بسم اللّه الرّحمن الرّحیم

دانشگاه صنعتی اصفهان ــ دانشکدهٔ مهندسی برق و کامپیوتر
(نیم‌سال تحصیلی ۴۰۰۱)

طراحی الگوریتم‌ها

حسین فلسفین

Catalan numbers

$$\begin{array}{rcl}(A_1)(A_2A_3\cdots A_n) & \to & t_1\times t_{n-1}\\ (A_1A_2)(A_3A_4\cdots A_n) & \to & t_2\times t_{n-2}\\ (A_1A_2A_3)(A_4A_5\cdots A_n) & \to & t_3\times t_{n-3}\\ \vdots & \vdots & \vdots\\ (A_1A_2\cdots A_{n-1})(A_n) & \to & t_{n-1}\times t_1\end{array}$$

$$t_n=\sum_{k=1}^{n-1}t_kt_{n-k}$$

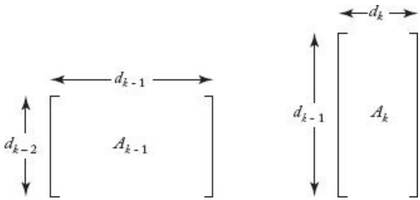
It can be proved by induction that: $t_n=\frac{1}{n}\binom{2(n-1)}{n-1}$

The n***th Catalan number:*** $C_n=\frac{1}{n+1}\binom{2n}{n}$

We have: $t_n=C_{n-1}$

It can be shown that: $C_n\approx\frac{4^n}{\sqrt{\pi n^3}}$ ☞ رشدش بد است!

Because we are multiplying the $(k-1)$st matrix, A_{k-1}, times the kth matrix, A_k, the number of columns in A_{k-1} must equal the number of rows in A_k. If we let d_0 be the number of rows in A_1 and d_k be the number of columns in A_k for $1 \leq k \leq n$, the dimension of A_k is $d_{k-1} \times d_k$.



For $1 \leq i \leq j \leq n$, let $M[i][j] =$ minimum number of multiplications needed to multiply A_i through A_j, if $i < j$. $M[i][i] = 0$.

Example:

$$\begin{array}{ccccccccccc} A_1 & \times & A_2 & \times & A_3 & \times & A_4 & \times & A_5 & \times & A_6 \\ 5\times 2 & & 2\times 3 & & 3\times 4 & & 4\times 6 & & 6\times 7 & & 7\times 8 \\ d_0 \quad d_1 & & d_1 \quad d_2 & & d_2 \quad d_3 & & d_3 \quad d_4 & & d_4 \quad d_5 & & d_5 \quad d_6 \end{array}$$

$$\begin{aligned} (A_4A_5)\,A_6 \text{ Number of multiplications } &= d_3\times d_4\times d_5 + d_3\times d_5\times d_6 \\ &= 4\times 6\times 7 + 4\times 7\times 8 = 392 \\ A_4\,(A_5A_6) \text{ Number of multiplications } &= d_4\times d_5\times d_6 + d_3\times d_4\times d_6 \\ &= 6\times 7\times 8 + 4\times 6\times 8 = 528 \end{aligned}$$

Therefore:

$$M[4][6] = \min(392, 528) = 392.$$

$$
\begin{array}{ccc}
(A_1)(A_2A_3\cdots A_n) & ☞ & M[1][1]+M[2][n]+d_0d_1d_n \\
(A_1A_2)(A_3A_4\cdots A_n) & ☞ & M[1][2]+M[3][n]+d_0d_2d_n \\
(A_1A_2A_3)(A_4A_5\cdots A_n) & ☞ & M[1][3]+M[4][n]+d_0d_3d_n \\
\vdots & \vdots & \vdots \\
(A_1A_2\cdots A_{n-2})(A_{n-1}A_n) & ☞ & M[1][n-2]+M[n-1][n]+d_0d_{n-2}d_n \\
(A_1A_2\cdots A_{n-1})(A_n) & ☞ & M[1][n-1]+M[n][n]+d_0d_{n-1}d_n
\end{array}
$$

$$
M[1][n]=\min_{1\leq k\leq n-1}(M[1][k]+M[k+1][n]+d_0d_kd_n)
$$

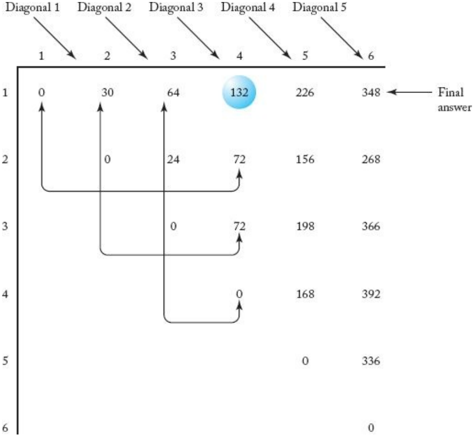
We can generalize this result: For $1 \le i \le j \le n$

$$M[i][j] = \min_{i \le k \le j-1} (M[i][k] + M[k+1][j] + d_{i-1} d_k d_j), \text{ if } i < j,$$

$$M[i][i] = 0.$$

$M[i][j]$ is calculated from all entries on the same row as $M[i][j]$ but to the left of it, along with all entries in the same column as $M[i][j]$ but beneath it. Using this property, we can compute the entries in M as follows: First we set all those entries in the main diagonal to 0; next we compute all those entries in the diagonal just above it, which we call diagonal 1; next we compute all those entries in diagonal 2; and so on. We continue in this manner until we compute the only entry in diagonal n, which is our final answer, $M[1][n]$.

$$
\begin{array}{ccccccccccc}
A_1 & \times & A_2 & \times & A_3 & \times & A_4 & \times & A_5 & \times & A_6 \\
5\times 2 & & 2\times 3 & & 3\times 4 & & 4\times 6 & & 6\times 7 & & 7\times 8 \\
d_0\quad d_1 & & d_1\quad d_2 & & d_2\quad d_3 & & d_3\quad d_4 & & d_4\quad d_5 & & d_5\quad d_6
\end{array}
$$



The main diagonal: $M[i][i] = 0$ ***for*** $1 \le i \le 6$.

The 1st diagonal:

$$\begin{aligned} M[1][2] &= \underset{1 \le k \le 1}{minimum}(M[1][k] + M[k+1][2] + d_0 d_k d_2) \\ &= M[1][1] + M[2][2] + d_0 d_1 d_2 \\ &= 0 + 0 + 5 \times 2 \times 3 = 30. \end{aligned}$$

The 2nd diagonal:

$$\begin{aligned} M[1][3] &= \underset{1 \le k \le 2}{minimum}(M[1][k] + M[k+1][3] + d_0 d_k d_3) \\ &= minimum(M[1][1] + M[2][3] + d_0 d_1 d_3, \\ &\qquad M[1][2] + M[3][3] + d_0 d_2 d_3) \\ &= minimum(0 + 24 + 5 \times 2 \times 4, 30 + 0 + 5 \times 3 \times 4) = 64. \end{aligned}$$

The 3rd diagonal:

$$\begin{aligned} M[1][4] &= \underset{1 \le k \le 3}{minimum}(M[1][k] + M[k+1][4] + d_0 d_k d_4) \\ &= minimum(M[1][1] + M[2][4] + d_0 d_1 d_4, \\ &\qquad M[1][2] + M[3][4] + d_0 d_2 d_4, \\ &\qquad M[1][3] + M[4][4] + d_0 d_3 d_4) \\ &= minimum(0 + 72 + 5 \times 2 \times 6, 30 + 72 + 5 \times 3 \times 6, \\ &\qquad 64 + 0 + 5 \times 4 \times 6) = 132. \end{aligned}$$

Inputs: *the number of matrices* n*, and an array of integers* d*, indexed from* 0 *to* n*, where* $d[i-1] \times d[i]$ *is the dimension of the* i*th matrix.*

Outputs: $minmult$*, the minimum number of elementary multiplications needed to multiply the* n *matrices; a two-dimensional array* P *from which the optimal order can be obtained.*

```
int minmult (int n,
             const int d[],
             index P[][])
{
  index i, j, k, diagonal;
  int M[1..n][1..n];

  for (i = 1; i <= n; i++)
     M[i][i] = 0;
  for (diagonal = 1; diagonal <= n - 1; diagonal++) // Diagonal-1 is
     for (i = 1; i <= n - diagonal; i++){           // just above the
        j = i + diagonal;                           // main diagonal
        M[i][j] =
          minimum   (M[i][k] + M[k + 1][j] + d[i - 1]*d[k]*d[j]);
        i <= k <= j-1

        P[i][j] = a value of k that gave the minimum;
     }
  return M[1][n];
}
```

Every-Case Time Complexity

Basic operation: We can consider the instructions executed for each value of k to be the basic operation. Included is a comparison to test for the minimum.
Input size: n, the number of matrices to be multiplied.

We have a loop within a loop within a loop.

Because $j = i + diagonal$, for given values of $diagonal$ and i, the number of passes through the k loop is

$$j - 1 - i + 1 = i + diagonal - 1 - i + 1 = diagonal.$$

For a given value of $diagonal$, the number of passes through the for-i loop is $n - diagonal$. Because diagonal goes from 1 to $n-1$, the total number of times the basic operation is done equals

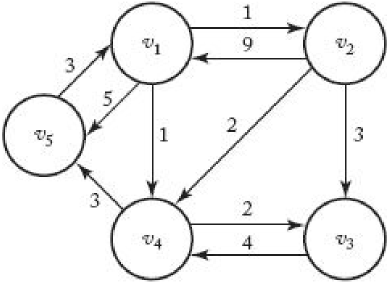
$$\sum_{diagonal=1}^{n-1} ((n - diagonal) \times diagonal)$$

It can be established that this expression equals

$$\frac{n(n-1)(n+1)}{6} \in \Theta(n^3).$$

(why?)

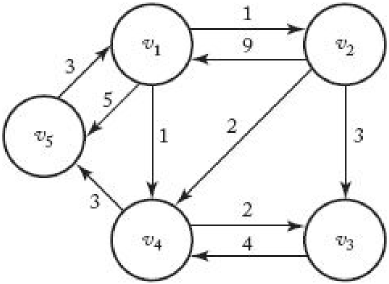
Let's informally review some graph theory.



☞ *Vertices - Edges (Arcs) - Directed Graph (Digraph) - Weights - Weighted Graph*

☞ *Path is a sequence of vertices such that there is an edge from each vertex to its successor.*

☞ *A path from a vertex to itself is called a cycle. If a graph contains a cycle, it is cyclic; otherwise, it is acyclic (Tree - Forest).*



☞ *A path is called* ***simple*** *if it never passes through the same vertex twice.*

☞ *The path $[v_1, v_2, v_3]$ in the figure is simple, but the path $[v_1, v_4, v_5, v_1, v_2]$ is not simple.*

☞ *The* ***length*** *of a path in a weighted graph is the sum of the weights on the path; in an unweighted graph it is simply the number of edges in the path.*

A problem that has many applications is finding the shortest paths from each vertex to all other vertices. Clearly, a shortest path must be a simple path.

Brute-force algorithm: For each vertex, determine the lengths of all the paths from that vertex to each other vertex, and compute the minimum of these lengths. This algorithm is worse than exponential-time.

For example, suppose there is an edge from every vertex to every other vertex. Then a subset of all the paths from one vertex to another vertex is the set of all those paths that start at the first vertex, end at the other vertex, and pass through all the other vertices. Because the second vertex on such a path can be any of $n-2$ vertices, the third vertex on such a path can be any of $n-3$ vertices, ..., and the second-to-last vertex on such a path can be only one vertex, the total number of paths from one vertex to another vertex that pass through all the other vertices is

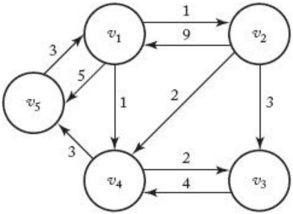
$$(n-2)(n-3)(n-4)\cdots 1 = (n-2)!$$

which is worse than exponential.

We represent a weighted graph containing n vertices by an array W where

$$W[i][j] = \begin{cases} \text{weight on edge} & \text{if there is an edge from } v_i \text{ to } v_j \\ \infty & \text{if there is no edge from } v_i \text{ to } v_j \\ 0 & \text{if } i = j. \end{cases}$$

Because vertex v_j is said to be adjacent to v_i if there is an edge from v_i to v_j, this array is called the adjacency matrix representation of the graph.



	1	2	3	4	5
1	0	1	∞	1	5
2	9	0	3	2	∞
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	∞	∞	∞	0

W

1	2	3	4	5
0	1	3	1	4
8	0	3	2	5
10	11	0	4	7
6	7	2	0	3
3	4	6	4	0

D

All-Pairs Shortest Paths

We consider the problem of finding shortest paths between all pairs of vertices in a graph. We are given a weighted, directed graph $G = (V, E)$ with a weight function $w : E \mapsto \mathbb{R}$ that maps edges to real-valued weights. We wish to find, for every pair of vertices $u, v \in V$, a shortest (least-weight) path from u to v, where the weight of a path is the sum of the weights of its constituent edges. We typically want the output in tabular form: the entry in u's row and v's column should be the weight of a shortest path from u to v (matrix D in the previous slide).

Lemma (Subpaths of shortest paths are shortest paths)

Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \mapsto \mathbb{R}$, let $p = \langle v_0, v_1, \ldots, v_k \rangle$ be a shortest path from vertex v_0 to vertex v_k and, for any i and j such that $0 \leq i \leq j \leq k$, let $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ be the subpath of p from vertex v_i to vertex v_j. Then, p_{ij} is a shortest path from v_i to v_j.

Proof. ***If we decompose path p into $v_0 \overset{p_{0i}}{\leadsto} v_i \overset{p_{ij}}{\leadsto} v_j \overset{p_{jk}}{\leadsto} v_k$ then we have that $w(p) = w(p_{0i}) + w(p_{ij}) + w(p_{jk})$. Now, assume that there is a path p'_{ij} from v_i to v_j with weight $w(p'_{ij}) < w(p_{ij})$. Then, $v_0 \overset{p_{0i}}{\leadsto} v_i \overset{p'_{ij}}{\leadsto} v_j \overset{p_{jk}}{\leadsto} v_k$ is a path from v_0 to v_k whose weight $w(p_{0i}) + w(p'_{ij}) + w(p_{jk})$ is less than $w(p)$, which contradicts the assumption that p is a shortest path from v_0 to v_k.*** $\square$

For the all-pairs shortest-paths problem on a graph $G = (V, E)$, it can be proven that that all subpaths of a shortest path are shortest paths. Suppose that we represent the graph by an adjacency matrix $W = (w_{ij})$. Consider a shortest path p from vertex i to vertex j , and suppose that p contains at most m edges. Assuming that there are no negative-weight cycles, m is finite. If $i = j$, then p has weight 0 and no edges. If vertices i and j are distinct, then we decompose path p into $i \overset{p'}{\leadsto} k \to j$, where path p' now contains at most $m - 1$ edges. p' is a shortest path from i to k, and so $\delta(i, j) = \delta(i, k) + w_{kj}$.

A recursive solution to the all-pairs shortest-paths problem

Now, let $l_{ij}^{(m)}$ be the minimum weight of any path from vertex i to vertex j that contains at most m edges. When $m = 0$, there is a shortest path from i to j with no edges if and only if $i = j$. Thus,

$$l_{ij}^{(0)} = \begin{cases} 0, & \text{if } i = j, \\ \infty, & \text{if } i \neq j. \end{cases}$$

For $m \geq 1$, we compute $l_{ij}^{(m)}$ as the minimum of $l_{ij}^{(m-1)}$ (the weight of a shortest path from i to j consisting of at most $m-1$ edges) and the minimum weight of any path from i to j consisting of at most m edges, obtained by looking at all possible predecessors k of j.

$$l_{ij}^{(m)} = \min\left(l_{ij}^{(m-1)}, \min_{1\le k\le n}\left\{l_{ik}^{(m-1)} + w_{kj}\right\}\right) = \min_{1\le k\le n}\left\{l_{ik}^{(m-1)} + w_{kj}\right\} \quad (*)$$

The latter equality follows since $w_{jj} = 0$ for all j.

What are the actual shortest-path weights $\delta(i, j)$? If the graph contains no negative-weight cycles, then for every pair of vertices i and j for which $\delta(i, j) < \infty$, there is a shortest path from i to j that is simple and thus contains at most $n-1$ edges. A path from vertex i to vertex j with more than $n-1$ edges cannot have lower weight than a shortest path from i to j. The actual shortest-path weights are therefore given by

$$\delta(i, j) = l_{ij}^{(n-1)} = l_{ij}^{(n)} = l_{ij}^{(n+1)} = \cdots .$$

Computing the shortest-path weights bottom up

Taking as our input the matrix $W = (w_{ij})$, we now compute a series of matrices $L^{(1)}, L^{(2)}, L^{(2)}, \ldots, L^{(n-1)}$, where for $m = 1, 2, \ldots, n-1$, we have $L^{(m)} = \left(l_{ij}^{(m)}\right)$. The final matrix $L^{(n-1)}$ contains the actual shortest-path weights. Observe that $l_{ij}^{(1)} = w_{ij}$ for all vertices $i, j \in V$, and so $L^{(1)} = W$.

The heart of the algorithm is the following procedure, which, given matrices $L^{(m-1)}$ and W, returns the matrix $L^{(m)}$. That is, it extends the shortest paths computed so far by one more edge.

EXTEND-SHORTEST-PATHS(L, W)

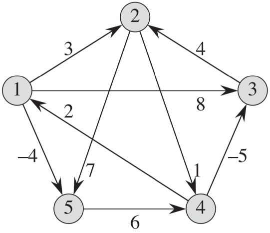
$n = L.rows$
let $L' = (l'_{ij})$ be a new $n \times n$ matrix
for $i = 1$ **to** n
    for $j = 1$ **to** n
        $l'_{ij} = \infty$
        for $k = 1$ **to** n
            $l'_{ij} = \min(l'_{ij}, l_{ik} + w_{kj})$
return L'

The procedure computes a matrix $L' = (l'_{ij})$, which it returns at the end. It does so by computing equation (*) for all i and j, using L for $L^{(m-1)}$ and L' for $L^{(m)}$. (It is written without the superscripts to make its input and output matrices independent of m.) Its running time is $\Theta(n^3)$ due to the three nested for loops.

The matrix $L^{(n-1)}$ contains the shortest-path weights. The following procedure computes the sequence $L^{(1)}, L^{(2)}, L^{(2)}, \ldots, L^{(n-1)}$ in $\Theta(n^4)$ time.

SLOW-ALL-PAIRS-SHORTEST-PATHS(W)

$n = W.rows$
$L^{(1)} = W$
for $m = 2$ **to** $n - 1$
    let $L^{(m)}$ be a new $n \times n$ matrix
    $L^{(m)} =$ EXTEND-SHORTEST-PATHS$(L^{(m-1)}, W)$
return $L^{(n-1)}$



$$L^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad L^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

$$L^{(3)} = \begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad L^{(4)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$