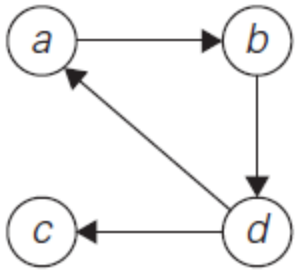
Dynamic Programming

Warshall's Algorithm

- Computes the transitive closure of a directed graph
- Existence of directed path of arbitrary lengths between nodes of a given graph.

- $T = \{t_{ij}\}$ → an $n \times n$ Boolean matrix
- $t_{ij} = 0$ → there exist no path from ith node to jth node
- $t_{ij} = 1$ → there exist some path from ith node to jth node



$A =$

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	0	1	0

$T =$

	a	b	c	d
a	1	1	1	1
b	1	1	1	1
c	0	0	0	0
d	1	1	1	1

Warshall's Algorithm

- Brute-Force Approach:
  - Calculate with use of depth-first-search or breadth-first-search
  - Start from *i*th node to find all nodes reachable from it
  - In row *i*, put 1 in place of nodes reachable
  - Repeat for all nodes
  - Not good!

Warshall's Algorithm

- Construct transitive closure through a series of Boolean $n \times n$ matrices

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

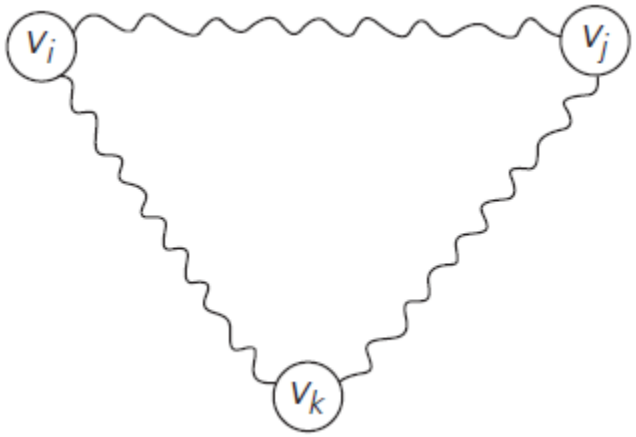
- $R^{(0)}$ → does not allow any intermediate node (adjacency matrix)
- $R^{(n)}$ → allows all nodes as intermediate (transitive closure)

- $r_{ij}^{(k)} = 1$ iff there is a path from i to j using intermediate nodes not higher than k

Warshall's Algorithm

- $R^{(k)}$ can be calculated using $R^{(k-1)}$
- $r_{ij}^{(k)} = r_{ij}^{(k-1)}$ intermediate list does not contain kth node
- $r_{ij}^{(k)} = r_{ik}^{(k-1)}\ and\ r_{kj}^{(k-1)}$ intermediate list contains kth node

- $r_{ij}^{(k-1)} = 1 \rightarrow r_{ij}^{(k)} = 1$
- $r_{ij}^{(k)} = r_{ij}^{(k-1)}\ or\ (r_{ik}^{(k-1)}\ and\ r_{kj}^{(k-1)})$



Warshall's Algorithm

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

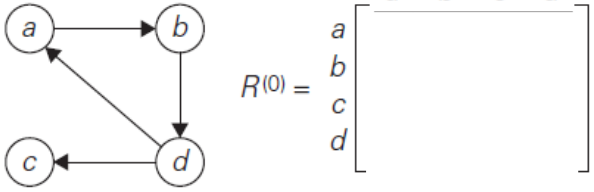
```
//Implements Warshall's algorithm for computing the transitive closure
//Input: The adjacency matrix A of a digraph with n vertices
//Output: The transitive closure of the digraph
R^(0) ← A
for k ← 1 to n do
    for i ← 1 to n do
        for j ← 1 to n do
            R^(k)[i, j] ← R^(k−1)[i, j] or (R^(k−1)[i, k] and R^(k−1)[k, j])
return R^(n)
```

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

Space efficiency:$\theta(n^2)$

Warshall's Algorithm

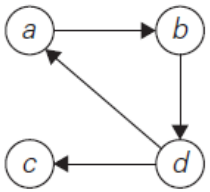
- $r_{ij}^{(k-1)} = 1 \rightarrow r_{ij}^{(k)} = 1$
- $r_{ij}^{(k)} = r_{ij}^{(k-1)}\ or\ (r_{ik}^{(k-1)}\ and\ r_{kj}^{(k-1)})$



$R^{(0)} =$

	a	b	c	d
a				
b				
c				
d				

Warshall's Algorithm



- $r_{ij}^{(k-1)} = 1 \rightarrow r_{ij}^{(k)} = 1$
- $r_{ij}^{(k)} = r_{ij}^{(k-1)}\ or\ (r_{ik}^{(k-1)}\ and\ r_{kj}^{(k-1)})$

$R^{(0)} =$

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	0	1	0

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 $R^{(1)}$.

$R^{(1)} =$

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	**1**	1	0

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 $R^{(2)}$.

$R^{(2)} =$

	a	b	c	d
a	0	1	0	**1**
b	0	0	0	1
c	0	0	0	0
d	1	1	1	**1**

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 $R^{(3)}$.

$R^{(3)} =$

	a	b	c	d
a	0	1	0	1
b	0	0	0	1
c	0	0	0	0
d	1	1	1	1

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 $R^{(4)}$.

$R^{(4)} =$

	a	b	c	d
a	**1**	1	**1**	1
b	**1**	**1**	**1**	1
c	0	0	0	0
d	1	1	1	1

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).

Floyd's Algorithm

- Find the length of the shortest paths from each node to all others
- Weighted, connected graph, with no cycle of a negative length
- Distant matrix → $n \times n$ matrix indicating the shortest paths

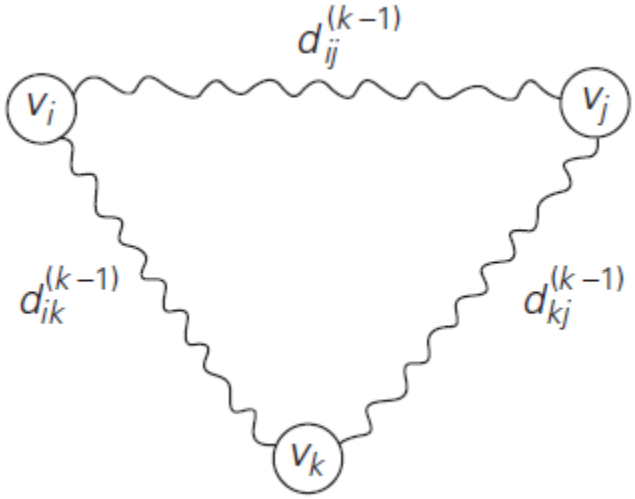
$$D^{(0)}, \ldots, D^{(k-1)}, D^{(k)}, \ldots, D^{(n)}$$

- $D^{(0)}$ → does not allow any intermediate nodes (weight matrix)
- $D^{(n)}$ → allow all nodes as intermediate nodes (distant matrix)
- $d_{ij}^{(k)}$ → length of the shortest path among all paths from node i to node j with use of possible intermediate nodes numbered not higher than k

Floyd's Algorithm

- $D^{(k)}$ can be calculated using $D^{(k-1)}$
- $d_{ij}^{(k)} = d_{ij}^{(k-1)}$ intermediate list does not contain kth node
- $d_{ij}^{(k)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$ intermediate list contains kth node

- $d_{ij}^{(0)} = w_{ij}$
- $d_{ij}^{(k)} = \min\{ d_{ij}^{(k-1)}, (d_{ik}^{(k-1)} + d_{kj}^{(k-1)})\}$



Floyd's Algorithm

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

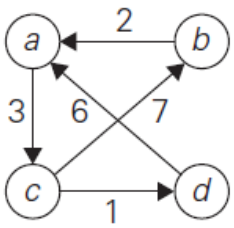
```
//Implements Floyd's algorithm for the all-pairs shortest-paths problem
//Input: The weight matrix W of a graph with no negative-length cycle
//Output: The distance matrix of the shortest paths' lengths
D ← W //is not necessary if W can be overwritten
for k ← 1 to n do
    for i ← 1 to n do
    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

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

Space efficiency: $\theta(n^2)$

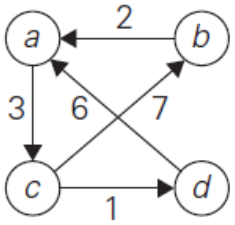
Floyd's Algorithm

- $d_{ij}^{(0)} = w_{ij}$
- $d_{ij}^{(k)} = \min\{ d_{ij}^{(k-1)}, (d_{ik}^{(k-1)} + d_{kj}^{(k-1)})\}$



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

Floyd's Algorithm



- $d_{ij}^{(0)} = w_{ij}$
- $d_{ij}^{(k)} = \min\{ d_{ij}^{(k-1)}, (d_{ik}^{(k-1)} + d_{kj}^{(k-1)})\}$

$D^{(0)} =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	∞	∞
c	∞	7	0	1
d	6	∞	∞	0

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

$D^{(1)} =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	**5**	∞
c	∞	7	0	1
d	6	∞	**9**	0

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*).

$D^{(2)} =$

	a	b	c	d
a	0	∞	3	∞
b	2	0	5	∞
c	**9**	7	0	1
d	6	∞	9	0

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*).

$D^{(3)} =$

	a	b	c	d
a	0	**10**	3	**4**
b	2	0	5	**6**
c	9	7	0	1
d	6	**16**	9	0

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*).

$D^{(4)} =$

	a	b	c	d
a	0	10	3	4
b	2	0	5	6
c	**7**	7	0	1
d	6	16	9	0

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*).

Discussion

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

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

Discussion

1. Applying Warshall's algorithm yields the following sequence of matrices (in which newly updated elements are shown in bold):

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

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

$$R^{(4)} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} = T$$

Discussion

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

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

Discussion

7. Applying Floyd's algorithm to the given weight matrix generates the following sequence of matrices:

$$D^{(0)} = \begin{bmatrix} 0 & 2 & \infty & 1 & 8 \\ 6 & 0 & 3 & 2 & \infty \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \infty & \infty & \infty & 0 \end{bmatrix} \qquad D^{(1)} = \begin{bmatrix} 0 & 2 & \infty & 1 & 8 \\ 6 & 0 & 3 & 2 & \mathbf{14} \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \mathbf{5} & \infty & \mathbf{4} & 0 \end{bmatrix}$$

$$D^{(2)} = \begin{bmatrix} 0 & 2 & \mathbf{5} & 1 & 8 \\ 6 & 0 & 3 & 2 & 14 \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & 5 & \mathbf{8} & 4 & 0 \end{bmatrix} \qquad D^{(3)} = \begin{bmatrix} 0 & 2 & 5 & 1 & 8 \\ 6 & 0 & 3 & 2 & 14 \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & 5 & 8 & 4 & 0 \end{bmatrix}$$

$$D^{(4)} = \begin{bmatrix} 0 & 2 & \mathbf{3} & 1 & \mathbf{4} \\ 6 & 0 & 3 & 2 & \mathbf{5} \\ \infty & \infty & 0 & 4 & \mathbf{7} \\ \infty & \infty & 2 & 0 & 3 \\ 3 & 5 & \mathbf{6} & 4 & 0 \end{bmatrix} \qquad D^{(5)} = \begin{bmatrix} 0 & 2 & 3 & 1 & 4 \\ 6 & 0 & 3 & 2 & 5 \\ \mathbf{10} & \mathbf{12} & 0 & 4 & 7 \\ \mathbf{6} & \mathbf{8} & 2 & 0 & 3 \\ 3 & 5 & 6 & 4 & 0 \end{bmatrix} = D$$

Discussion

11. *Jack Straws* In the game of Jack Straws, a number of plastic or wooden "straws" are dumped on the table and players try to remove them one-by-one without disturbing the other straws. Here, we are only concerned with if various pairs of straws are connected by a path of touching straws. Given a list of the endpoints for $n > 1$ straws (as if they were dumped on a large piece of graph paper), determine all the pairs of straws that are connected. Note that touching is connecting, but also two straws can be connected indirectly via other connected straws [1994 East-Central Regionals of the ACM International Collegiate Programming Contest].

Discussion

11. First, for each pair of the straws, determine whether the straws intersect. (Although this can be done in $n \log n$ time by a sophisticated algorithm, the quadratic brute-force algorithm would do because of the quadratic efficiency of the subsequent step; both geometric algorithms can be found, e.g., in R. Sedgewick's "Algorithms," Addison-Wesley, 1988.) Record the obtained information in a boolean n-by-n matrix, which must be symmetric. Then find the transitive closure of this matrix in n^2 time by DFS or BFS

Discussion

6. How would you construct an optimal binary search tree for a set of n keys if all the keys are equally likely to be searched for? What will be the average number of comparisons in a successful search in such a tree if $n = 2^k$?

Discussion

6. The binary search tree in question should have a maximal number of nodes on each of its levels except the last one. (For simplicity, we can put all the nodes of the last level in their leftmost positions possible to make it complete.) The keys of a given sorted list can be distributed among the nodes of the binary tree by performing its in-order traversal.

   Let p/n be the probability of searching for each key, where $0 \leq p \leq 1$. The complete binary tree with 2^k nodes will have 2^i nodes on level i for $i = 0, ..., k-1$ and one node on level k. Hence, the average number of comparisons in a successful search will be given by the following formula:

$$
\begin{aligned}
C(2^k) &= \sum_{i=0}^{k-1} (p/2^k)(i+1)2^i + (p/2^k)(k+1) \\
&= (p/2^k)\frac{1}{2}\sum_{i=0}^{k-1}(i+1)2^{i+1} + (p/2^k)(k+1) \\
&= (p/2^k)\frac{1}{2}\sum_{j=1}^{k} j2^j + (p/2^k)(k+1) \\
&= (p/2^k)\frac{1}{2}[(k-1)2^{k+1}+2] + (p/2^k)(k+1) \\
&= (p/2^k)[(k-1)2^k + k + 2].
\end{aligned}
$$

Discussion

10. ***Matrix chain multiplication*** Consider the problem of minimizing the total number of multiplications made in computing the product of n matrices

$$A_1 \cdot A_2 \cdot ... \cdot A_n$$

whose dimensions are d_0 by d_1, d_1 by d_2, ..., d_{n-1} by d_n, respectively. (Assume that all intermediate products of two matrices are computed by the brute-force (definition-based) algorithm.

a. Give an example of three matrices for which the number of multiplications in $(A_1 \cdot A_2) \cdot A_3$ and $A_1 \cdot (A_2 \cdot A_3)$ differ at least by a factor 1000.

b.▷ How many different ways are there to compute the chained product of n matrices?

c.► Design a dynamic programming algorithm for finding an optimal order of multiplying n matrices.

Discussion

10. a. Multiplying two matrices of dimensions α-by-β and β-by-γ by the definition-based algorithm requires $\alpha\beta\gamma$ multiplications. (There are $\alpha\gamma$ elements in the product, each requiring β multiplications to be computed.) If the dimensions of A_1, A_2, and A_3 are d_0-by-d_1, d_1-by-d_2, and d_2-by-d_3, respectively, then $(A_1 \cdot A_2) \cdot A_3$ will require

$$d_0 d_1 d_2 + d_0 d_2 d_3 = d_0 d_2 (d_1 + d_3)$$

multiplications, while $A_1 \cdot (A_2 \cdot A_3)$ will need

$$d_1 d_2 d_3 + d_0 d_1 d_3 = d_1 d_3 (d_0 + d_2)$$

multiplications. Here is a simple choice of specific values to make, say, the first of them be 1,000 times larger than the second:

$$d_0 = d_2 = 10^3, \quad d_1 = d_3 = 1.$$