$A_1 \cdot A_2 \cdot A_3 \cdot A_4$ —— $k = 1, 2, 3$ —— $i, i+1, \cdots, j-1$

$m[1,4] = m[1,1] + m[2,4] + P_0 \cdot P_1 \cdot P_4$

$= m[1,2] + m[3,4] + P_0 \cdot P_2 \cdot P_4$

$= m[1,3] + m[4,4] + P_0 \cdot P_3 \cdot P_4$

(1,2), (2,3), (3,4).

(1,3), (2,4)

(1,4)

$l = 4$

$A_2 \cdot A_3$ —— $k = 2$ $\quad l = 2$

$m[2,3] = m[2,2] + m[3,3] + P_1 \cdot P_2 \cdot P_3$

$l = 2$. 时, $i = 1$, $j = 2$ $k = 1$ $\quad A_1 A_2 \cdots A_n$

$n = 2$ $\quad i = 2$, $j = 3$ $k = 2$ $\quad i = 1 - n - l + 1$

$\vdots$ $\quad j = i + n - 1$

$i = n-1$, $j = n$ $k = n-1$ $\quad k = i \sim j - 1$

$l = 3$. 时 $i = 1$, $j = 3$ $k = 1, 2$ $\quad A_1 A_2 \cdots A_n$

$i = 2$, $j = 4$ $k = 2, 3$ $\quad i = 1 - n - l + 1$

$i = 3$, $j = 5$ $k = 3, 4$ $\quad j = i + n - 1$

$\vdots$

$i = n-2$, $j = n$ $k = n-2$, $n-1$ $\quad k = i \sim j - 1$

$l = 4$. 时 $i = 1$, $j = 4$ $\quad A_1 A_2 \cdots A_n$

$i = 2$, $j = 5$ $\quad i = 1 - n - l + 1$

$i = 3$, $j = 6$ $\quad j = i + n - 1$

$\vdots$

$i = n-3$, $j = n$ $\quad k = i \sim j - 1$

$l = 2$. 时 $i = 1$, $j = 2$, $k = 1$

S

	1	2	3
1		1	2
2			2

$A_1 A_2 A_3$ $\quad i = 1$, $j = 3$

$S[1][3] = 2$.

$- \boxed{A_1 A_2} \ \& \ A_3$

$- \boxed{i = 1 \quad j = 2 \quad S[1][2] = 1}$

$(A_1 A_2) A_3$

$$m[i,j] = \begin{cases} p_{i-1} p_i p_j & \text{如果 } i = j \\ \min_{i \le k < j} \{ m[i,k] + m[k+1,j] + p_{i-1} p_k p_j \} & \text{如果 } i < j \end{cases} \tag{15.7}$$

$m[i,j]$ 的值给出了子问题最优解的代价，但它并未提供足够的信息来构造最优解。为此，我们用 $s[i,j]$ 保存 $A_i A_{i+1} \cdots A_j$ 最优括号化方案的分离点位置 k，即使得 $m[i,j] = m[i,k] + m[k+1,j] + p_{i-1} p_k p_j$ 成立的 k 值。

步骤 3：计算最优代价

现在，我们可以很容易地基于递归公式(15.7)写出一个递归算法，来计算 $A_1 A_2 \cdots A_n$ 相乘的最小代价 $m[1,n]$。像我们在钢条切割问题一节中所看到的，以及即将在 15.3 节中看到的那样，此递归算法是指数时间的，并不比枚举所有括号化方案的暴力搜索方法更好。

注意到，我们需要求解的不同子问题的数目是相对较少的：每对满足 $1 \le i \le j \le n$ 的 i 和 j 对应一个唯一的子问题，共有 $\binom{n}{2} + n = \Theta(n^2)$ 个。递归算法会在递归两用到的不同分支中多次遇到同一个子问题，这种子问题重叠的性质是应用动态规划的另一个标识(第一个标识是最优子结构)。

我们采用自底向上表格法代替基于公式(15.7)的递归算法来计算最优代价：我们将在 15.3 节中给出对应的带备忘的自顶向下方法。下面给出的过程 MATRIX-CHAIN-ORDER 实现了自底向上表格法，此过程假定矩阵 A_i 的规模为 $p_{i-1} \times p_i$($i = 1, 2, \cdots, n$)。它的输入是一个序列 $p = \langle p_0, p_1, \cdots, p_n \rangle$，其长度为 $p.length = n+1$。过程用一个辅助表 $m[1..n, 1..n]$ 来保存代价 $m[i,j]$。用另一个辅助表 $s[1..n-1, 2..n]$ 记录最优值 $m[i,j]$ 对应的分割点 k。我们就可以利用表 s 构造最优解。

为了实现自底向上方法，我们必须确定计算 $m[i,j]$ 时需要访问哪些其他表项。公式(15.7)暗示，$j-i+1$ 个矩阵链相乘的最优计算代价 $m[i,j]$ 只依赖于那些少于 $j-i+1$ 个矩阵链相乘的最优计算代价。也就是说，对 $k = i, i+1, \cdots, j-1$，矩阵 $A_i A_{i+1} \cdots A_k$ 是 $k-i+1 < j-i+1$ 个矩阵的乘积，矩阵 $A_k A_{k+1} \cdots A_j$ 是 $j-k = j-i+1-(k-i+1)$ 个矩阵的乘积。因此，算法应该按长度递增的顺序求解矩阵链括号化问题。并按对应长度顺序填写表 m。对矩阵链 $A_i A_{i+1} \cdots A_j$ 最优括号化的子问题，其长度为矩阵乘积的长度 $j-i+1$。

```
MATRIX-CHAIN-ORDER(p)
n = p.length - 1
let m[1..n, 1..n] and s[1..n-1, 2..n] be new tables
for i = 1 to n
    m[i,i] = 0
for l = 2 to n        // l is the chain length
    for i = 1 to n - l + 1
        j = i + l - 1
        m[i,j] = ∞
        for k = i to j - 1
            q = m[i,k] + m[k+1,j] + p_{i-1} p_k p_j
            if q < m[i,j]
                m[i,j] = q
                s[i,j] = k
return m and s
```

214 · 第四部分 高级设计和分析技术

算法首先在第 3—4 行对所有 $i = 1, 2, \cdots, n$ 计算 $m[i,i] = 0$(长度为 1 的链的最小计算代价)。接着在第 5—13 行 **for** 循环的第一个循环步中，利用递归公式(15.7)对所有 $i = 1, 2, \cdots, n-1$ 计算 $m[i, i+1]$(长度 $l = 2$ 的链的最小计算代价)。在第二个循环步中，算法对所有 $i = 1, 2, \cdots, n-2$ 计算 $m[i, i+2]$(长度 $l = 3$ 的链的最小计算代价)。依此类推。在每个循环步中，第 10—13 行计算代价 $m[i,j]$ 时仅依赖于已经计算出的表项 $m[i,k]$ 和 $m[k+1,j]$。

图 15-5 展示了对一个长度为 6 的矩阵链为对象算法的过程。由于我们定义 $m[i,j]$ 主对角线已经具有意义，因此表 m 只使用主对角线之上的部分。图中的表是经过旋转的，主对角线已经旋转到了水平方向。矩阵链的规模列在了图的下方，在这种布局中，我们可以看到子矩阵链 $A_i A_{i+1} \cdots A_j$ 相乘的代价 $m[i,j]$ 恰好位于矩子 A_i 的东北方向和西南方向的直线与矩子 A_j 的西北东南方向的直线的交点上。表中同一行中的表项都对应长度相同的矩阵链。MATRIX-CHAIN-ORDER 按自下向上，自左至右的顺序计算所有行。当计算表项 $m[i,j]$ 时，会用到乘积 $p_{i-1} p_k p_j$($k = i, i+1, \cdots, j-1$)，以及 $m[i,k]$ 西南方向和东南方向上的所有表项。

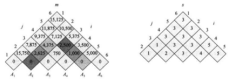

图 15-5 当 $n = 6$ 和矩阵规模如下面时，MATRIX-CHAIN-ORDER 计算出的 m 表和 s 表。

矩阵	A_1	A_2	A_3	A_4	A_5	A_6
规模	30×35	35×15	15×5	5×10	10×20	20×25

我们将两个表进行了旋转，使得主对角线方向变为水平方向。表 m 只使用主对角线和上三角部分，表 s 只使用上三角部分。6 个矩阵相乘所需的最小标量乘法运算次数为 $m[1,6] = 15\,125$。表中有些表项被标记了斜线阴影，相同的阴影表示过程在第 10 行中计算 $m[2,5]$ 时同时访问了这些表项。

$$\begin{aligned} m[2,5] &= \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 = 0 + 2\,500 + 35 \cdot 15 \cdot 20 &= 13\,000 \\ m[2,3] + m[4,5] + p_1 p_3 p_5 = 2\,625 + 1\,000 + 35 \cdot 5 \cdot 20 &= 7\,125 \\ m[2,4] + m[5,5] + p_1 p_4 p_5 = 4\,375 + 0 + 35 \cdot 10 \cdot 20 &= 11\,375 \end{cases} \\ &= 7\,125 \end{aligned}$$

简单分析 MATRIX-CHAIN-ORDER 的嵌套循环结构，可以看到算法的运行时间为 $O(n^3)$。循环嵌套的深度为三层，每层的循环变量(i, l 和 k)最多取 $n-1$ 个值，由习 15.2-5 要求证明此算法的运行时间实际上是 $\Omega(n^3)$。算法还需要 $\Theta(n^2)$ 的内存空间来保存表 m 和 s。因此，MATRIX-CHAIN-ORDER 比起穷举所有可能的括号化方案来寻找最优解的指数阶算法要高效得多。

$A_1 A_2$ —— m: 1 2 3; 1 0 $P_0P_1P_2$; 2 0 | S: 1 2; 1 | 2 | $l=2$ (1,2)

$A_1 A_2 \cdot A_3$ —— m: 1 2 3; 1 0 $P_0P_1P_2$ $m[1,2]$; 2 0 $P_1P_2P_3$; 3 0 | S: 1 2 3; 1 | 1 | k; 2 | 2 $l=3$ (1,2) (2,3)

$A_1 A_2 \cdot A_3 \cdot A_4$ ——

m: 1 2 3 4; 1 0; 2 0; 3 0; 4 0 $\quad S$: 1 2 3 4; 1; 2; 3; 4

$k \in [i, j)$

$A_1 A_2$ —— $P_0 \cdot P_1 \cdot P_1 \cdot P_2$ size —— $P_0 \cdot P_2$ $\quad m[1,2] = P_0 \cdot P_1 \cdot P_2$

$i = 1, j = 2$ $k = 1$ code —— $P_0 \cdot P_1 \cdot P_2$

$m[1,2] = \underset{0}{\cancel{m[1,1]}} + \cancel{m[2,2]} + P_0 \cdot P_1 \cdot P_2$ $\quad S[1,2] = 1$

$A_1 A_2 A_3$ —— $P_0 \cdot P_1 \cdot P_1 \cdot P_2 \cdot P_2 \cdot P_3$

$i = 1, j = 3$ $k = 1$. $\quad S[1,3]$

$m[1,3]$ —— $\cancel{m[1,1]} + \cancel{m[2,3]} + P_0 \cdot P_1 \cdot P_3$ (0; $P_1 \cdot P_2 \cdot P_3$)

$m[1,3]$ —— $\cancel{m[1,2]} + \cancel{m[3,3]} + P_0 \cdot P_2 \cdot P_3$ ($P_0 \cdot P_1 \cdot P_3$; 0) min

$A_2 \cdot A_3$ —— $\cancel{m[2,2]} + \cancel{m[3,3]} + P_1 \cdot P_2 \cdot P_3$ $\quad S[2,3] = 2$

$A_2 \cdot A_3 \cdot A_4$ —— $k = 2, 3$ $\quad l = 3$

$m[2,4] = \cancel{m[2,2]} + m[3,4] + P_1 \cdot P_2 \cdot P_4$

$= m[2,3] + \cancel{m[4,4]} + P_1 \cdot P_3 \cdot P_4$

m: 2 3 4; 2 0; 3 0; 4 0 $\quad (2,3), (3,4)$ $(2,4)$ $\quad l = 3$