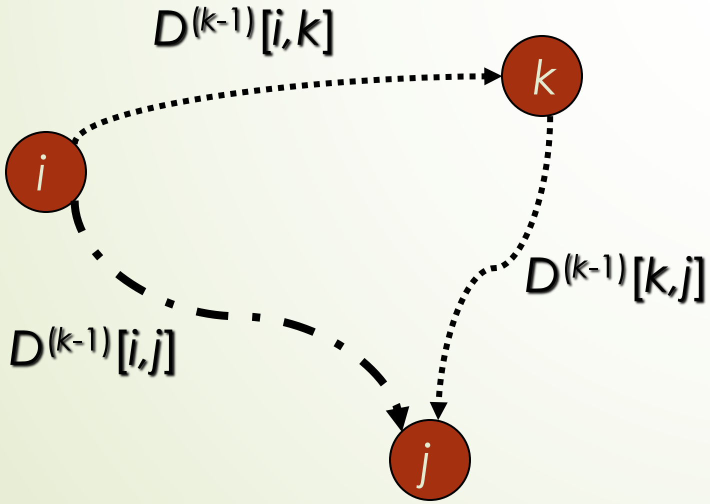
Original Weight matrix

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

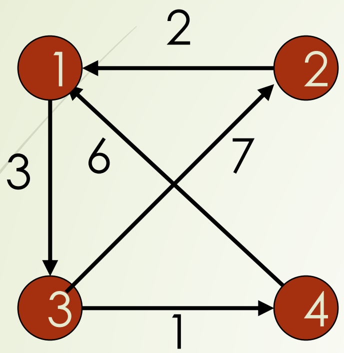
Floyd's Algorithm (matrix generation)

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

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



Floyd's Algorithm (example)



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

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

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

Floyd's Algorithm (pseudocode and analysis)

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

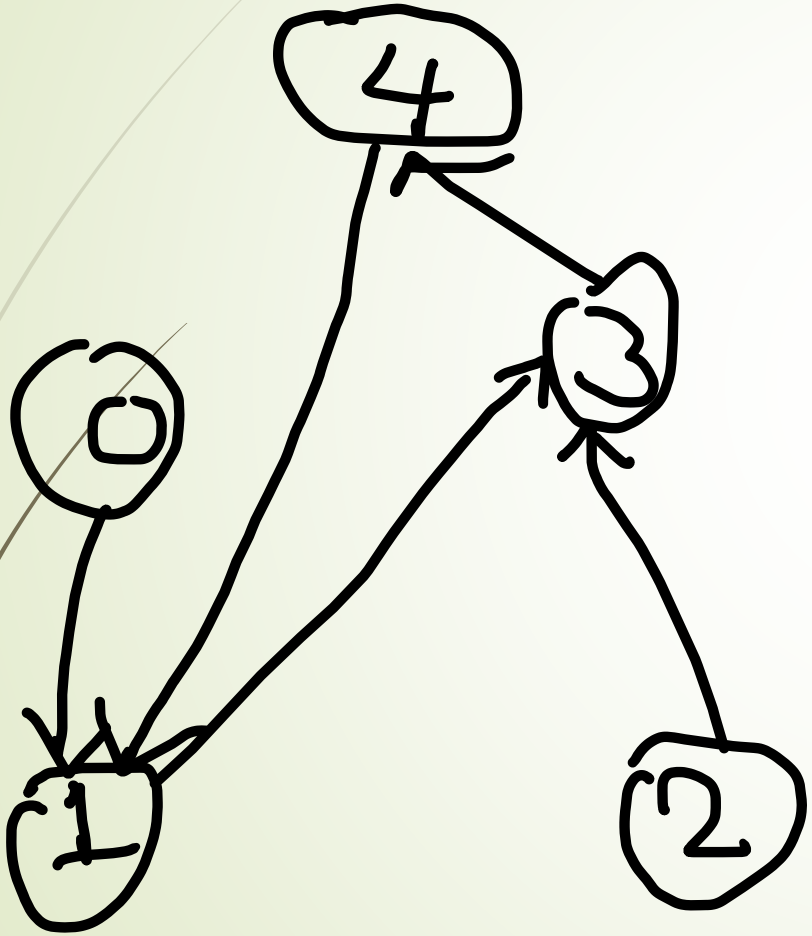
```
    //Implements Floyd's algorithm for the all-pairs shortest-paths problem
    //Input: The weight matrix W of a graph with no negative-length cycle
    //Output: The distance matrix of the shortest paths' lengths
    D ← W //is not necessary if W can be overwritten
    for k ← 1 to n do
        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, hence none

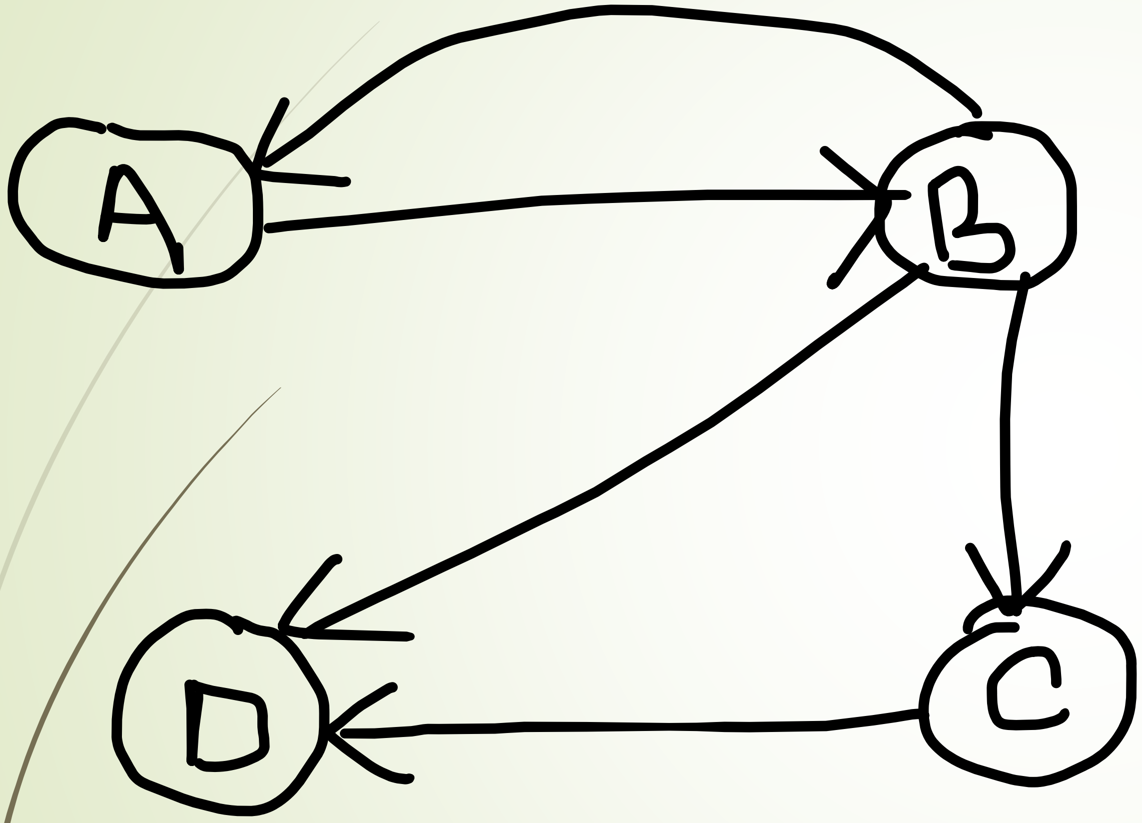
Note: Shortest paths themselves can be found, too (Problem 10 in Section 8.4)

In-class exercises … 1



5. Using dynamic programming, Find the transitive close ($A^{(5)}$ for the graph shown on the left.

In-class exercises … 2



4. Consider the graph shown here on the left. Making use of the Warshall's Algorithm, find the Transitive Closure for the said graph.

In-class exercises - 3

Solve the all-pairs shortest-path problem for the digraph with the weight matrix

$$\begin{bmatrix} 0 & 2 & \infty & 1 & 8 \\ 6 & 0 & 3 & 2 & \infty \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \infty & \infty & \infty & 0 \end{bmatrix}$$

Dynamic Programming … Cont'd

3. Knapsack problem

Using Dynamic Programming

3. Knapsack Problem by Dynamic Programming

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

Knapsack Problem by DP

Given n items of

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

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

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 ($i \leq n$) and capacity j ($j \leq W$).

Let $V[i,j]$ be optimal value of such instance. The i^{th} item may or may not be part of the optimal solution

- If item i is part of the optimal solution, then the value of the optimal solution is $\boldsymbol{v_i + V[i-1, j-w_i]}$
- If item i is not part of the optimal solution, then the value of the optimal solution is $\boldsymbol{V[i-1, j]}$

Knapsack Problem by DP

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

Let $V[i,j]$ be optimal value of such instance. The i^{th} item may or may not be part of the optimal solution

- If item i *is* part of the optimal solution, then the value of the optimal solution is $\mathbf{v_i + V[i\text{-}1,j\text{-}w_i]}$
- If item i is not part of the optimal solution, then the value of the optimal solution is $\mathbf{V[i\text{-}1,j]}$

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

Initial conditions: $V[0,j] = 0$ and $V[i,0] = 0$, we can also initialize the remaining cells to -1, to helps us know when to call a recursive function. So we will use **Memoization + Recursion**

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

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

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

capacity *j*

item *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						?

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

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

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

capacity *j*

item ***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					

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

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

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

capacity *j*

item *i*		0	1	2	3	4	5
	0	0	0	0	0	0	0
$w_1 = 2, v_1 = 12$	1	0	**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					

When i = 1, and j = 1, $j-w_i = 1 - 2 < 0$

So V[1,1] = V[0,1]

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

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

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

capacity *j*

item *i*		0	1	2	3	4	5
	0	<u>0</u>	0	<u>0</u>	0	0	0
$w_1 = 2, v_1 = 12$	1	0	0	**12**			
$w_2 = 1, v_2 = 10$	2	0					
$w_3 = 3, v_3 = 20$	3	0					
$w_4 = 2, v_4 = 15$	4	0					

When i = 1, and j = 2, $j-w_i = 2 - 2 = 0$

So $V[1,2] = \max(V[0,2], 12+V[0,0])$

$= \max(0, 12+0)$

$= 12$

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

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

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

capacity *j*

item *i*		0	1	2	3	4	5
	0	0	<u>0</u>	0	<u>0</u>	0	0
$w_1 = 2, v_1 = 12$	1	0	0	12	**12**		
$w_2 = 1, v_2 = 10$	2	0					
$w_3 = 3, v_3 = 20$	3	0					
$w_4 = 2, v_4 = 15$	4	0					

When i = 1, and j = 3, $j\text{-}w_i = 3 - 2 = 1 \geq 0$

So V[1,3] = max(V[0,3], 12+V[0,1])

= max(0, 12+0)

= 12

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

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

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

capacity *j*

item *i*		0	1	2	3	4	5
	0	0	0	<u>0</u>	0	<u>0</u>	0
$w_1 = 2, v_1 = 12$	1	0	0	12	12	**12**	
$w_2 = 1, v_2 = 10$	2	0					
$w_3 = 3, v_3 = 20$	3	0					
$w_4 = 2, v_4 = 15$	4	0					

When i = 1, and j = 4, $j-w_i = 4 - 2 = 2 \geq 0$

So V[1,4] = max(V[0,4], 12+V[0,2])

= max(0, 12+0)

= 12

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

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

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

capacity *j*

item *i*		0	1	2	3	4	5
	0	0	0	0	<u>0</u>	0	<u>0</u>
$w_1 = 2, v_1 = 12$	1	0	0	12	12	12	**12**
$w_2 = 1, v_2 = 10$	2	0					
$w_3 = 3, v_3 = 20$	3	0					
$w_4 = 2, v_4 = 15$	4	0					

When i = 1, and j = 5, $j\text{-}w_i = 5 - 2 = 3 \geq 0$

So V[1,5] = max(V[0,5], 12+V[0,3])

= max(0, 12+0)

= 12

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

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

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

capacity *j*

item *i*		0	1	2	3	4	5
	0	0	0	0	0	0	0
$w_1 = 2, v_1 = 12$	1	<u>0</u>	<u>0</u>	12	12	12	12
$w_2 = 1, v_2 = 10$	2	0	**10**				
$w_3 = 3, v_3 = 20$	3	0					
$w_4 = 2, v_4 = 15$	4	0					

When i = 2, and j = 1, $j\text{-}w_i = 1 - 1 = 0$

So $V[2,1] = \max(V[1,1], 10+V[1,0])$

$= \max(0, 10+0)$

$= 10$

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

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

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

capacity *j*

item *i*		0	1	2	3	4	5
	0	0	0	0	0	0	0
$w_1 = 2, v_1 = 12$	1	0	<u>0</u>	<u>12</u>	12	12	12
$w_2 = 1, v_2 = 10$	2	0	10	**12**			
$w_3 = 3, v_3 = 20$	3	0					
$w_4 = 2, v_4 = 15$	4	0					

When i = 2, and j = 2, $j-w_i = 2 - 1 = 1$

So V[2,2] = max(V[1,2], 10+V[1,1])

= max(12, 10+0)

= 12

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

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

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

capacity *j*

item *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

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

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

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

capacity *j*

item *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**

Composition of solution:

Item 4 (value 15)
Item 2 (value 10)
Item 1 (value 12)

Knapsack Problem by DP with Memoization (example)

Example: Knapsack of capacity $W = 5$

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

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

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

capacity *j*

item *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**

Knapsack Problem using Memoization (DP)

```
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's 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 V[0..n, 0..W] whose entries are initialized with −1's except for
    //row 0 and column 0 initialized with 0's
    if V[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]))
        V[i, j] ← value
    return V[i, j]
```

Runtime of algorithm: $O(nW)$
Space requirements: $O(nW)$

Runtime to determine the composition of the optimal solution: $O(n)$

In-class exercises

2. A knapsack has a maximum capacity of 60. There are 4 items with weights {20, 30, 40, 70} and values {70, 80, 90, 200}. Using dynamic programming, determine the maximum value of the items that can be carried in the knapsack.

3. Consider 3 items with the following : item 1 has weight 5 and value 4, item 2 has weight 12 and value 10 and item 3 has weight 8 and value 5. The total capacity of the knapsack is 11. Using dynamic programming, find the ***maximum number of items*** that can fit into the knapsack as well as ***this maximum value***.