

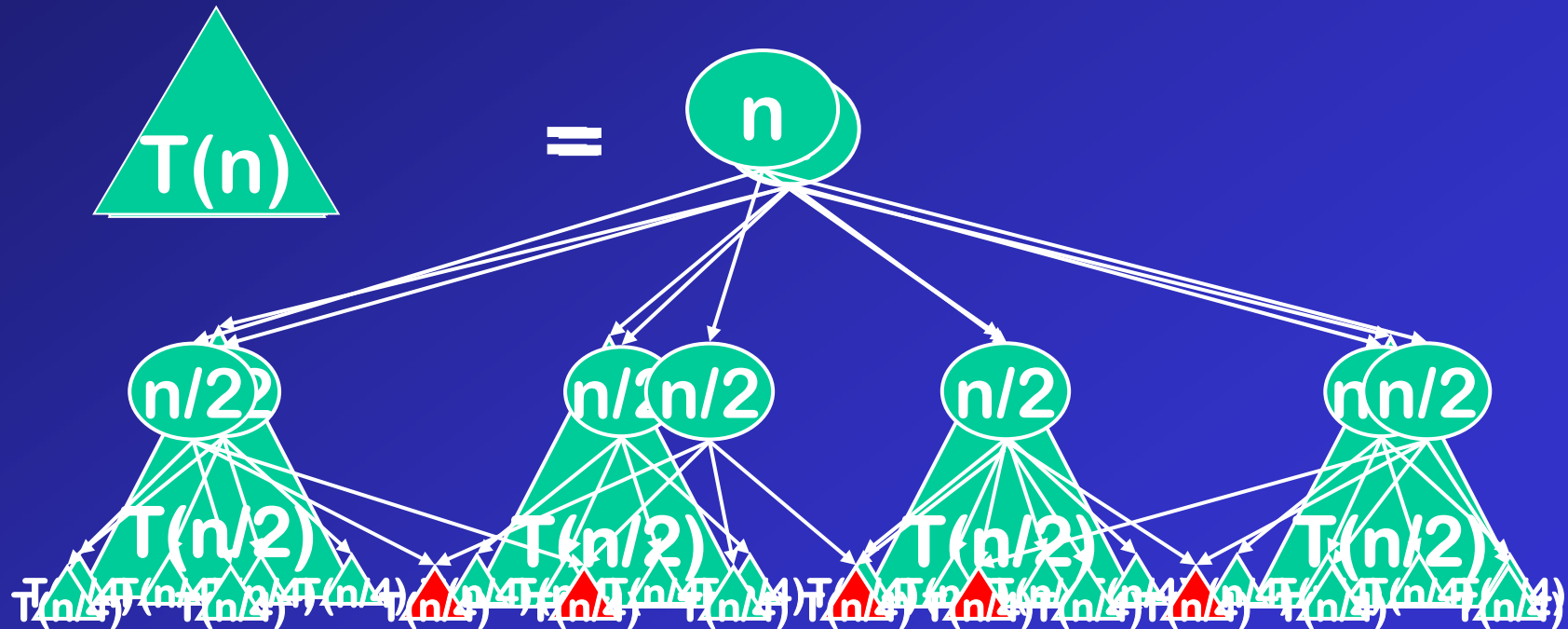
第三章 动态规划

➤ 本章主要知识点：

- 3.1 矩阵连乘问题
- 3.2 动态规划算法的基本要素
- 3.3 最长公共子序列问题
- 3.4 最大子段和
- 3.5 凸多边形的最优三角剖分
- 3.6 多边形游戏
- 3.7 图像压缩
- 3.8 电路布线
- 3.9 流水作业调度
- 3.10 0-1背包问题

引言

- 动态规划算法与分治法类似，其基本思想也是将待求解问题分解成若干个子问题。
- 但是经分解得到的子问题往往不是互相独立的。不同子问题的数目常常只有多项式量级。在用分治法求解时，有些子问题被重复计算了许多次。
- 如果能够保存已解决的子问题的答案，而在需要时再找出已求得的答案，就可以避免大量重复计算，从而得到多项式时间算法。



动态规划基本步骤

- 找出最优解的性质，并刻画其结构特征。
- 递归地定义最优值。
- 以自底向上的方式计算出最优值。
- 根据计算最优值时得到的信息，构造最优解。

3.1 矩阵连乘问题

给定 n 个矩阵 $\{A_1, A_2, \dots, A_n\}$ ，其中 A_i 与 A_{i+1} 是可乘的， $i=1, 2, \dots, n-1$ 。如何确定计算矩阵连乘积的计算次序，使得依此次序计算矩阵连乘积需要的数乘次数最少。

$$\begin{array}{ccc} 1 & 2 & 1 \\ 2 & 1 & 3 \end{array} \quad \begin{array}{cc} 1 & 1 \\ 2 & 3 \\ 1 & 1 \end{array} \quad \longrightarrow \quad \begin{array}{cc} 6 & 8 \\ 7 & 8 \end{array}$$

两个矩阵相乘所需做的数乘次数为 $l*n*m$

矩阵乘法满足结合律，故矩阵连乘可以有许许多多不同的计算顺序。

计算顺序由加括号的方式确定。

加括号的方式决定了矩阵连乘的计算量

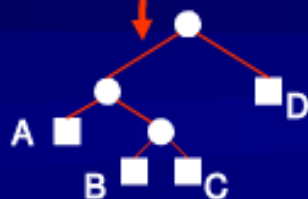
3.1 矩阵连乘问题

■ 例如

4个矩阵连乘积ABCD，设它们的维数分别为
A:50×10, B:10×40, C:40×30, D:30×5。

矩阵连乘的完全加括号形式以及相应的计算量

完全加括号形式	连乘积所需计算量
(A (B (C D)))	$40 \times 30 \times 5 + 10 \times 40 \times 5 + 50 \times 10 \times 5 = 10500$
(A ((B C) D))	$10 \times 40 \times 30 + 10 \times 30 \times 5 + 50 \times 10 \times 5 = 16000$
((A B) (C D))	$50 \times 10 \times 40 + 40 \times 30 \times 5 + 50 \times 40 \times 5 = 36000$
(((A B) C) D)	$50 \times 10 \times 40 + 50 \times 40 \times 30 + 50 \times 30 \times 5 = 87500$
((A (B C)) D)	$10 \times 40 \times 30 + 50 \times 10 \times 30 + 50 \times 30 \times 5 = 34500$



每种计算次序都可以用一个二叉树来表示

3.1 矩阵连乘问题

- 已确定计算次序的矩阵连乘积称为**完全加括号**
- 由于每种加括号方式都可以分解为两个子连乘积的加括号问题，因此**完全加括号**的矩阵连乘积可递归地定义为：
 - 矩阵连乘积**A**是完全加括号的，则**A**可表示为两个已完全加括号的矩阵连乘积**B**和**C**的乘积，即**A=(BC)**。
 - 其中单个矩阵视为已完全加括号的；

$$A = A_1 \times A_2 \times A_3 \times \dots \times A_n$$

$$\underset{\text{B}}{A} = (A_1 \times A_2 \times \dots \times A_k) \times (A_{k+1} \times A_{k+2} \times \dots \times A_n) \underset{\text{C}}$$

3.1 矩阵连乘问题---穷举搜索法

➤ 列举出所有可能的计算次序，并计算出每一种计算次序相应需要的数乘次数，从中找出一种数乘次数最少的计算次序。

➤ 算法复杂度分析：

- 对于n个矩阵的连乘积，设其不同的计算次序为 $P(n)$ 。
- 由于每种加括号方式都可以分解为两个子矩阵的加括号问题： $(A_1 \dots A_k)(A_{k+1} \dots A_n)$ 可以得到关于 $P(n)$ 的递推式如下：

$$P(n) = \begin{cases} 1 & n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & n > 1 \end{cases}$$

▪

$$\Rightarrow P(n) = C(n-1), \text{ 其中, } C(n) = \frac{1}{n+1} \binom{2n}{n} = \Omega(4^n / n^{3/2})$$

- 也就是说， $P(n)$ 是随 n 的增长成指数增长的。

3.1 矩阵连乘问题---分治法

■ 分治法

- 将矩阵连乘积 $\mathbf{A}_i\mathbf{A}_{i+1}\dots\mathbf{A}_j$ 简记为 $\mathbf{A}[i:j]$ 。
- 设 n 个矩阵的连乘积 $\mathbf{A}_1\mathbf{A}_2\dots\mathbf{A}_n$ 的计算次序在矩阵 \mathbf{A}_k 和 \mathbf{A}_{k+1} 之间将矩阵链断开, $1\leq k<n$, 则其相应的完全加括号方式为 $(\mathbf{A}_1\dots\mathbf{A}_k)(\mathbf{A}_{k+1}\dots\mathbf{A}_n)$ 。
- 完全加括号是一个递归定义, 可递归地计算 $\mathbf{A}[1:k]$ 和 $\mathbf{A}[k+1:n]$, 然后将计算结果相乘得到 $\mathbf{A}[1:n]$ 。

3.1 矩阵连乘问题---分治法

可以递归地定义 $m[i,j]$ 为:

$$m[i, j] = \begin{cases} 0 & i = j \\ \min_{i \leq k < j} \{m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j\} & i < j \end{cases}$$

前部分
的时间

后部分
的时间

合并时间

k 的位置只有 $j-i$ 种可能

3.1 矩阵连乘问题---分治法

```
int recurMatrixChain(int i, int j)
{
    if (i == j) return 0;
    int u = recurMatrixChain(i, i) + recurMatrixChain(i+1, j) + p[i-1]*p[i]*p[j];
    s[i][j] = i;
    for (int k = i+1; k < j; k++) {
        int t = recurMatrixChain(i, k) + recurMatrixChain(k+1, j) + p[i-1]*p[k]*p[j];
        if (t < u) {
            u = t;
            s[i][j] = k;
        }
    }
    return u;
}
```

3.1 矩阵连乘问题---分治法

- 该递归算法的的计算时间 **$T(n)$** 可递归定义如下:

$$T(n) \geq \begin{cases} O(1) & n = 1 \\ 1 + \sum_{k=1}^{n-1} (T(k) + T(n-k) + 1) & n > 1 \end{cases}$$

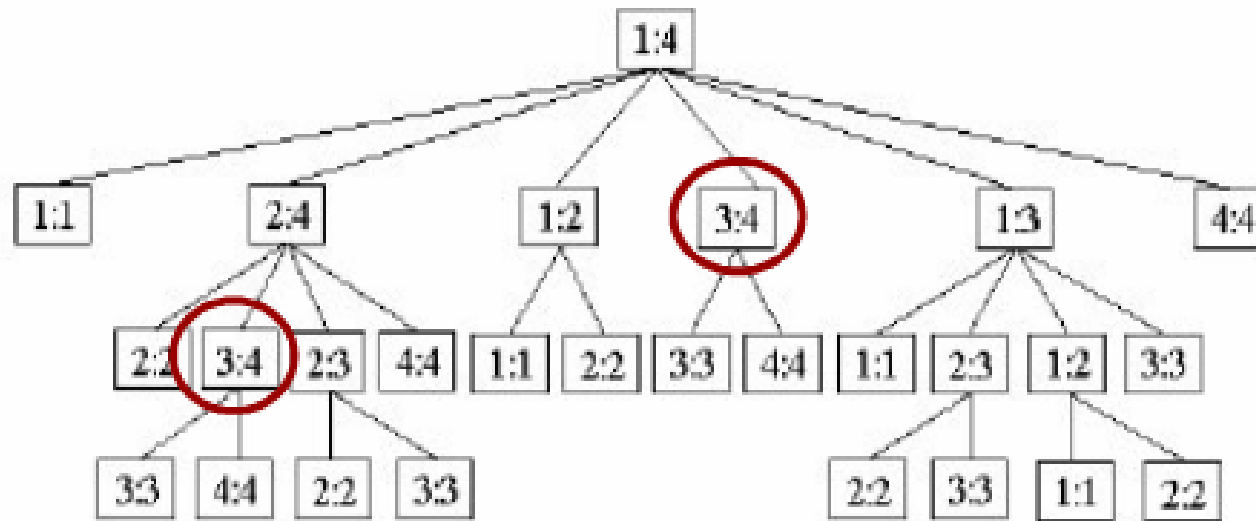
- 当 **$n > 1$** 时, $T(n) \geq 1 + (n-1) + \sum_{k=1}^{n-1} T(k) + \sum_{k=1}^{n-1} T(n-k) = n + 2 \sum_{k=1}^{n-1} T(k)$

$$T(n) \geq 2^{n-1} = \Omega(2^n)$$

- 该算法的计算时间 **$T(n)$** 有指数下界。
- 分治法是该问题的一个可行方法, 但不是是一个有效算法。

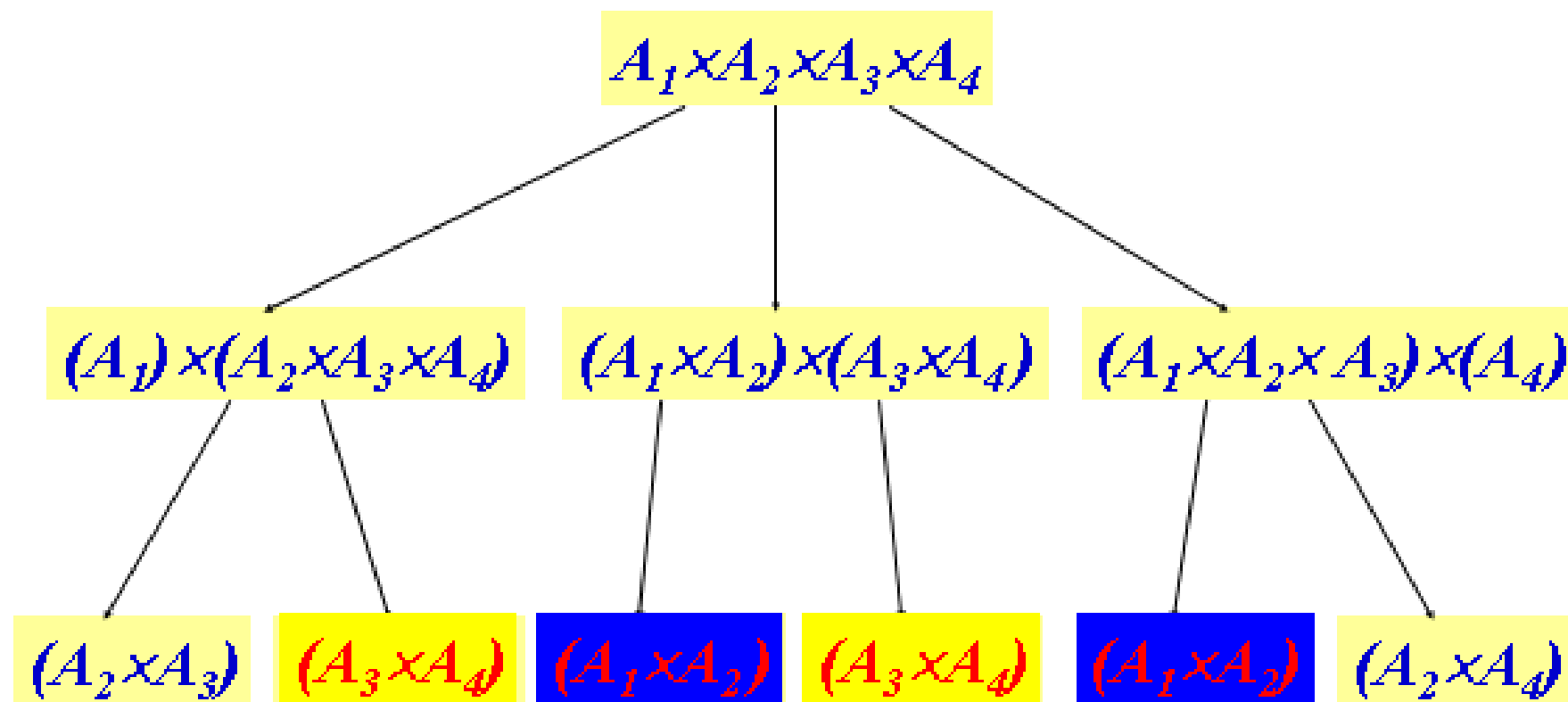
3.1 矩阵连乘问题---分治法

- 为何分治法的效率如此低下？
- **recurmatrixChain(1,4)**计算**A[1: 4]**的递归树。



- 从图中可看出，许多子问题被重复计算。
- 这是分治法效率低下的根本原因。

3.1 矩阵连乘问题---动态规划法



- 该问题可用分治思想解决，并存在大量冗余，可用动态规划求解。

3.1 矩阵连乘动态规划法——1.分析最优解的结构

- 下面我们考虑用动态规划求解。
- 预处理：
 - 将矩阵连乘积 $A_i A_{i+1} \dots A_j$ 简记为 $A[i:j]$ ，这里 $i \leq j$ 。
 - 考察计算 $A[i:j]$ 的最优计算次序。设这个计算次序在矩阵 A_k 和 A_{k+1} 之间将矩阵链断开， $i \leq k < j$ ，则其相应完全加括号方式为 $(A_i A_{i+1} \dots A_k) (A_{k+1} A_{k+2} \dots A_j)$ 。
 - 计算量： $A[i:k]$ 的计算量加上 $A[k+1:j]$ 的计算量，再加上 $A[i:k]$ 和 $A[k+1:j]$ 相乘的计算量。
- 分析最优解的结构
 - **特征**：计算 $A[i:j]$ 的最优次序所包含的计算矩阵子链 $A[i:k]$ 和 $A[k+1:j]$ 的次序也是最优的。
 - 矩阵连乘计算次序问题的最优解包含着其子问题的最优解。这种性质称为**最优子结构性质**。问题的最优子结构性质是该问题可用动态规划算法求解的显著特征。

3.1 矩阵连乘动态规划法—— 1.分析最优解的结构

1、划分阶段（子问题）并刻画

$$(A((BC)D)) \Rightarrow (A)((BC)(D)) \quad ((A(BC))D) \Rightarrow ((A)(BC))(D)$$

$$(((AB)C)D) \Rightarrow ((AB)(C))(D)$$

将原问题划分成两个子问题。如果原问题获得最优值。则子问题的应该也是最优的。

将矩阵连乘积 $A_1 A_2 \dots A_n$ 简记为 $A(1:n)$

设最优计算次序在矩阵

A_k 和 A_{k+1} 之间将矩阵链断开， $1 \leq k < n$ ，则其相应完全加括号方式为

$$(A_1 \dots A_k) (A_{k+1} A_{k+2} \dots A_n)$$

3.1 矩阵连乘动态规划法—— 2.建立递归关系

- 设计算 $A[i:j]$, $1 \leq i \leq j \leq n$, 所需要的最少数乘次数 $m[i,j]$, 则原问题的最优值为 $m[1,n]$ 。
- 当 $i=j$ 时, $A[i:j]=A_i$, 因此, $m[i,i]=0$, $i=1,2,\dots,n$ 。
- 当 $i < j$ 时, $m[i,j]=m[i,k]+m[k+1,j]+p_{i-1}p_kp_j$, 这里 A_i 的维数为 $p_{i-1} \times p_i$ 。
- 可以递归地定义 $m[i,j]$ 为:

$$m[i, j] = \begin{cases} 0 & i = j \\ \min_{i \leq k < j} \{m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j\} & i < j \end{cases}$$

- k 的位置只有 $j-i$ 种可能。

3.1 矩阵连乘动态规划法—— 3.计算最优值

- 对于 $1 \leq i \leq j \leq n$ 不同的有序对 (i, j) 对应于不同的子问题。
- 由此可见，在递归计算时，许多子问题被重复计算多次。这也是该问题可用动态规划算法求解的又一显著特征。
- 用动态规划算法解此问题，可依据其递归式以自底向上的方式进行计算。在计算过程中，保存已解决的子问题答案。每个子问题只计算一次，而在后面需要时只要简单查一下，从而避免大量的重复计算，最终得到多项式时间的算法。

3.1 矩阵连乘动态规划法——算法描述

➤ 算法描述:

```
void MatrixChain(int *p, int n, int **m, int **s)
{
    for (int i = 1; i <= n; i++) m[i][i] = 0;
    for (int r = 2; r <= n; r++)
        for (int i = 1; i <= n - r + 1; i++) {
            int j = i + r - 1;
            m[i][j] = m[i + 1][j] + p[i - 1] * p[i] * p[j];
            s[i][j] = i;
            for (int k = i + 1; k < j; k++) {
                int t = m[i][k] + m[k + 1][j] + p[i - 1] * p[k] * p[j];
                if (t < m[i][j]) { m[i][j] = t; s[i][j] = k; }
            }
        }
}
```

3.1 矩阵连乘动态规划法——实例分析

例 四个乘 50×10 10×40 40×30 30×5 的矩阵连乘。

$P[0]=50$ $P[1]=10$ $P[2]=40$ $P[3]=30$ $P[4]=5$

m

0			
	0		
		0	
			0

for (int i = 1; i <= n; i++) m[i][i] = 0;

3.1 矩阵连乘动态规划法——实例分析

例 四个乘 50×10 10×40 40×30 30×5 的矩阵连乘。

$P[0]=50$ $P[1]=10$ $P[2]=40$ $P[3]=30$ $P[4]=5$

m

0			
	0		
		0	6000
			0

• $i=n-1=3, j=4, k=3$

$$m[3,4] = m[3,3] + m[4,4] + p[2] \cdot p[3] \cdot p[4] \\ = 40 \cdot 30 \cdot 5 = 6000$$

$i=2, j=3, k=2$

$$m[2,3] = m[2,2] + m[3,3] + p[1] \cdot p[2] \cdot p[3] \\ = 10 \cdot 40 \cdot 30 = 12000 \quad k=2$$

0			
	0	12000	8000
		0	6000
			0

$i=2, j=4, k=2 \sim 3$

$$m[2,4] = \min\{m[2][2] + m[3][4] + p[1] \cdot p[2] \cdot p[4], \\ m[2][3] + m[4][4] + p[1] \cdot p[3] \cdot p[4]\} \\ = \min\{6000 + 10 \cdot 40 \cdot 5, 12000 + 10 \cdot 30 \cdot 5\} \\ = 8000$$

$K=2$

3.1 矩阵连乘动态规划法——实例分析

$P[0]=50$ $P[1]=10$ $P[2]=40$ $P[3]=30$ $P[4]=5$

$$m[1,2]=m[1,1]+m[2,2]+p[0]*p[1]*p[2]$$

$$=50*10*40=20000$$

0	20000	27000	10500
	0	12000	8000
		0	6000
			0

S=

0	1	1	1
	0	2	2
		0	3
			0

$$m[1,3]=\min \{m[1,1]+m[2,3]+p[0]*p[1]*p[3]$$

$$, m[1,2]+m[3,3]+p[0]*p[2]*p[3]\}$$

$$=\{12000+50*10*30, 20000+50*40*30\}$$

$$=27000$$

$k=1$

$$m[1,4]=\min \{m[1,1]+m[2,4]+p[0]*p[1]*p[4]$$

$$, m[1,2]+m[3,4]+p[0]*p[2]*p[4],$$

$$m[1,3]+m[4,4]+p[0]*p[3]*p[4]\}$$

$$=\min\{8000+2500, 20000+8000+10000,$$

$$27000+7500\}$$

$$=10500$$

$$k=1 \quad (A_1)(A_2A_3A_4)$$

$$k=2, \quad A_1 (A_2 (A_3 A_4))$$

$$\begin{aligned}
 m[1,4] &= \min \{ m[1,1] + m[2,4] + p[0] \cdot p[1] \cdot p[4], \\
 &\quad m[1,2] + m[3,4] + p[0] \cdot p[2] \cdot p[4], \\
 &\quad m[1,3] + m[4,4] + p[0] \cdot p[3] \cdot p[4] \} \\
 &= \min \{ 8000 + 2500, 20000 + 8000 + 10000, \\
 &\quad 27000 + 7500 \} \\
 &= 10500
 \end{aligned}$$

$$k=1 \quad (A_1)(A_2 A_3 A_4)$$

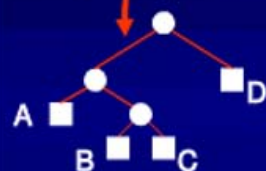
$$k=2, \quad A_1 (A_2 (A_3 A_4))$$

■ 例如

4个矩阵连乘积ABCD，设它们的维数分别为
A:50×10, B:10×40, C:40×30, D:30×5。

矩阵连乘的完全加括号形式以及相应的计算量

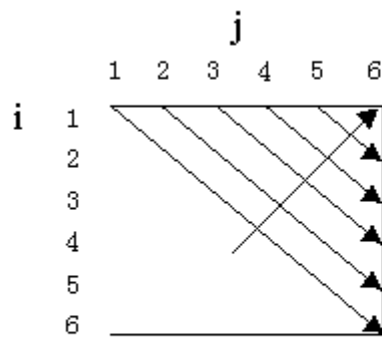
完全加括号形式	连乘积所需计算量
(A(B(CD)))	$40 \times 30 \times 5 + 10 \times 40 \times 5 + 50 \times 10 \times 5 = 10500$
(A((BC)D))	$10 \times 40 \times 30 + 10 \times 30 \times 5 + 50 \times 10 \times 5 = 16000$
((AB)(CD))	$50 \times 10 \times 40 + 40 \times 30 \times 5 + 50 \times 40 \times 5 = 36000$
(((AB)C)D)	$50 \times 10 \times 40 + 50 \times 40 \times 30 + 50 \times 30 \times 5 = 87500$
((A(BC))D)	$10 \times 40 \times 30 + 50 \times 10 \times 30 + 50 \times 30 \times 5 = 34500$



每种计算次序都可以用一个二叉树来表示

3.1 矩阵连乘动态规划法——实例分析

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



(a) 计算次序

		j					
		1	2	3	4	5	6
i	1	0	15750	7875	9375	11875	15125
	2		0	2625	4375	7125	10500
	3			0	750	2500	5375
	4				0	1000	3500
	5					0	5000
	6						0

(b) $m[i][j]$

		j					
		1	2	3	4	5	6
i	1	0	1	1	3	3	3
	2		0	2	3	3	3
	3			0	3	3	3
	4				0	4	5
	5					0	5
	6						0

(c) $s[i][j]$

$$m[2][5] = \min \begin{cases} m[2][2] + m[3][5] + p_1 p_2 p_5 = 0 + 2500 + 35 \times 15 \times 20 = 13000 \\ m[2][3] + m[4][5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \times 5 \times 20 = 7125 \\ m[2][4] + m[5][5] + p_1 p_4 p_5 = 4375 + 0 + 35 \times 10 \times 20 = 11375 \end{cases}$$

用动态规划法求最优解 ---- s数组的作用

因此，从 $s[1][n]$ 记录的信息可知计算 $A[1:n]$ 的最优加括号方式为： $(A[1 : s[1][n]])(A[s[1][n] + 1 : n])$

如：根据 $s[i][j]$ 的记录，可以计算 $A[1:6]$ 的最优加括号方式：

$(A_1 A_2 A_3)(A_4 A_5 A_6) \quad s[1][n] = 3$

$(A_1 (A_2 A_3))((A_4 A_5) A_6) \quad s[1][3] = 1, s[4][6] = 5$

$(A_1 (A_2 A_3))((A_4 A_5) A_6) \quad s[2][3] = 2, s[4][5] = 4$

3.1 矩阵连乘动态规划法——练习题

矩阵连乘实例

例：设要计算矩阵连乘 $A_1A_2A_3A_4A_5$ ，其中各矩阵的维数分别为：

$$A_1: 3 \times 5$$

$$A_2: 5 \times 10$$

$$A_3: 10 \times 4$$

$$A_4: 4 \times 2$$

$$A_5: 2 \times 5$$

3.1 矩阵连乘动态规划法——练习题

矩阵连乘实例

解:

m[i][j]	1	2	3	4	5
1	0				
2		0			
3			0		
4				0	
5					0

$$m[1][2] = m[1][1] + m[2][2] + 3 \times 5 \times 10 = 150$$

$$s[1][2] = 1$$

$$m[2][3] = m[2][2] + m[3][3] + 5 \times 10 \times 4 = 200$$

$$s[2][3] = 2$$

$$m[3][4] = m[3][3] + m[4][4] + 10 \times 4 \times 2 = 80$$

$$s[3][4] = 3$$

$$m[4][5] = m[4][4] + m[5][5] + 4 \times 2 \times 5 = 40$$

$$s[4][5] = 4$$

m[i][j]	1	2	3	4	5
1	0	150 (1)			
2		0	200 (2)		
3			0	80 (3)	
4				0	40 (4)
5					0

3.1 矩阵连乘动态规划法——练习题

$$\begin{aligned} m[1][3] &= \min\{m[1][1] + m[2][3] + 3 \times 5 \times 4 = 260, \\ &\quad m[1][2] + m[3][3] + 3 \times 10 \times 4 = 270\} \\ &= 260 \qquad s[1][3] = 1 \end{aligned}$$

$$\begin{aligned} m[2][4] &= \min\{m[2][2] + m[3][4] + 5 \times 10 \times 2 = 180, \\ &\quad m[2][3] + m[4][4] + 5 \times 4 \times 2 = 240\} \\ &= 180 \qquad s[2][3] = 2 \end{aligned}$$

$$\begin{aligned} m[3][5] &= \min\{m[3][3] + m[4][5] + 10 \times 4 \times 5 = 240, \\ &\quad m[3][4] + m[5][5] + 10 \times 2 \times 5 = 180\} \\ &= 180 \qquad s[3][5] = 4 \end{aligned}$$

$m[i][j]$	1	2	3	4	5
1	0	150 (1)	260 (1)		
2		0	200 (2)	180 (2)	
3			0	80 (3)	180 (4)
4				0	40 (4)
5					0

3.1 矩阵连乘动态规划法——练习题

$$\begin{aligned}
 m[1][4] &= \min\{m[1][1] + m[2][4] + 3 \times 5 \times 2 = 210, \\
 &\quad m[1][2] + m[3][4] + 3 \times 10 \times 2 = 290, \\
 &\quad m[1][3] + m[4][4] + 3 \times 4 \times 2 = 284\} \\
 &= 210 \qquad s[1][4] = 1 \\
 m[2][5] &= \min\{m[2][2] + m[3][5] + 5 \times 10 \times 5 = 430, \\
 &\quad m[2][3] + m[4][5] + 5 \times 4 \times 5 = 300, \\
 &\quad m[2][4] + m[5][5] + 5 \times 2 \times 5 = 230\} \\
 &= 230 \qquad s[2][5] = 4
 \end{aligned}$$

$m[i][j]$	1	2	3	4	5
1	0	150 (1)	260 (1)	210 (1)	
2		0	200 (2)	180 (2)	230 (4)
3			0	80 (3)	180 (4)
4				0	40 (4)
5					0

3.1 矩阵连乘动态规划法——练习题

$$\begin{aligned}
 m[1][5] &= \min\{m[1][1] + m[2][5] + 3 \times 5 \times 5 = 305, \\
 &\quad m[1][2] + m[3][5] + 3 \times 10 \times 5 = 480, \\
 &\quad m[1][3] + m[4][5] + 3 \times 4 \times 5 = 360, \\
 &\quad m[1][4] + m[5][5] + 3 \times 2 \times 5 = 240\} \\
 &= 240 \qquad s[1][5] = 4
 \end{aligned}$$

$m[i][j]$	1	2	3	4	5
1	0	150 (1)	260 (1)	210 (1)	240 (4)
2		0	200 (2)	180 (2)	230 (4)
3			0	80 (3)	180 (4)
4				0	40 (4)
5					0

最优值 = 240

回溯求最优解:

$s[1][5] = 4$ $(A_1 A_2 A_3 A_4) A_5$
 $s[1][4] = 1$ $(A_1 (A_2 A_3 A_4)) A_5$
 $s[2][4] = 2$ $(A_1 (A_2 (A_3 A_4))) A_5$

3.2 动态规划算法的基本要素

➤ 从计算矩阵连乘积最优计算次序的动态规划算法可以看出，该算法的有效性依赖于问题本身所具有的两个重要性质：**最优子结构性**和**子问题重叠性质**。从一般意义上讲，问题的这两个重要性质是该问题可以用动态规划算法求解的基本要素：

- **最优子结构**
- **重叠子问题**
- **备忘录方法**

3.2 动态规划算法的基本要素--备忘录方法

- 属于动态规划算法的变形。也用一個表格来保存已解决的子问题的答案。
- 区别：备忘录方法的递归方式是自顶向下的，而动态规划算法是自底向上的。

```
int LookupChain(int i, int j)
{
    if (m[i][j] > 0) return m[i][j];
    if (i == j) return 0;
    int u = LookupChain(i, i) + LookupChain(i+1, j) + p[i-1]*p[i]*p[j];
    s[i][j] = i;
    for (int k = i+1; k < j; k++) {
        int t = LookupChain(i, k) + LookupChain(k+1, j) + p[i-1]*p[k]*p[j];
        if (t < u) { u = t; s[i][j] = k;}
    }
    m[i][j] = u;
    return u;
}
```

关于动态规划算法和备忘录方法的适用条件

- 综上所述，矩阵连乘积的最优计算次序问题可用自顶向下的备忘录方法或自底向上的动态规划算法在 $O(n^3)$ 计算时间内求解。
- 这两个算法都利用了子问题重叠性质。总共有 $\theta(n^2)$ 个不同的子问题，对每个子问题两种算法都只解一次并记录答案。当再次遇到该子问题时，简单地取用已得到的答案，节省了计算量，提高了算法的效率。
- **适用条件**：一般来说，当一个问题所有子问题都至少要解一次时，用动态规划算法比用备忘录方法好。此时，动态规划算法没有任何多余的计算，还可以利用其规则的表格存取方式来减少在动态规划算法中的计算时间和空间需求。当子问题空间中部分子问题可以不必求解时，易用备忘录方法则较为有利，因为其控制结构可以看出，该方法只解那些确实需要求解的子问题。