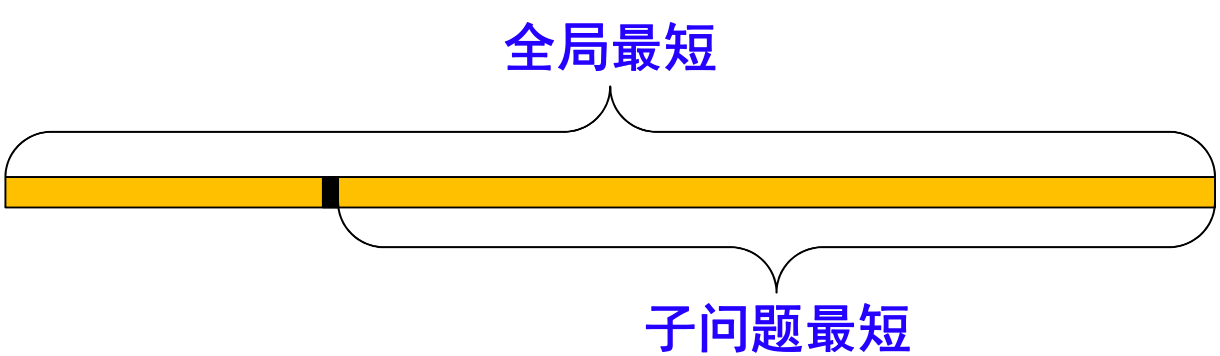
算法分析与设计

Analysis and Design of Algorithm

Lesson 08

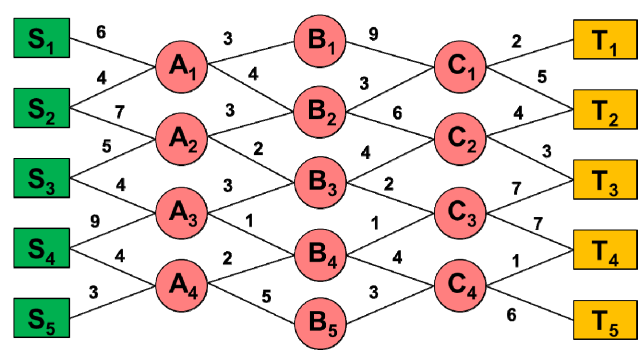
要点回顾

- 动态规划概述
  - 最优化问题
    - 概念：约束条件、可行解、目标函数、最优解
    - 例子：POS机找零钱
  - 最优性原理/优化原则
    - 最优子结构性质
    - 反例：总长模10的最小路径



要点回顾(cont.)

- 动态规划算法的设计要点
  - 建模：目标函数、约束条件
  - 分段：确定子问题的边界
  - 分析：原始问题与子问题之间的依赖关系
  - 判断：最优子结构性质
  - 求解：先定最小子问题（初值），自底向上求解
- 动态规划算法实例
  - 最短路径问题



矩阵连乘背景

- 给定*n*个矩阵 $\{A_1, A_2, ..., A_n\}$ ，其中 A_i 与 A_{i+1} 是可乘的，$i=1,2,...,n-1$ 。考察这*n*个矩阵的连乘积

$$A_1A_2...A_n$$

- 由于矩阵乘法满足结合律，所以计算矩阵的连乘可以有许多不同的计算次序。这种计算次序可以用加括号的方式来确定。
- 若一个矩阵连乘积的计算次序完全确定，也就是说该连乘积已完全加括号，则可以依此次序反复调用2个矩阵相乘的标准算法计算出矩阵连乘积。

矩阵相乘基本运算次数

- 矩阵A：i行j列，B：j行k列，以元素相乘作基本运算，计算AB的工作量

$$\begin{bmatrix} & & \\ \boxed{a_{t1} \quad \cdots \quad a_{tj}} \\ & & \end{bmatrix} \begin{bmatrix} & \boxed{\begin{matrix} b_{1s} \\ \vdots \\ b_{js} \end{matrix}} & \end{bmatrix} = \begin{bmatrix} & & \\ & \boxed{c_{ts}} & \\ & & \end{bmatrix}$$

$$c_{ts} = a_{t1} \times b_{1s} + \cdots + a_{tj} \times b_{js}$$

- AB：i行k列的矩阵，计算每个元素需要作j次乘法，总计乘法次数为：$ik \times j = ijk$

矩阵连乘背景

- 给定*n*个矩阵 $\{A_1, A_2, \ldots, A_n\}$ ，其中 A_i 与 A_{i+1} 是可乘的，$i = 1,2,\ldots,n-1$ 。考察这*n*个矩阵的连乘积

$$A_1 A_2 \ldots A_n$$

- 由于矩阵乘法满足结合律，所以计算矩阵的连乘可以有许多不同的计算次序。这种计算次序可以用加括号的方式来确定。

例子：设有四个矩阵*A*,*B*,*C*,*D*，它们的维数分别是：

$$A = 50 \times 10 \quad B = 10 \times 40 \quad C = 40 \times 30 \quad D = 30 \times 5$$

$(A((BC)D))$ **16000** $(A(B(CD)))$ **10500** $((AB)(CD))$ **36000**

$(((AB)C)D)$ **87500** $((A(BC))D)$ **34500**

矩阵连乘问题

问题： 给定n个矩阵｛$A_1,A_2,\ldots,A_n$｝，其中A_i为$P_{i-1}\times P_i$阶矩阵，i=1，2…，n。

试确定计算矩阵连乘积的计算次序，使得矩阵链相乘需要的**总次数最少**。

输入： 向量$P=<P_0,P_1,\ldots,P_n>$，其中P_0，P_1，…，P_n为n个矩阵的行数与列数。

输出： 矩阵链乘法加括号的位置。

矩阵连乘问题

实例：P=<10, 100, 5, 50>

A_1:10×100, A_2:100×5, A_3:5×50。

乘法次序：

$(A_1A_2)\ A_3$: 10×100×5+10×5×50=7500

$A_1(A_2A_3)$: 10×100×50+100×5×50=75000

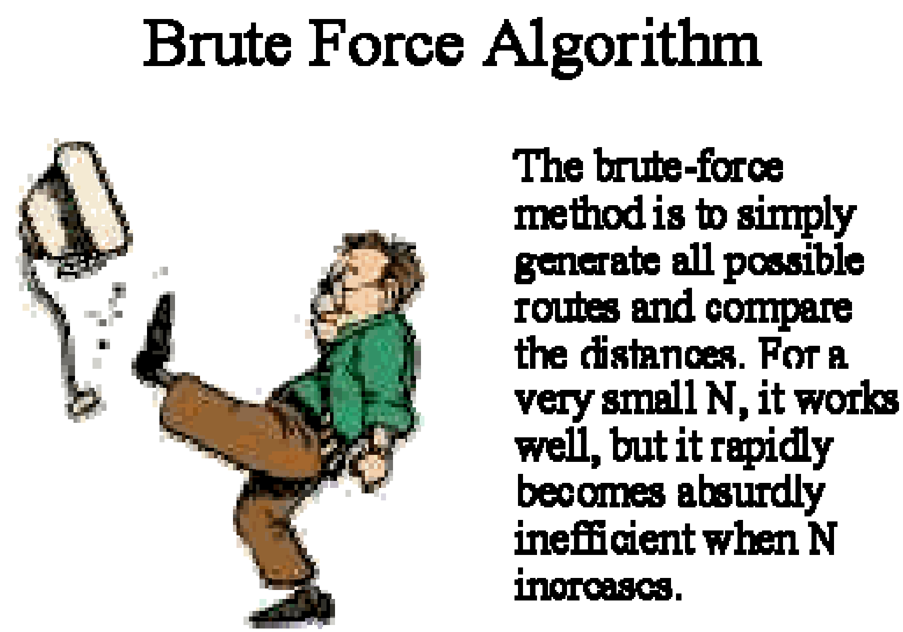
第一种次序计算次数最少。

矩阵连乘问题

实例： P=<25, 2, 40, 15, 30>

A:25×2, B:2×40, C:40×15, D:15×30。

穷举法： 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。



矩阵连乘问题

实例： $P=<25, 2, 40, 15, 30>$

A:25×2, B:2×40, C:40×15, D:15×30。

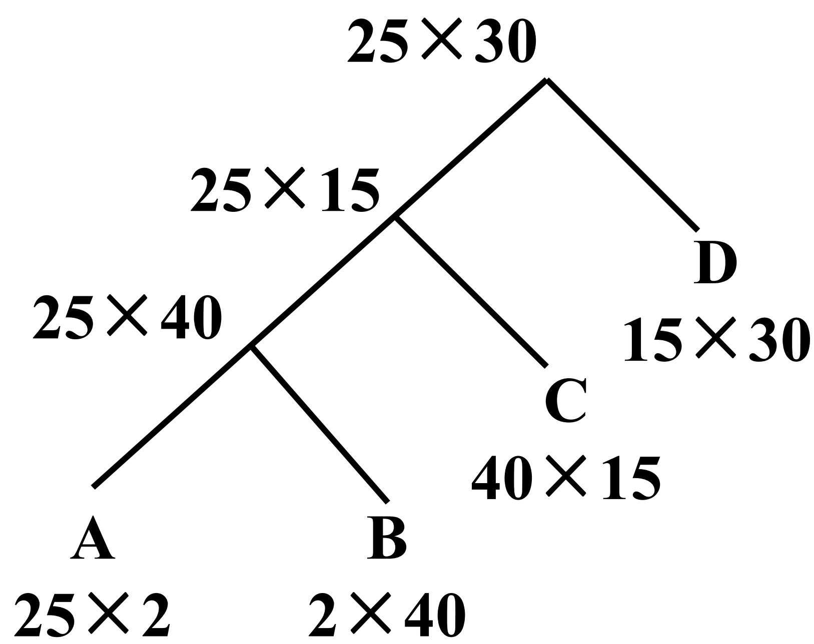
① (((AB)C)D)

$= 25\times2\times40$

$+ 25\times40\times15$

$+ 25\times15\times30$

$=$ **28 250**



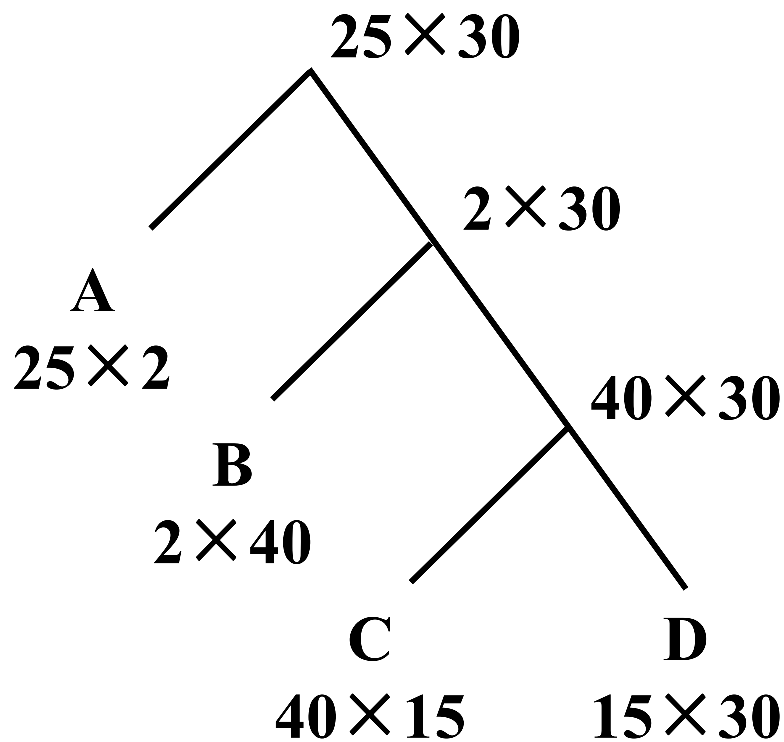
矩阵连乘问题

实例： $P=<25, 2, 40, 15, 30>$

A:25×2, B:2×40, C:40×15, D:15×30。

② **(A(B(CD)))**

$$= 40\times15\times30 + 2\times40\times30 + 25\times2\times30 = 21\,900$$



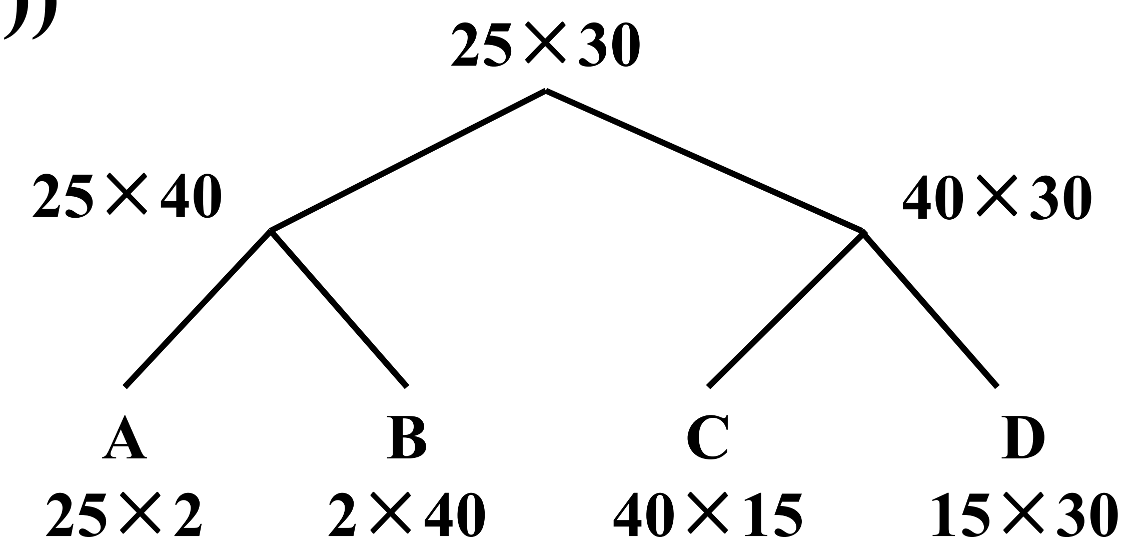
矩阵连乘问题

实例： P=<25, 2, 40, 15, 30>

A:25×2, B:2×40, C:40×15, D:15×30。

③ **((AB)(CD))**

＝ 50 000



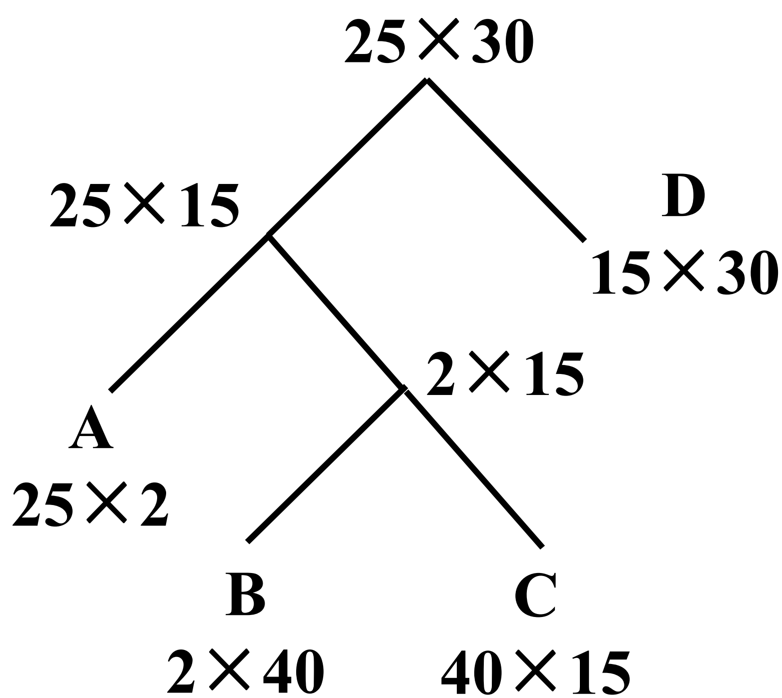
矩阵连乘问题

实例： $P=<25, 2, 40, 15, 30>$

A:25×2, B:2×40, C:40×15, D:15×30。

④ ((A(BC))D)

= 13 200



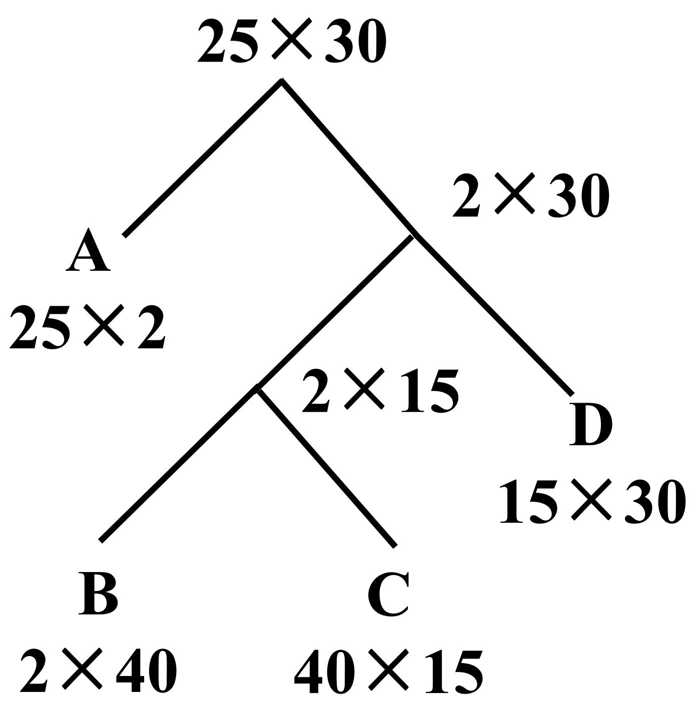
矩阵连乘问题

实例：P=<25, 2, 40, 15, 30>

A:25×2, B:2×40, C:40×15, D:15×30。

⑤ (A((BC)D))

= 3600



矩阵连乘问题穷举法

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：$(A_1...A_k)(A_{k+1}...A_n)$可以得到关于$P(n)$的递推式如下：

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

Catalan数

即：$P(n)=C(n-1)=\frac{1}{(n-1)+1}\binom{2(n-1)}{n-1}$

矩阵连乘问题穷举法

- **Stirling公式计算Catalan数**
- **不失一般性，假设：**

$$T(n)=\frac{1}{n+1}\times\binom{2n}{n}=\Omega\left(\frac{1}{n+1}\times\frac{(2n)!}{n!\times n!}\right)$$

$$=\Omega\left(\frac{1}{n+1}\times\frac{\sqrt{2\pi 2n}\left(\frac{2n}{e}\right)^{2n}}{\sqrt{2\pi n}\left(\frac{n}{e}\right)^{n}\times\sqrt{2\pi n}\left(\frac{n}{e}\right)^{n}}\right)=\Omega\left(\frac{2^{2n}}{n^{\frac{3}{2}}}\right)$$

复杂度是指数量级！！！

矩阵连乘问题动态规划法

1. 分段：子问题划分

将矩阵连乘积 $A_iA_{i+1}\dots A_j$ 简记为$A[i:j]$，这里$i\leqslant j$
输入向量：$<P_{i-1},P_i,\dots,P_j>$
其最好划分的运算次数：$m[i,j]$

2. 分析：子问题的依赖关系

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，$i\leqslant k<j$，即最优划分最后一次相乘发生在矩阵k的位置，则其相应完全加括号方式为 $(A_iA_{i+1}\dots A_k)(A_{k+1}A_{k+2}\dots A_j)$

矩阵连乘问题动态规划法

递推方程：

$m[i,j]$：得到$A[i:j]$的最少的相乘次数。可递归定义为：

$$
m[i,j]=\begin{cases} 0 & i=j \\ \min\limits_{i\le k<j}\{\boxed{m[i,k]}+\boxed{m[k+1,j]}+\boxed{p_{i-1}p_k p_j}\} & i<j \end{cases}
$$

$$
\boxed{(A_iA_{i+1}\ldots A_k)}\boxed{(A_{k+1}A_{k+2}\ldots A_j)}
$$

$$
\boxed{P_{i-1}\times P_k \qquad\qquad P_k\times P_j}
$$

该问题满足优化原则！

矩阵连乘问题动态规划法

3. 求解：计算最优值

- 对于$1 \leqslant i \leqslant j \leqslant n$不同的有序对$(i,j)$对应于不同的子问题。因此，不同子问题的个数最多只有

$$\binom{n}{2} + n = \Theta(n^2)$$

- 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

递归实现动态规划的部分伪码

- 算法1： RecurMatrixChain(P, i, j)

1. $m[i,j] \leftarrow \infty$
2. $s[i,j] \leftarrow i$
3. **for** $k \leftarrow i$ **to** j-1 **do**
4. $\quad q \leftarrow$ RecurMatrixChain(P, i, k)
   $\quad\quad$ + RecurMatrixChain(P, k+1, j) + $P_{i-1}P_kP_j$
5. $\quad$ **if** $q < m[i,j]$
6. $\quad$ **then** $m[i,j] \leftarrow q$
7. $\quad\quad s[i,j] \leftarrow k$
8. **return** $m[i,j]$

划分位置k

子问题$i{:}j$

找到了
更好的解

这里没有写出算法的全部描述（递归边界）

算法分析

- **时间复杂度的递推方程**

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

$$T(n) \geq O(n) + \sum_{k=1}^{n-1} T(k) + \sum_{k=1}^{n-1} T(n-k)$$

$$\boxed{T(n) \geq O(n) + 2\sum_{k=1}^{n-1} T(k)}$$

可以证明还是一个指数函数

时间复杂度

- 数学归纳法证明：$T(n) \geqslant 2^{n-1}$
  - $n=2$, 显然为真
  - 假设对于任何小于n的k，命题为真

$$T(n) \ge O(n) + 2\sum_{k=1}^{n-1} T(k)$$

代入归纳假设

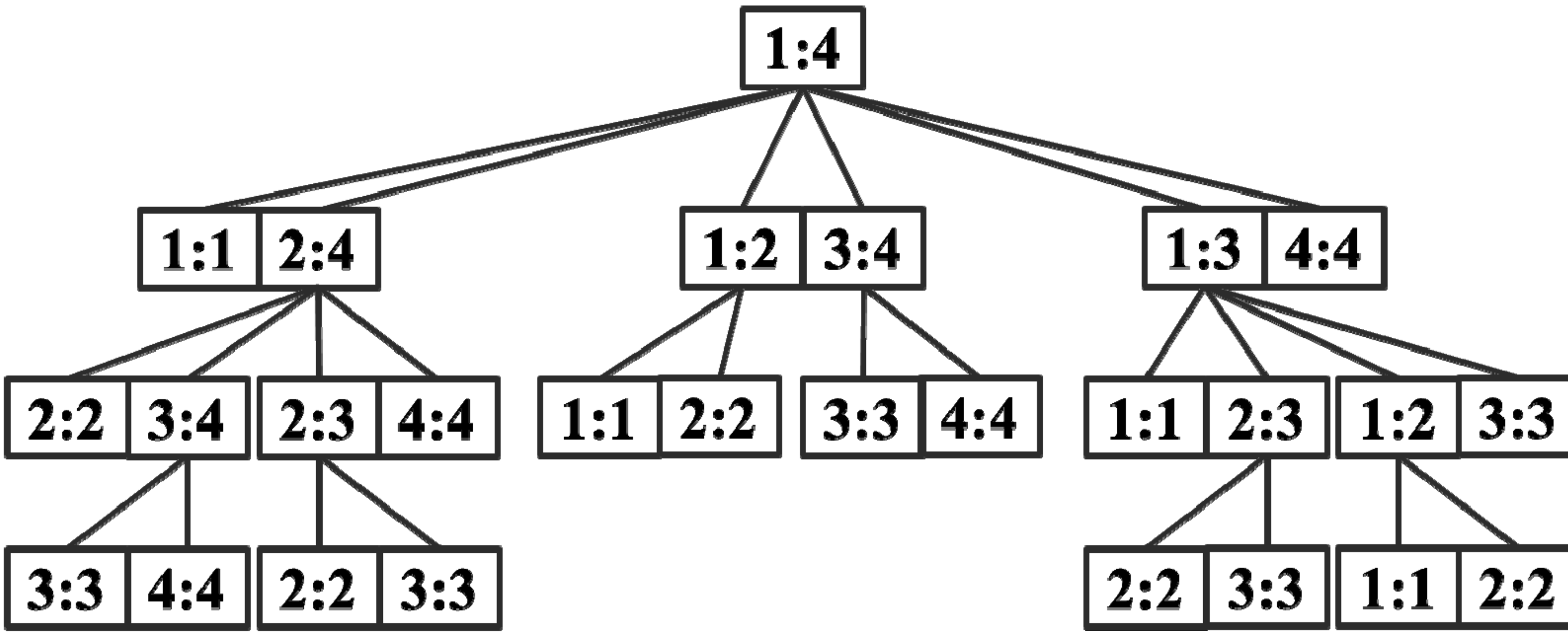
$$\ge O(n) + 2\sum_{k=1}^{n-1} 2^{k-1}$$

等比数列求和

$$= O(n) + 2(2^{n-1} - 1) \ge 2^{n-1}$$

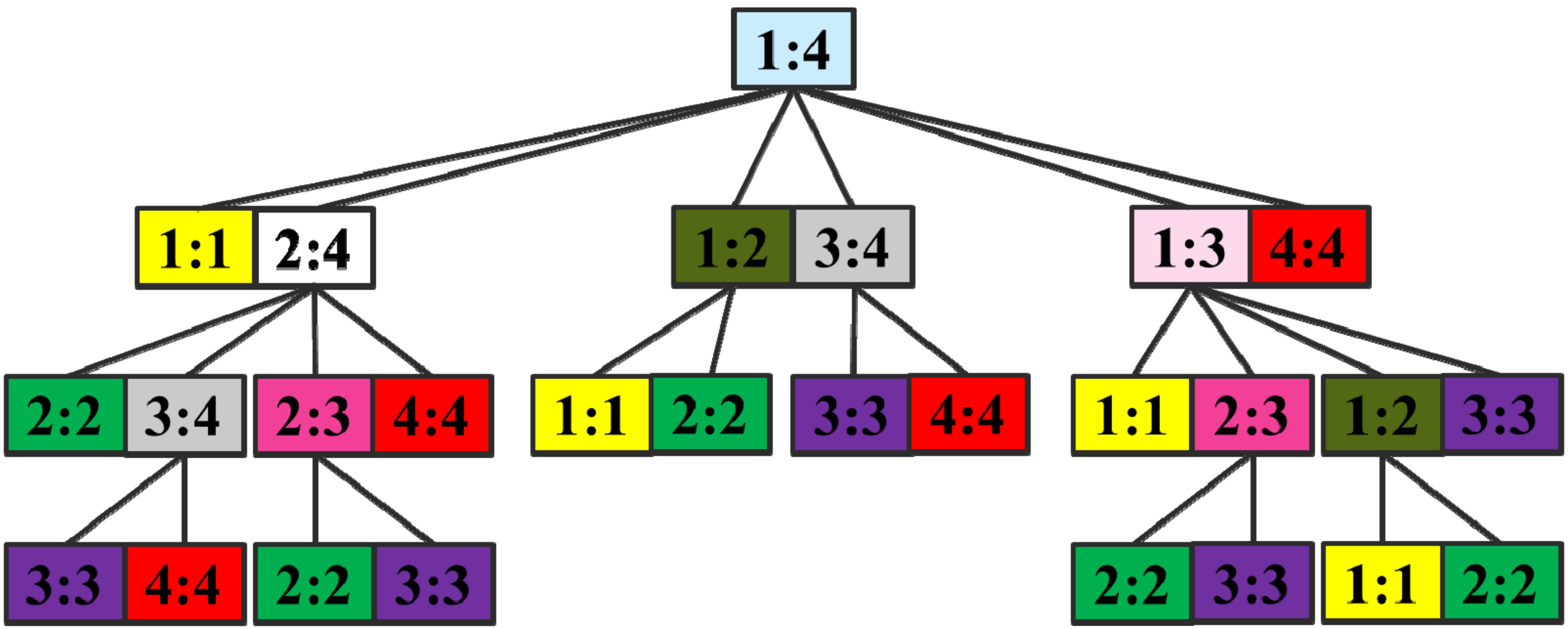
子问题的产生

用上述算法RecurMatrixChain(1,4)计算A[1:4]的递归树：



子问题的产生

用上述算法RecurMatrixChain(1,4)计算A[1:4]的递归树：



子问题的计数

边界	次数	边界	次数	边界	次数
1:1	4	1:2	2	1:3	1
2:2	5	2:3	2	2:4	1
3:3	5	3:4	2		
4:4	4			1:4	1

边界不同的子问题：**10**个

递归计算的子问题：**27**个

当n=5的时候，上面两个数值分别是15和81

小结

- 与穷举法相比较，动态规划算法利用了子问题优化函数间的依赖关系，时间复杂度有所降低
- 动态规划算法的递归实现效率并不高，原因在于同一子问题多次重复出现，每次出现都需要重新计算一遍
- 还有没有改进的空间?

动态规划算法的迭代实现

- **思想：** 采用空间换时间策略，记录每个子问题首次计算结果，后面再用时就直接取值，每个子问题只计算一次！

迭代计算的关键

- 每个子问题只计算一次
- 迭代过程
  - 从最小的子问题算起
  - 考虑计算顺序，以保证后面用到的值前面已经计算好了
  - 存储结构保存计算结果——备忘录
- 解的追踪
  - 设计标记函数标记每步的决策
  - 考虑根据标记函数追踪解的算法

矩阵链乘法不同子问题

- 长度1：只含1个矩阵，有n个子问题（不需要计算）
- 长度2：含2个矩阵，n-1个子问题
- 长度3：含3个矩阵，n-2个子问题

…

- 长度n-1：含n-1个矩阵，2个子问题
- 长度n：原始问题，只有1个

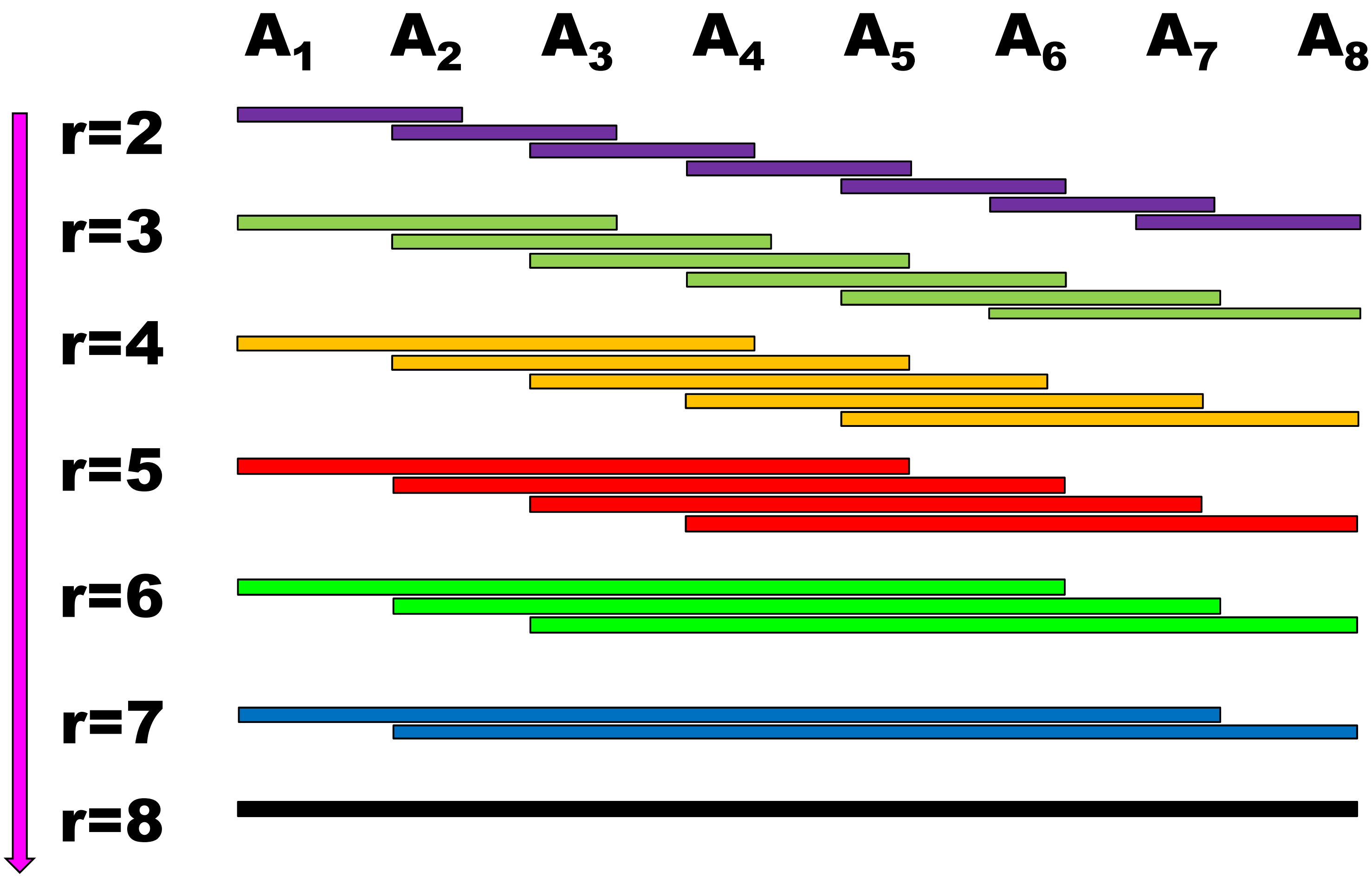
矩阵链乘法迭代顺序

- 长度为1：初值，$m[i,i]=0$
- 长度为2：1:2, 2:3, 3:4, …, n-1:n
- 长度为3：1:3, 2:4, 3:5, …, n-2:n

…

- 长度为n-1：1:n-1, 2:n
- 长度为n：1:n

n=8的子问题计算顺序



部分伪码

二维数组m与s为备忘录

- 算法MatrixChain(P, n)

```
令所有的m[i,j]初值为0
for r←2 to n do                          //r为链长
    for i←1 to n-r+1 do                  //左边界i
        j← i+r-1                         //右边界j
        m[i,j]← m[i+1,j]+P_{i-1}P_iP_j   //k=i
        s[i,j]←i                         //记录k
        for k←i+1 to j-1 do              //遍历k
            t← m[i,k]+ m[k+1,j]+P_{i-1}P_kP_j
            if t< m[i,j]
            then m[i,j]←t                //更新解
                 s[i,j]←k
```

遍历长度为r子问题

遍历所有划分

时间复杂度

1、根据伪码：行2，3，7都是$O(n)$，循环执行$O(n^3)$次，内部为$O(1)$

$$T(n)=O(n^3)$$

2、根据备忘录：估计每项工作量，求和。子问题有$O(n^2)$个，确定每个子问题的最少乘法次数需要对不同划分位置比较，需要$O(n)$时间。

$$T(n)=O(n^3)$$

追踪解工作量$O(n)$，总工作量$O(n^3)$

实例

- **输入：** P=<30, 35, 15, 5, 10, 20>

  n=5
- **矩阵链：** $A_1 A_2 A_3 A_4 A_5$，其中

  A_1:30×35, A_2:35×15, A_3:15×5,

  A_4:5×10, A_5:10×20
- **备忘录：** 存储所有子问题的最小乘法次数及得到这个值的划分位置

备忘录 *m*[*i*, *j*]

- P=<30, 35, 15, 5, 10, 20>

r=1	m[1,1]= 0	m[2,2]=0	m[3,3]=0	m[4,4]=0	m[5,5]=0
r=2	m[1,2]= 15750	m[2,3]= 2625	m[3,4]= 750	m[4,5]=1000	
r=3	m[1,3]= 7875	m[2,4]= 4375	m[3,5]= 2500		
r=4	m[1,4]= 9375	m[2,5]= 7125			
r=5	m[1,5]= 11875				

标记函数 $s[i, j]$

r=2	s[1,2]=1	s[2,3]=2	s[3,4]=3	s[4,5]=4
r=3	s[1,3]=1	s[2,4]=3	s[3,5]=3	
r=4	s[1,4]=3	s[2,5]=3		
r=5	s[1,5]=3			

解的追踪：s[1,5]=3 ➔ $(A_1 A_2 A_3)(A_4 A_5)$

s[1,3]=1 ➔ $A_1(A_2 A_3)$

输出

计算顺序为：$(A_1(A_2 A_3))(A_4 A_5)$

最少的乘法次数：m[1,5]=11875

两种实现的比较

- 递归实现：时间复杂度高，空间开销小
- 迭代实现：时间复杂度低，空间消耗多
  - 原因：递归实现子问题多次重复计算，子问题计算次数呈指数增长。迭代实现每个子问题只计算一次
- 动态规划时间复杂度：
  - 备忘录各项计算量之和+追踪解工作量
  - 追踪解的工作量通常是问题规模的多项式函数，一般不会超过计算的工作量