算法分析与设计

Analysis and Design of Algorithm

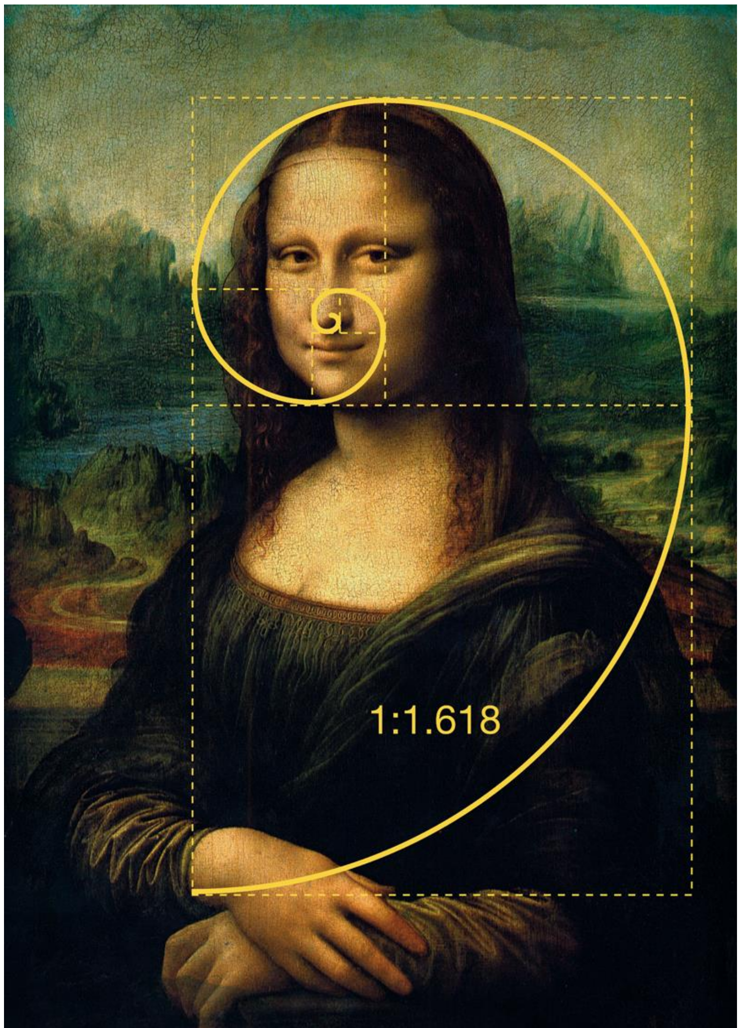
第7次课

第三章 动态规划

课程回顾

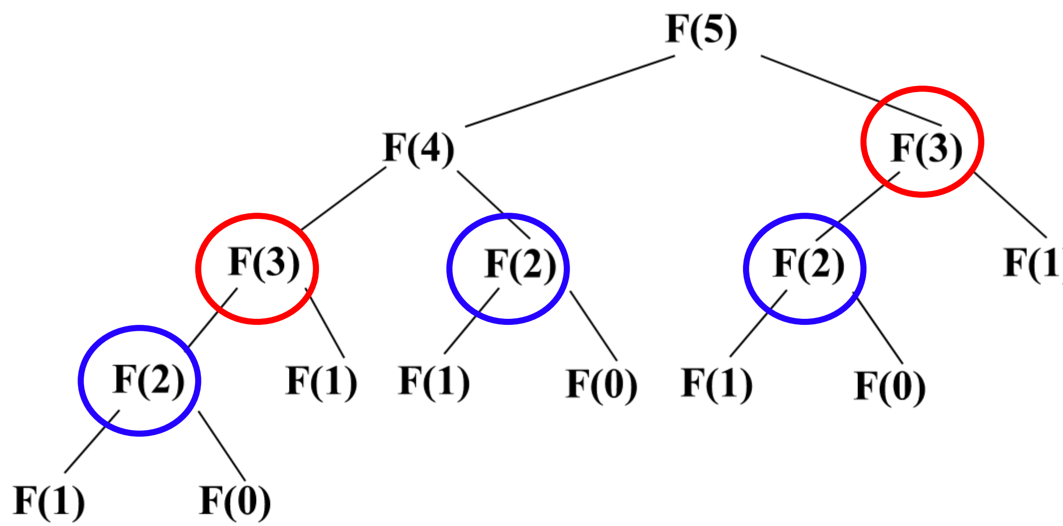
- 理解动态规划算法的概念
- 掌握动态规划算法的基本要素
  - 最优子结构性质
  - 重叠子问题性质
- 掌握设计动态规划算法的步骤。
  - 找出最优解的性质，并刻划其结构特征
  - 递归地定义最优值
  - 以自底向上的方式计算出最优值
  - 根据计算最优值时得到的信息，构造最优解

从Fibonacci数列开始



$$F(n)=\begin{cases}1 & , \quad n=0 \\ 1 & , \quad n=1 \\ F(n-1)+F(n-2) & , \quad n>1\end{cases}$$

n=5时分治法计算斐波那契数的过程：



例：计算Fibonacci数列

分析：注意到，计算$F(n)$是以计算它的两个重叠子问题$F(n-1)$和$F(n-2)$的形式来表达的，所以，可以设计一张表填入$n+1$个$F(n)$的值。

方案：可以将中间结果缓存到表格中，则斐波那契数$F(9)$的填表过程：

n	0	1	2	3	4	5	6	7	8	9
$F(n)$	0	1	1	2	3	5	8	13	21	34

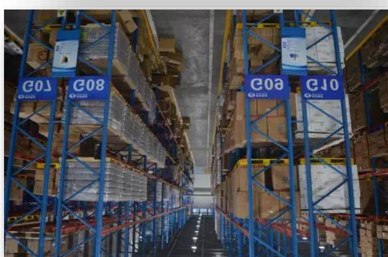
例：一个最短路径问题

- **问题**：找任意起点到任意终点的一条最短路径

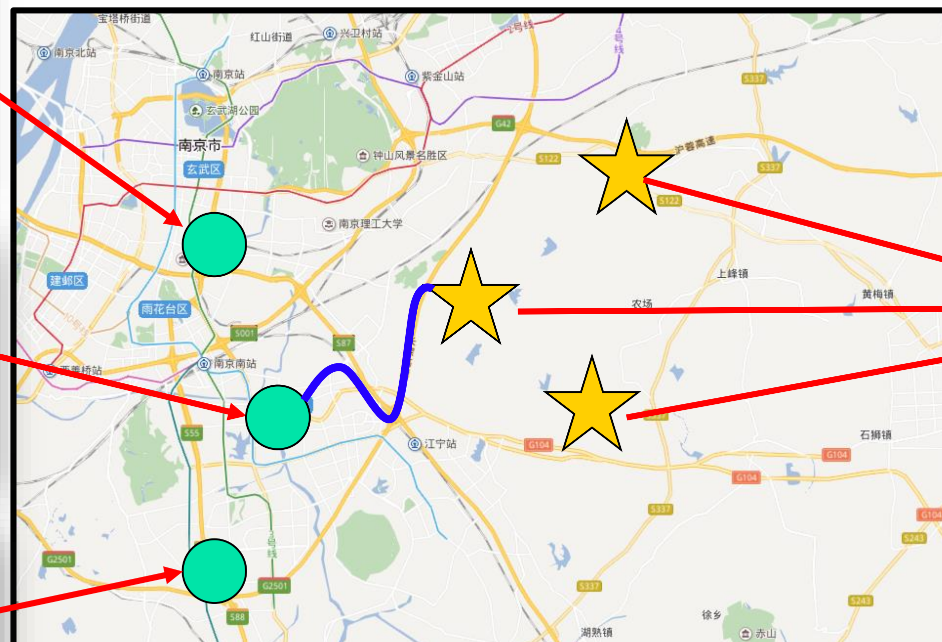
（例如：加工厂的选址）

仓库1

仓库2



仓库3



加工厂候选地址

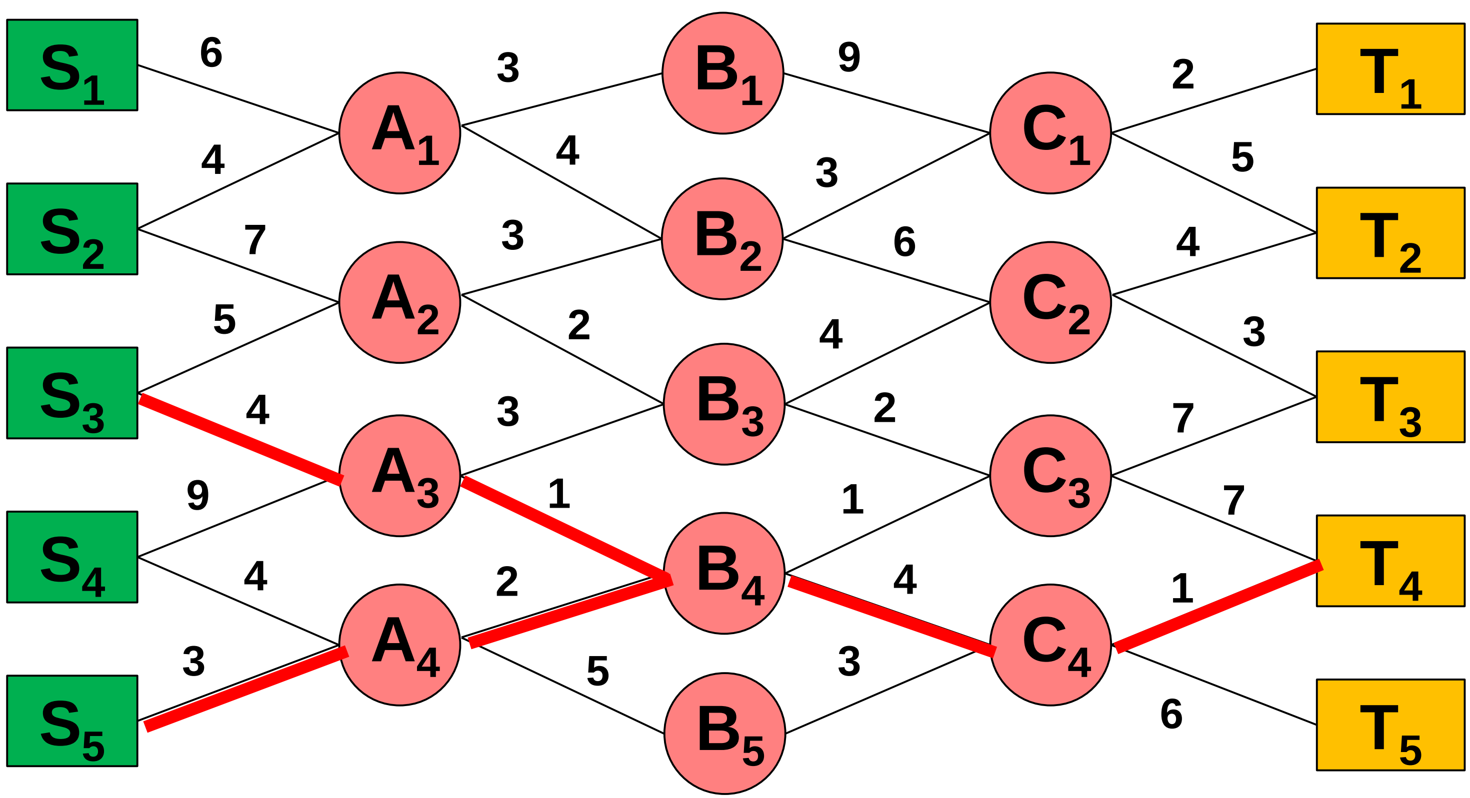
例：一个最短路径问题

- **问题：** 找任意起点到任意终点的一条最短路径

- **输入：**
  - 起点集合$\{S_1,S_2,\ldots,S_n\}$
  - 终点集合$\{T_1,T_2,\ldots,T_m\}$
  - 中间结点集合，边集，对于任意边e有长度
- **输出：** 一条从起点到终点的最短路

例：一个最短路径问题

- 一个实例



例：一个最短路径问题

- 最短路径的依赖关系

$$\boxed{F(C_l)} = \min_m \{C_l T_m\}$$ 决策1

$$\boxed{F(B_k)} = \min_l \{B_k C_l + \boxed{F(C_l)}\}$$ 决策2

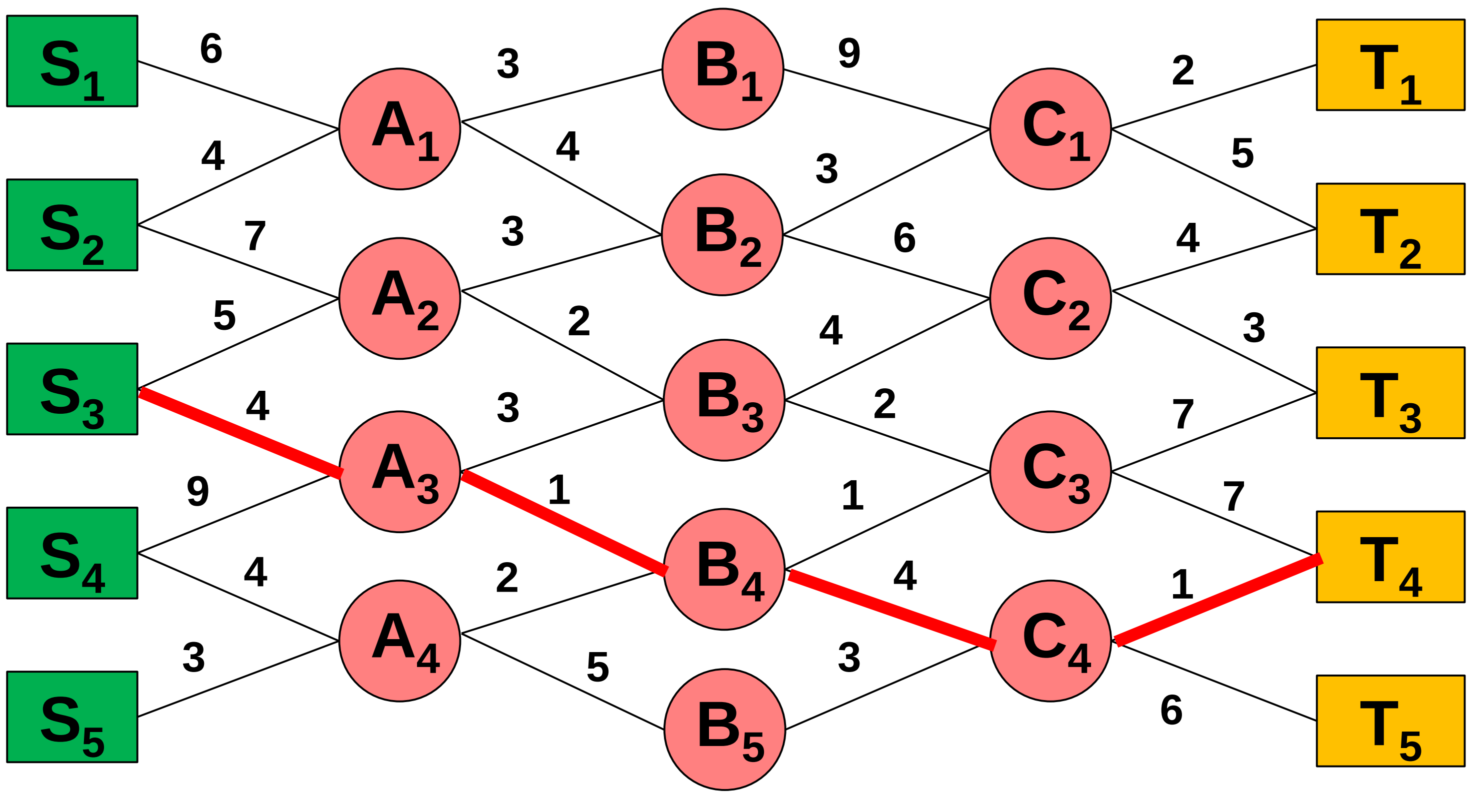
$$\boxed{F(A_j)} = \min_k \{A_j B_k + \boxed{F(B_k)}\}$$ 决策3

$$F(S_i) = \min_j \{S_i A_j + \boxed{F(A_j)}\}$$ 决策4

优化函数值之间存在依赖关系

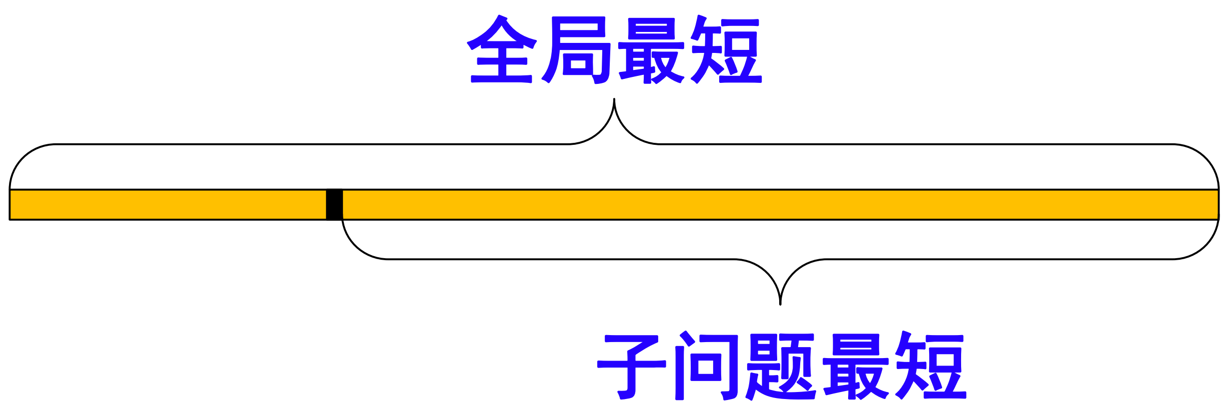
优化原则：最优子结构性质

- **优化函数的特点：**任何最短路的子路径相对于子问题始、终点最短



优化原则：最优子结构性质

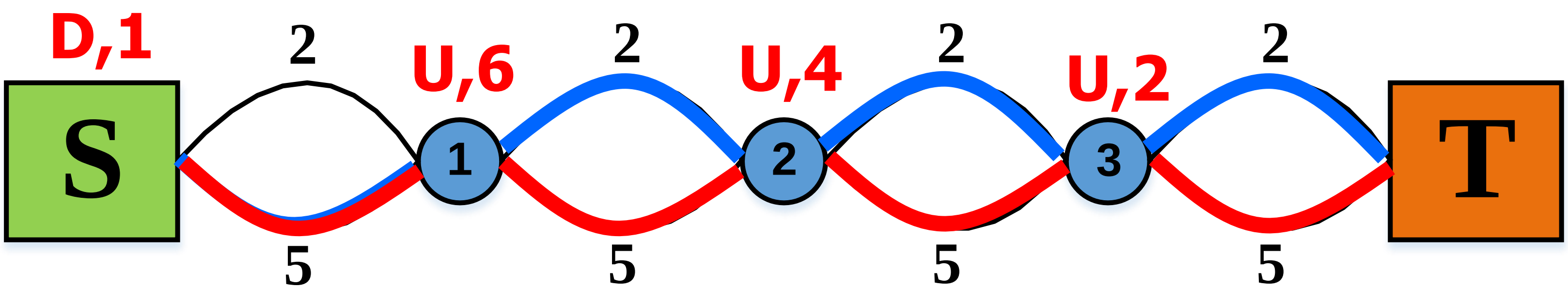
- **优化函数的特点：** 任何最短路的子路径相对于子问题始、终点最短



- **优化原则：** 一个最优决策序列的任何子序列本身一定是相对于子序列的初始和结束状态的最优决策序列

一个反例

- 求总长模10的最小路径



➔动态规划算法的解：下、上、上、上

- **最优解：**下、下、下、下

不满足优化原则，不能用动态规划！！！

典型的动态规划问题

完全加括号的矩阵连乘问题

中国MOOC 5.3 动态规划算法设计
5.4 动态规划算法的递归实现
5.5 动态规划算法的迭代实现

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

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

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

- 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：$i \times k \times j$

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。

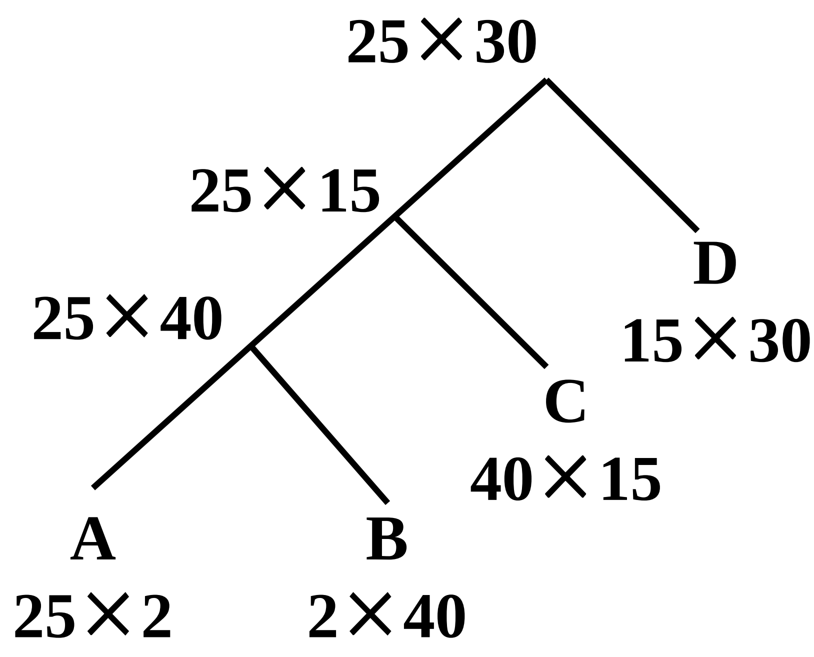
① (((AB)C)D)

= 25×2×40

+ 25×40×15

+ 25×15×30

= **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。

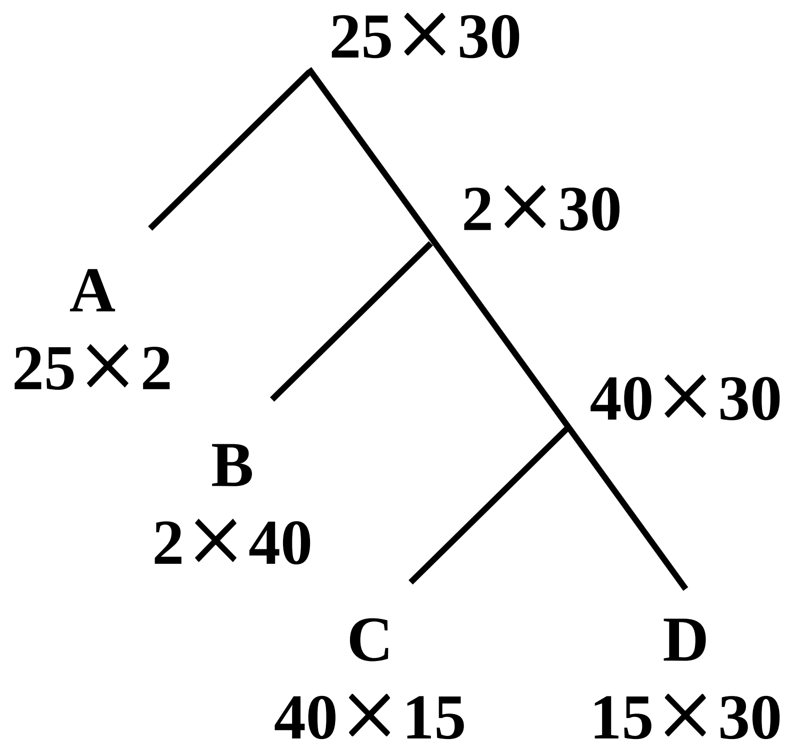
② (A(B(CD)))

= 40×15×30

\+ 2×40×30

\+ 25×2×30

= 21 900



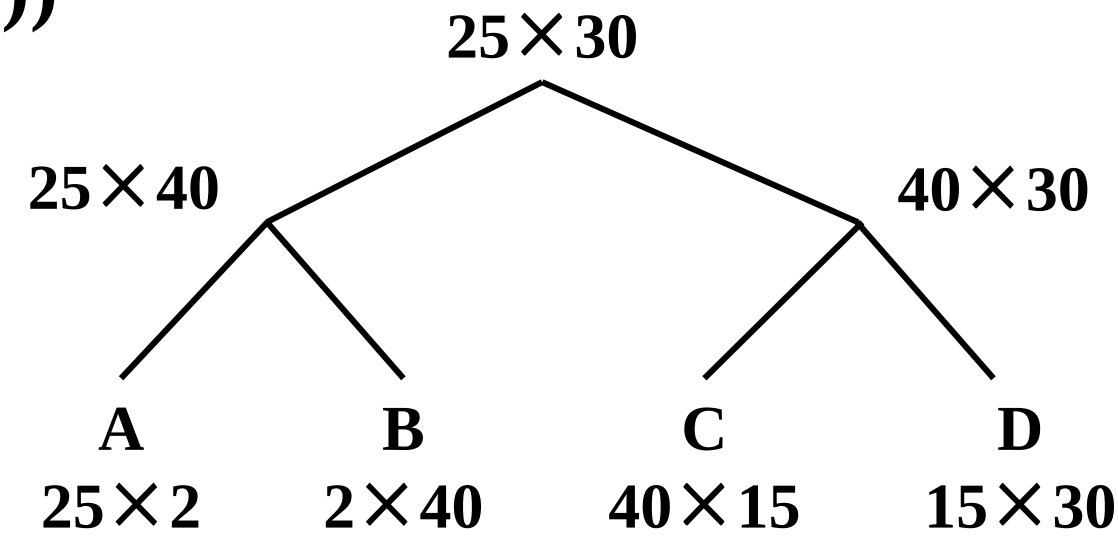
矩阵连乘问题

实例： $P=\langle 25, 2, 40, 15, 30\rangle$

A:25×2, B:2×40, C:40×15, D:15×30。

③ ((AB)(CD))

= **50 000**



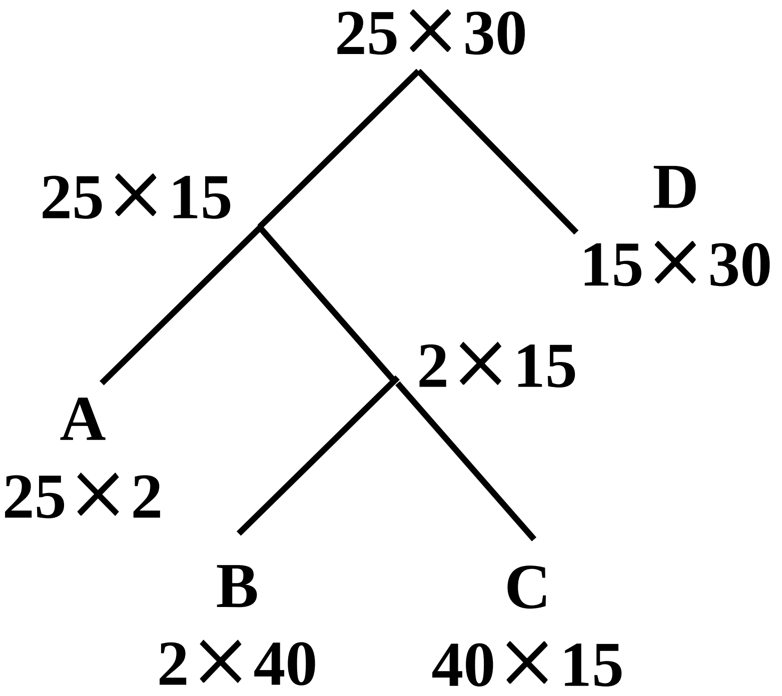
矩阵连乘问题

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



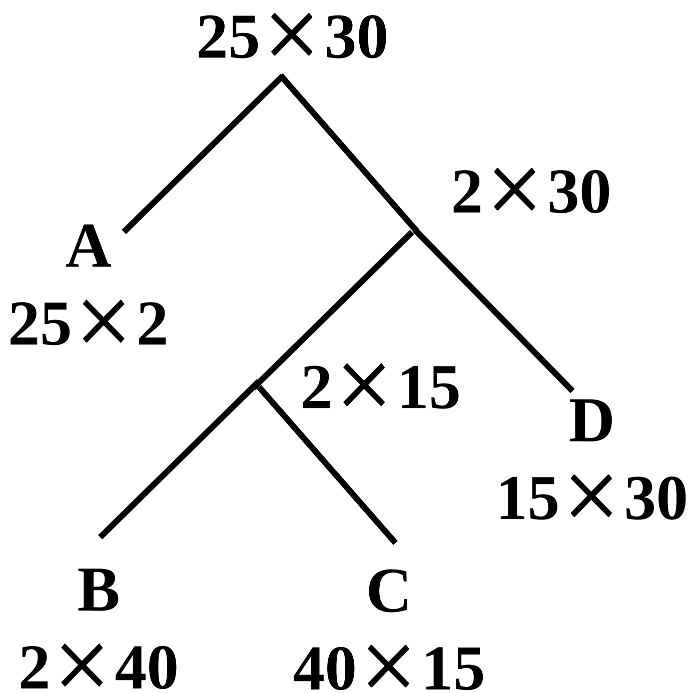
矩阵连乘问题

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

- **加*n*个括号的方法有** $\frac{1}{n+1}\binom{2n}{n}$ **种，这是一个Catalan数**

代入Stirling公式

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

矩阵连乘问题穷举法

- 加n个括号的方法有 $\frac{1}{n+1}\binom{2n}{n}$ 种，这是一个Catalan数

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

代入Stirling公式

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

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 $A_i A_{i+1} \ldots A_j$ 简记为$A[i:j]$，这里$i \leqslant j$

输入向量：$\langle P_{i-1}, P_i, \ldots, P_j \rangle$

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

矩阵连乘问题动态规划法

优化函数的递推方程：

$m[i,j]$：得到$A[i:j]$的最少的相乘次数。可递归定义为：

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

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

该问题满足优化原则！

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

一种基于递归的算法

- **算法1： RecurMatrixChain(P, i, j)**

```
m[i,j]←∞
s[i, j]←i
    k←i to j-1 do
        curMatrixChain(P, i, k)
        urMatrixChain(P, k+1, j) + P_{i-1}P_kP_j
        m[i,j]
            then m[i,j]←q
                 s[i,j]←k
return m[i,j]
```

划分位置k

子问题i:j

$m[i, j]$存储 $j]$的

最

$s[i, j]$存储$A[i, j]$
断开的位置

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**

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

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

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

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

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

代入归纳假设

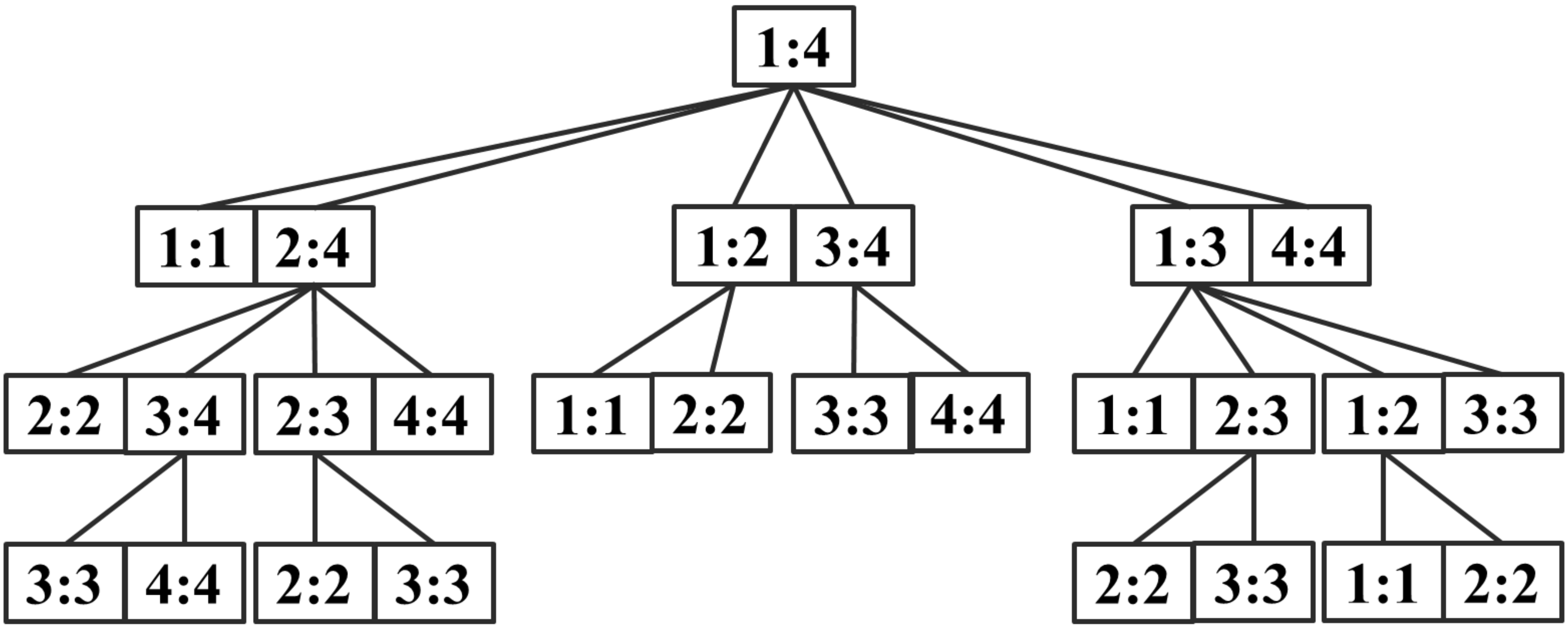
$$\geq O(n) + 2\sum_{k=1}^{n-1} 2^{k-1}$$

等比数列求和

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

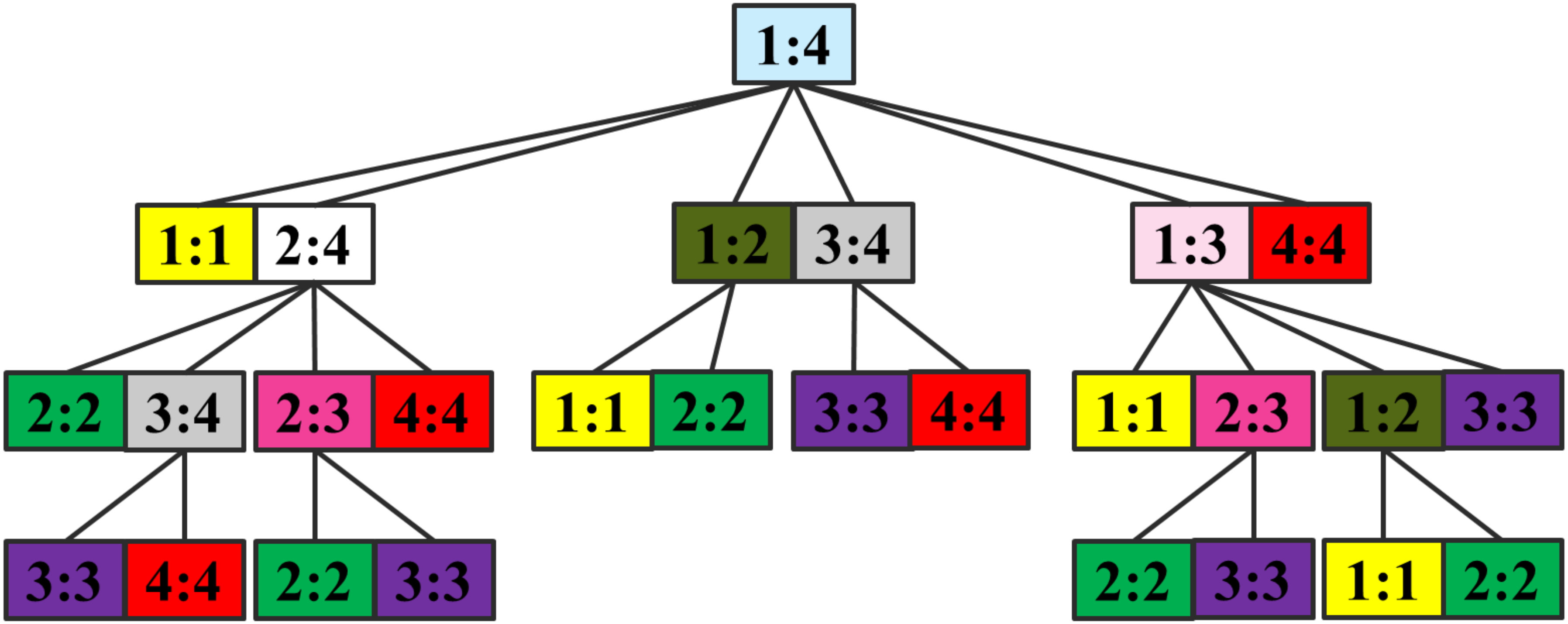
子问题的产生

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

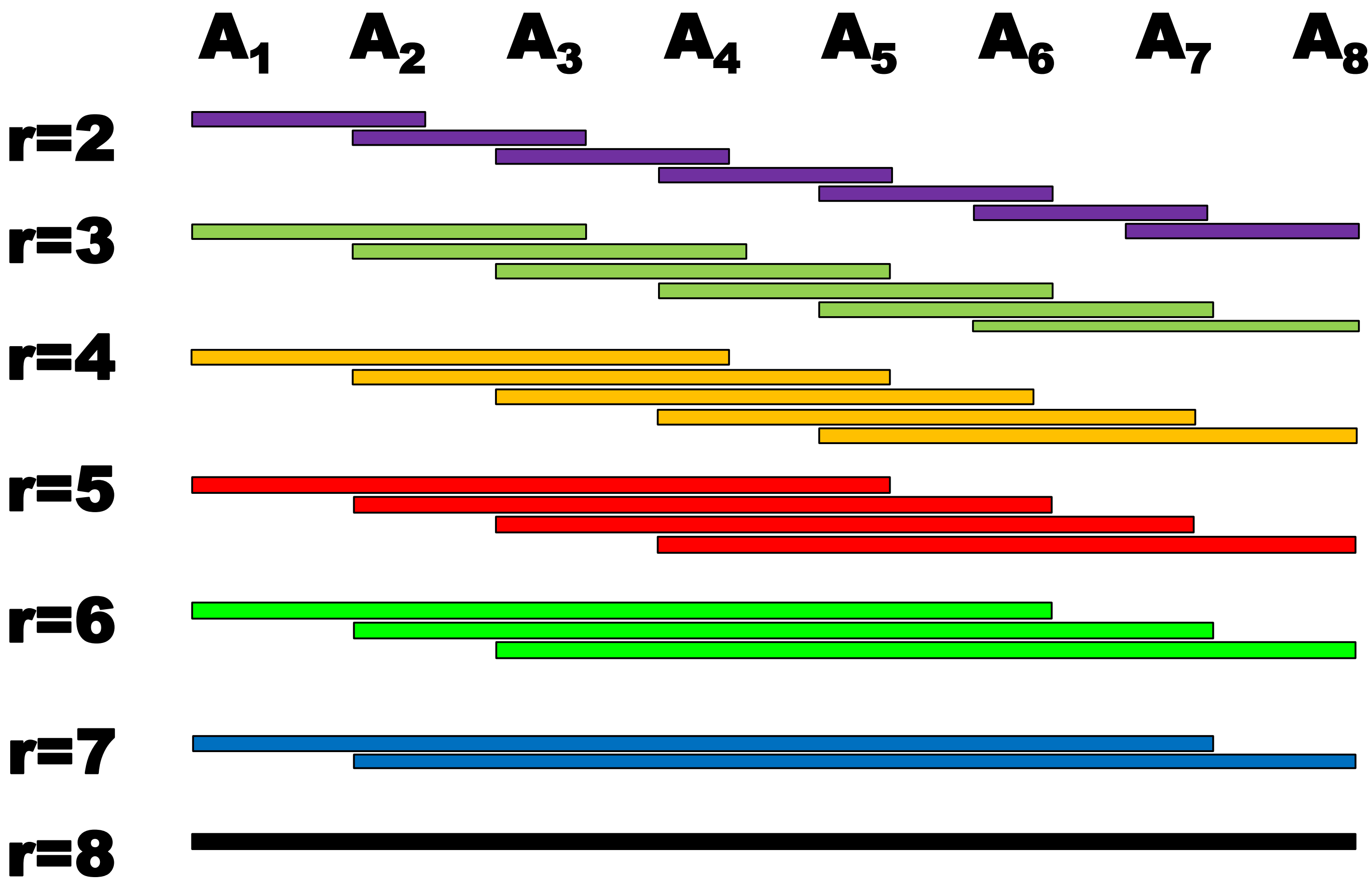
矩阵链乘法迭代顺序

- 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 $r \leftarrow 2$ **to** n **do** //r为链长

　for $i \leftarrow 1$ **to** $n-r+1$ **do** //左边界i

　　$j \leftarrow i+r-1$ //右边界j

　　$m[i,j] \leftarrow m[i+1,j]+P_{i-1}P_iP_j$ //$k=i$

　　$s[i,j] \leftarrow i$ //记录k

　　for $k \leftarrow i+1$ **to** $j-1$ **do** //遍历k

　　　$t \leftarrow m[i,k]+ m[k+1,j]+P_{i-1}P_kP_j$

　　　if $t < m[i,j]$

　　　then $m[i,j] \leftarrow t$ //更新解

　　　　$s[i,j] \leftarrow k$

遍历长度为r子问题

遍历所有划分

时间复杂度

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

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

- **根据m和s：**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
- **中间结果：** 存储所有子问题的最小乘法次数$m[i, j]$及得到这个值的划分位置$s[i, j]$

子问题最优解 $m[i, j]$

- 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					
r=3					
r=4					
r=5					

举例：如何计算$m[2, 5]=\min\begin{cases} m[2,2]+m[3,5]+P_1P_2P_5 \\ m[2,3]+m[4,5]+P_1P_3P_5 \\ m[2,4]+m[5,5]+P_1P_4P_5 \end{cases}$

子问题最优解 $m[i, j]$

- 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				

1. $m[2,5]= m[2,2]+ m[3,5]+ P_1P_2P_5 =$

$0+2500+35\times15\times20=13000$

子问题最优解 $m[i, j]$

- 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	$\boxed{\text{m[2,3]=2625}}$	m[3,4]=750	$\boxed{\text{m[4,5]=1000}}$	
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				

2. $m[2,5]= m[2,3]+ m[4,5]+ P_1P_3P_5 =$

2625+1000+35×5×20=7125

子问题最优解 *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				

3. $m[2,5]= m[2,4]+ m[5,5]+ P_1P_4P_5 =$

$4375+0+35\times10\times20=11375$

子问题最优解 $m[i, j]$

- 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				

$m[2,5]=\min\{13000, 7125, 11375\}=7125$

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

小结：动态规划的基本要素

一、最优子结构

- 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。
- 在分析问题的最优子结构性质时，通常可采用反证法。
- 利用问题的最优子结构性质，以自底向上的方式递归地从子问题的最优解逐步构造出整个问题的最优解。最优子结构是问题能用动态规划算法求解的前提。

同一个问题可以有多种方式刻划它的最优子结构，有些表示方法的求解速度更快(空间占用小，问题的维度低)。

小结：动态规划的基本要素

二、重叠子问题

- 递归算法求解问题时，每次产生的子问题并不总是新问题，有些子问题被反复计算多次。这种性质称为子问题的重叠性质。
- 动态规划算法，对每一个子问题只解一次，而后将其解保存在一个表格中，当再次需要解此子问题时，只是简单地用常数时间查看一下结果。
- 通常不同的子问题个数随问题的大小呈多项式增长。因此用动态规划算法只需要多项式时间，从而获得较高的解题效率。

基于备忘录的递归方法

回顾：基于递归的算法

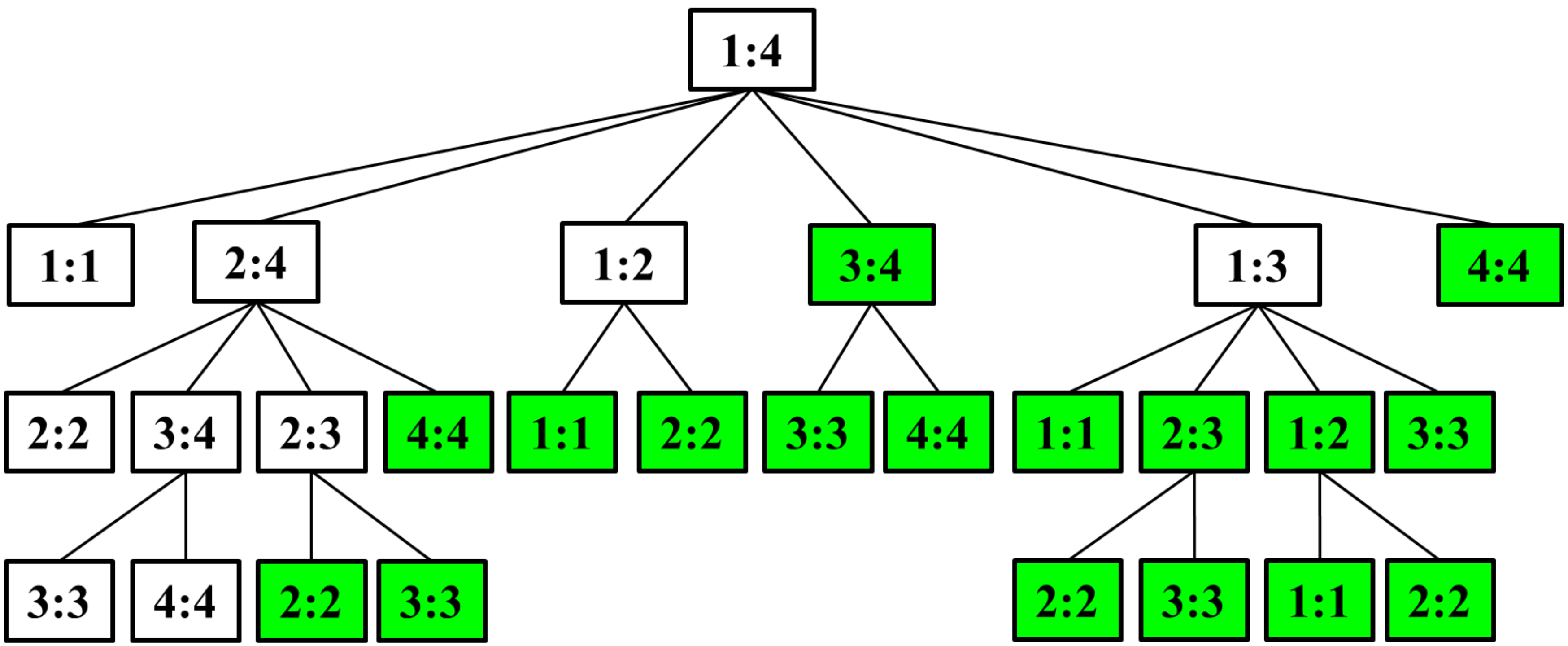
- **RecurMatrixChain(P, i, j)**

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) + 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** $m[i,j]$

递归算法的时间复杂度为$\Omega(2^n)$。如何改进？

动态规划算法的变形-备忘录法

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



动态规划算法的变形-备忘录法

备忘录方法

- 该方法的控制结构与直接递归方法的控制结构相同，区别在于备忘录方法为每个解过的子问题建立了备忘录以备需要时查看，避免了相同子问题的重复求解。

```
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;
}
```

动态规划法和备忘录法的异同

- 相同点：
  - 最优子结构
  - 重叠子问题
- 不同点
  - 动态规划：自底向上（从最小的问题开始解，不断填充子问题解的矩阵）
  - 备忘录：自顶向下（从最大的问题开始解，并查看子问题解的矩阵，若矩阵中有值，则无需额外计算）