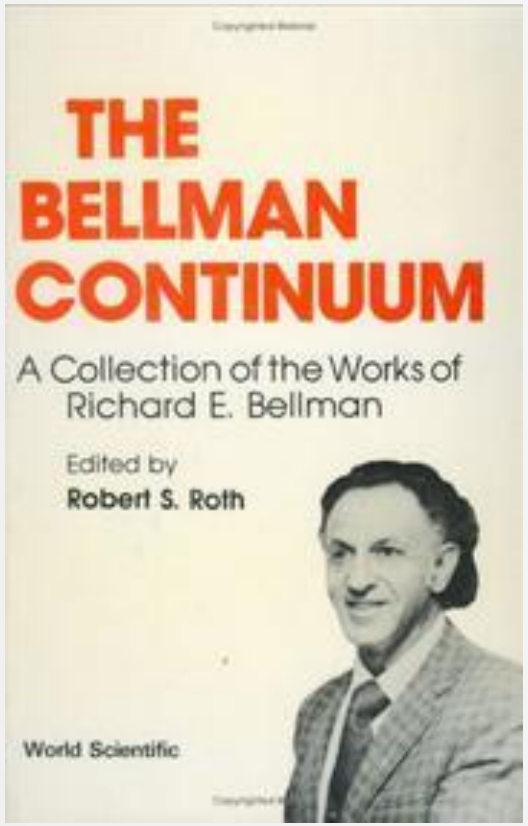
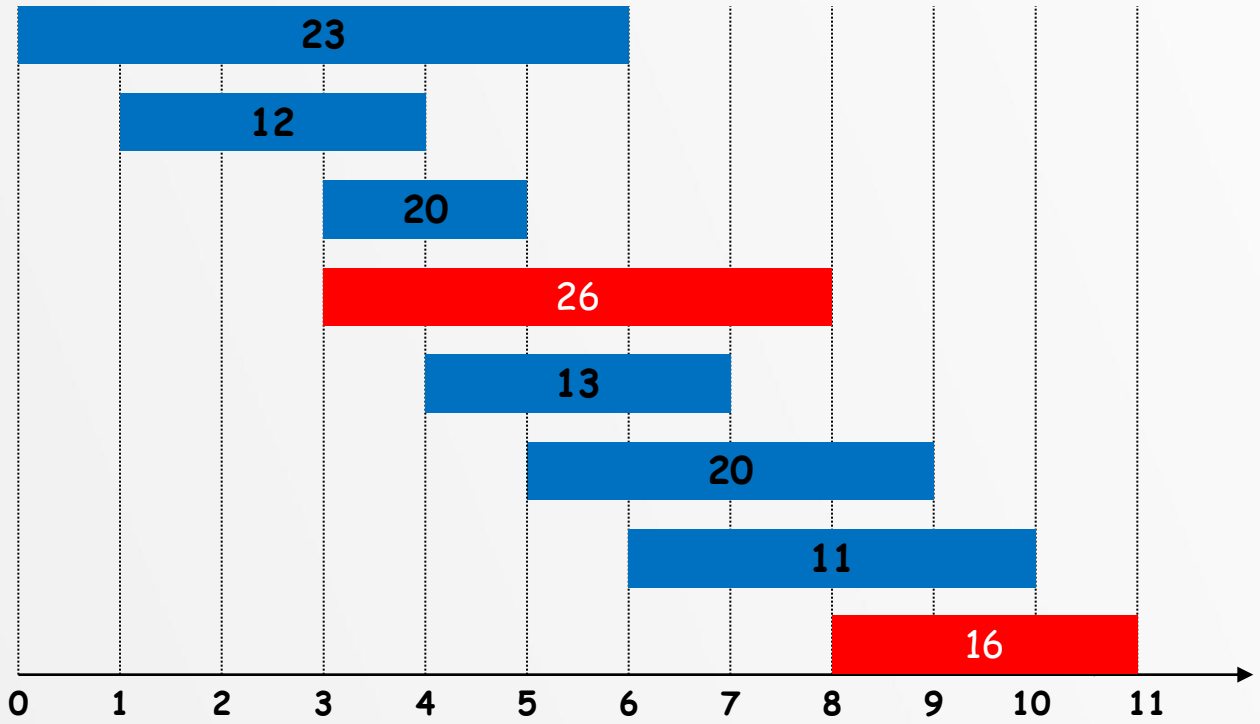


R.E.Bellman.

动态规划

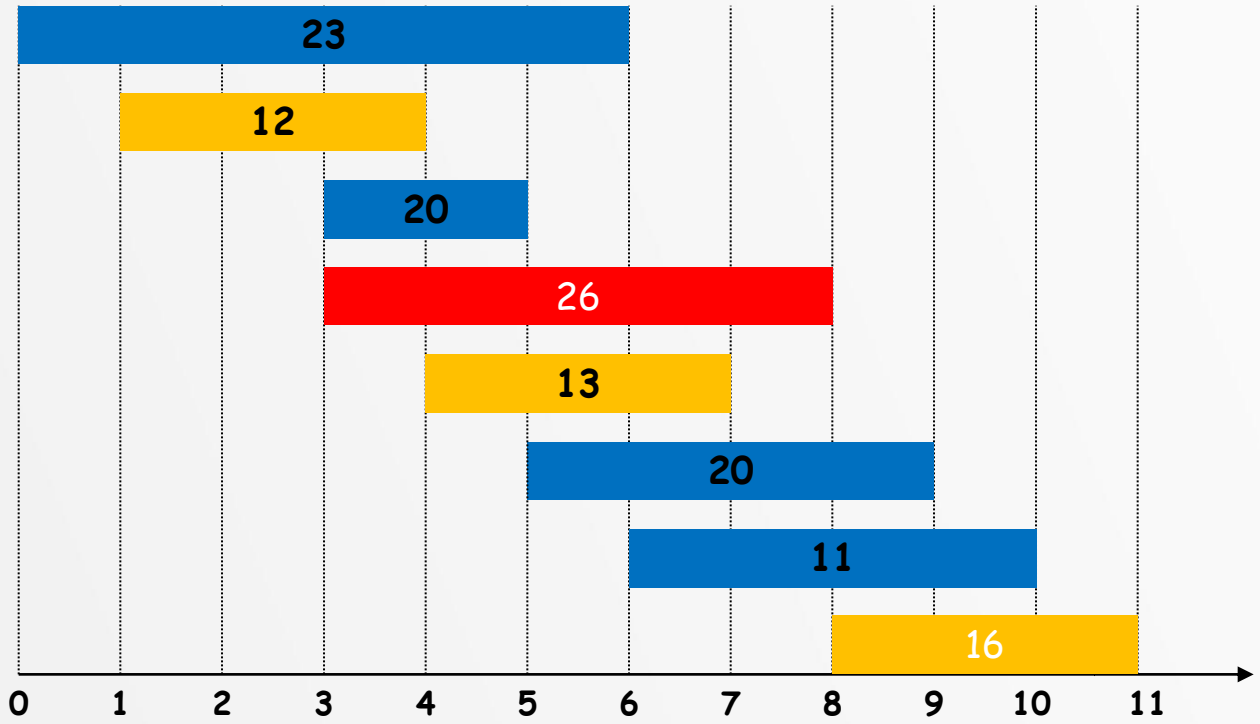
带权重的活动安排问题

- n个活动，活动 j 在 s_j 时刻开始， f_j时刻结束，活动j具有权重或价值 v_j .
- 目标: 找到具有最大价值和且互不重叠（兼容）的活动子集



带权重的活动安排问题

- 如果活动权重都为1，问题退化为活动安排问题，可用贪心算法得到全局最优解

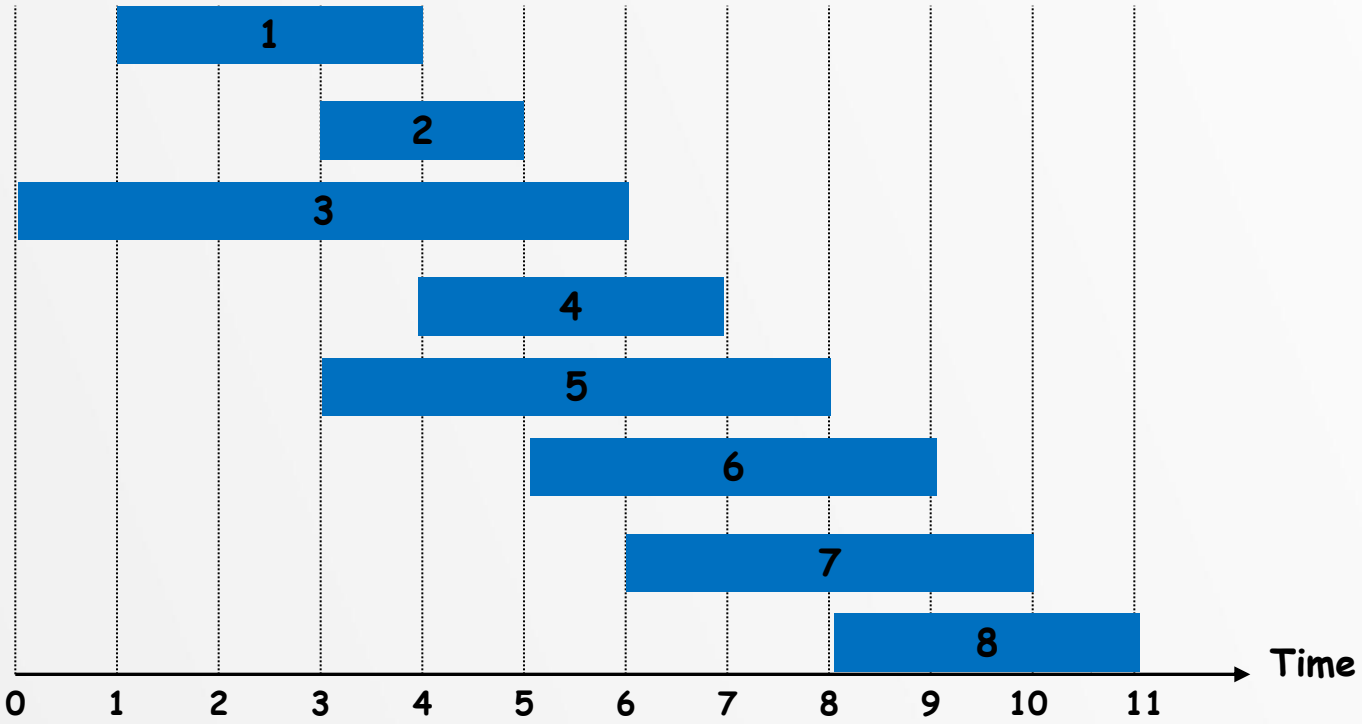


- 但权重不同，无法保证获得的是全局最优解，因此不能使用贪心算法

带权重的活动安排问题

- 将活动按结束时间从小到大进行排序 $f_1 \leq f_2 \leq \ldots \leq f_n$
- 定义 p(j) = 与活动 j 兼容的最大活动序号i，i < j

- 例如：p(8) = 5, p(7) = 3, p(2) = 0.



带权重的活动安排问题

- dp[i]的含义：活动1, 2, ..., i所能获得的最大价值（最优子结构与子问题）
  - **情况1：dp[i]中包含第i个活动，此时p**
    - 放弃不兼容的活动 { p(i) + 1, p(i) + 2, ..., i - 1 }
    - 必定包括剩余的1, 2, ..., p(i)个活动构成的最优解
  - **情况2：dp[i]中不包含第i个活动**
    - 必定包括活动1, 2, ..., i-1所形成的最优解
- 写出dp递推式
  - **dp[i] = max{ v_i + dp[p(i)], dp[i-1] }**

带权重的活动安排问题

- **dp[i] = max{ v_i + dp[p(i)], dp[i-1] }**
- 确定初始化条件dp[0] = 0， dp[1] = v1
- 遍历次序：先计算p(i)，然后从左向右
- 调试：输出dp数组

带权重的活动安排问题

- **dp[i] = max{ v_i + dp[p(i)], dp[i-1] }**

纯粹分治递归法求解

Input: n, s_1,…,s_n , f_1,…,f_n , v_1,…,v_n

Sort jobs by finish times so that $f_1 \le f_2 \le \ldots \le f_n$.
Compute p(1), p(2), …, p(n);
return M-Compute-Opt(n);;

```
M-Compute-Opt(n)
{
   if (n=0)
       return 0;
   else
       return max(v_n + M-Compute-Opt(p(n)), M-Compute-Opt(n-1));
}
```

指数时间算法
递归函数调用次数
O(1.618^n)

带权重的活动安排问题

- **dp[i] = max{ v_i + dp[p(i)], dp[i-1] }**

添加备忘录：存储每个子问题的解，当需要时通过查询获得子问题答案

Input: n, s_1,…,s_n , f_1,…,f_n , v_1,…,v_n

Sort jobs by finish times so that $f_1 \le f_2 \le \ldots \le f_n$.
Compute p(1), p(2), …, p(n)

```
for j = 1 to n
   M[j] = empty
M[0] = 0

M-Compute-Opt(n) {
   if (M[n] is empty)
      M[n] = max(v_n + M-Compute-Opt(p(n)), M-Compute-Opt(n-1))
   return M[n]
}
```

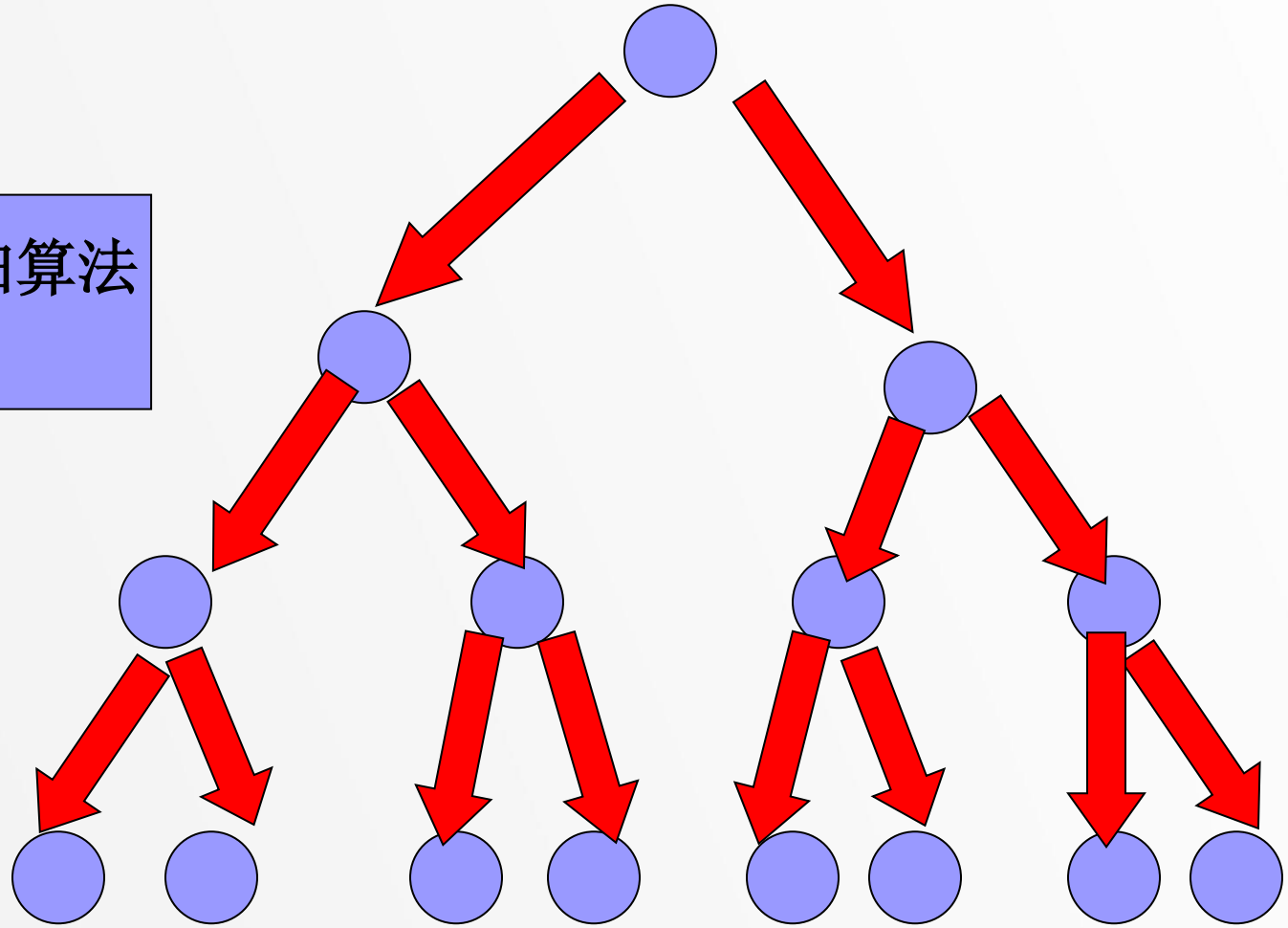
把计算过的子问题的解保存起来；
递归函数调用次数O(n)

带权重的活动安排问题

- dp[i] = max{ v_i + dp[p(i)], dp[i-1] }

添加备忘录：存储每个子问题的解，当需要时通过查询获得子问题答案

备忘录方法：带记忆的分治递归算法
本质是自顶向下的



带权重的活动安排问题

- **dp[i] = max{ v_i + dp[p(i)], dp[i-1] }**

动态规划求解

Input: n, s_1,…,s_n , f_1,…,f_n , v_1,…,v_n

Sort jobs by finish times so that $f_1 \le f_2 \le \ldots \le f_n$.

Compute p(1), p(2), …, p(n)

最小子问题

```
M[0] = 0
   for j = 1 to n
      M[j] = max(v_j + M[p(j)], M[j-1])
}
```

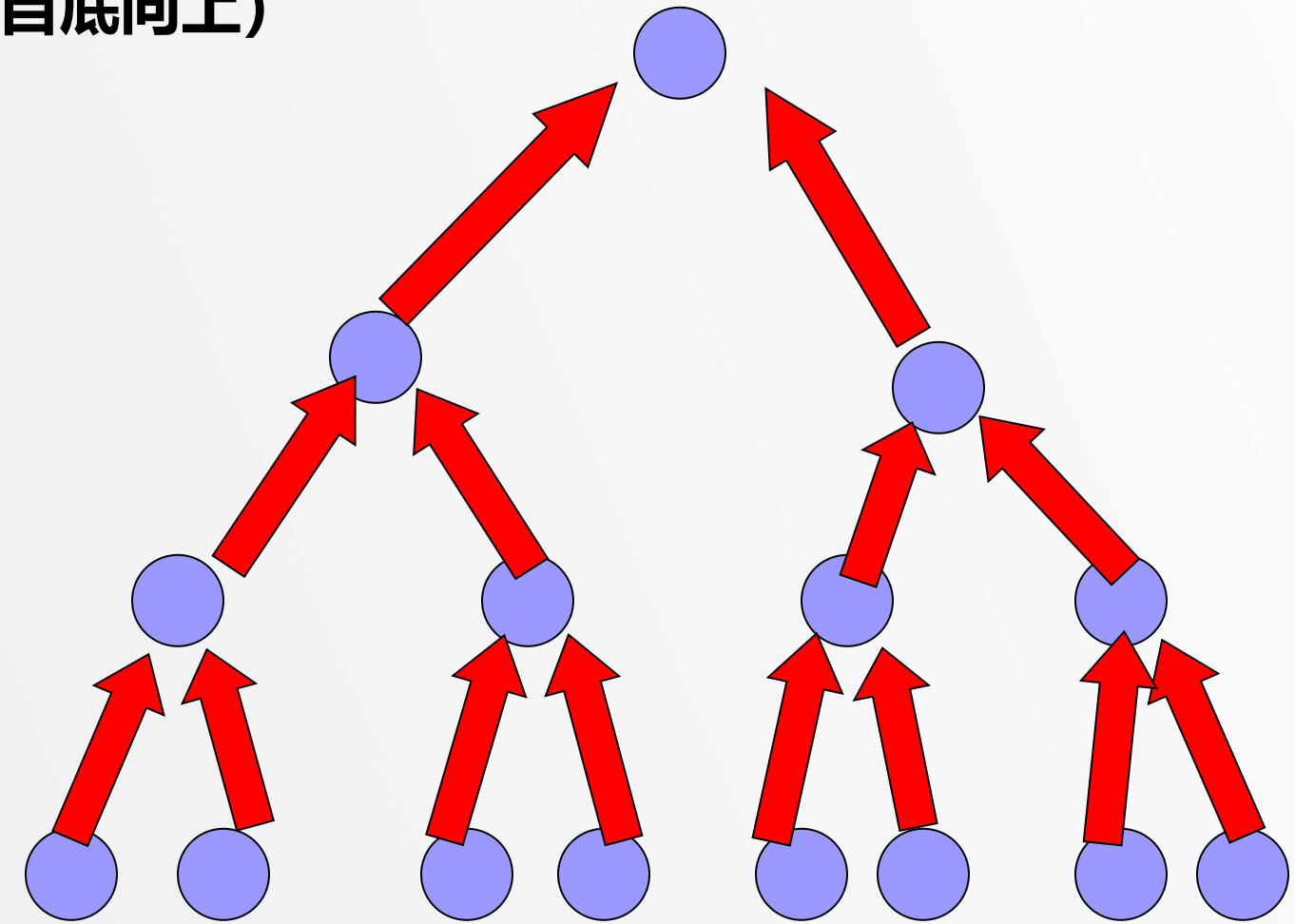
问题规模扩大

时间复杂度O(n log n) +O(n) = O(n log n)

带权重的活动安排问题

- dp[i] = max{ v_i + dp[p(i)], dp[i-1] }

动态规划求解（自底向上）



带权重的活动安排问题

给定8个候选活动的开始时间、结束时间和权重，请根据递归式求解这个具体的活动安排问题，要求给出求解过程表格、最优解和最优值。

j	1	2	3	4	5	6	7	8
s(j)	1	3	0	4	3	5	6	8
f(j)	4	5	6	7	8	9	10	11
v(j)	12	20	23	13	26	20	11	16

1. 依据结束时间排序， 2. 求p(i)，

3. dp[i] = max{ v_i + dp[p(i)], dp[i-1] }， dp[0] = 0

j	1	2	3	4	5	6	7	8
p(j)	0	0	0	1	0	2	3	5
OPT(j)	12	20	23	25	26	40	40	42

选择活动5和活动8，能够获得最大权重和42

带权重的活动安排问题

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
s(j)	1	2	0	7	3	9	11	10	16	15	21	17	14	25	29	20
t(j)	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34

思考：是否带有权重?

带权重的活动安排问题

某知名理发师晓华在网上爆火后收到源源不断的发型设计预约请求，每个预约j都有开始时间s(j)和结束时间t(j)。对于每个预约，晓华都可以选择接或不接，但是她接的任意两个预约的时间不能重叠，因为任意时刻都只能为至多一位顾客提供服务。给定一个预约请求序列如下，设计算法替理发师找到最优的预约集合（总工时最长），并给出得到的最大工时时长。要求给出求解过程表格。

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
s(j)	1	2	0	7	3	9	11	10	16	15	21	17	14	25	29	20
t(j)	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34

矩阵连乘问题

- 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

$$A_1 A_2 \ldots A_n$$

- 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。
- 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

矩阵连乘问题

- 对于 *p*q* 矩阵 *A* 和一个 *q*r* 矩阵 *B*, *AB* 需要多少次标准乘法计算？

$$c_{ij}=\sum_{k=1}^{q}a_{ik}b_{kj}$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1r} \\ c_{21} & c_{22} & \cdots & c_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{r2} & \cdots & c_{pr} \end{bmatrix}=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1q} \\ a_{21} & a_{22} & \cdots & a_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ a_{p1} & a_{p2} & \cdots & a_{pq} \end{bmatrix}\times\begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1r} \\ b_{21} & b_{22} & \cdots & b_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ b_{q1} & b_{q2} & \cdots & b_{qr} \end{bmatrix}$$

pqr次标准乘法

矩阵连乘问题

- 设有四个矩阵 $A, B, C, D,$ 它们的维数分别是:

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

- 总共有五中完全加括号的方式

$$(A((BC)D)) \quad (A(B(CD))) \quad ((AB)(CD))$$
$$(((AB)C)D) \quad ((A(BC))D)$$

16000, 10500, 36000, 87500, 34500

如果 A_1, A_2, and A_3 是 20X100, 100X10, 和 10X50 矩阵,

$A_1XA_2XA_3$乘积运算次数是多少?

20 X 10 X 50 = 10000

((A_1XA_2) X A_3)

20 X 100 X 10 = 20000

30000 次运算

100X 10 X 50 = 50000

(A_1 X (A_2 X A_3))

20 X 100 X 50 = 100000

150000 次运算

矩阵连乘问题

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，$i=1,2,\ldots,n-1$。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**。

穷举法求解思路

◆**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。

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：
$((A_1 \ldots A_k)(A_k+1 \ldots A_n))$可以得到关于P(n)的递推式如下：

$$P(n)=\begin{cases}1 & n=1\\ \sum_{k=1}^{n-1}P(k)P(n-k) & n>1\end{cases} \Rightarrow P(n)=\frac{1}{n}C_{n-1}^{2(n-1)}=\Omega(4^n/n^{3/2})$$

Catalan 数 $P(n) = C(n-1)$

矩阵连乘问题

- 最优解结构分析（明确dp下标及含义）

将矩阵连乘积 $A_iA_{i+1}...A_j$简记为A[i:j]，这里i≤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≤k<j，则其相应完全加括号方式为 $(A_iA_{i+1}...A_k)(A_{k+1}A_{k+2}...A_j)$

总计算量=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

分析最优解的子问题结构

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。

分析最优解的子问题结构

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

子问题不独立，
适合动态规划算法设计

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。

建立递归关系（推导dp关系式）

- 设计算A[i:j]，1≤i≤j≤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,...,n
- 当i<j时，

$$m[i,j]=\min_k\{m[i,k]+m[k+1,j]+p_{i-1}p_kp_j\}$$

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：

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

k 的位置只有 $j-i$ 种可能

实际子问题数目

- 对于1≤i≤j≤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

带备忘录的分治递归算法

- 备忘录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。

```
m←0
private static int lookupChain(int i, int j)
  {
    if (m[i][j] > 0) return m[i][j];
    if (i == j) return 0;
    int u = 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;
  }
```

已经计算过直接返回

以递归方式进行计算

计算复杂度：$O(n^3)$

用动态规划法求最优解

```
public static void matrixChain(int [] p, int [][] m, int [][] s)
  {
     int n=p.length-1;
     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;}
              }
           }
  }
```

用动态规划法求最优解

```
public static void matrixChain(int [] p, int [][] m, int [][] s)
  {
     int n=p.length-1;
     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;}
              }
           }
  }
```

m[i,i]

m[i,i+r]

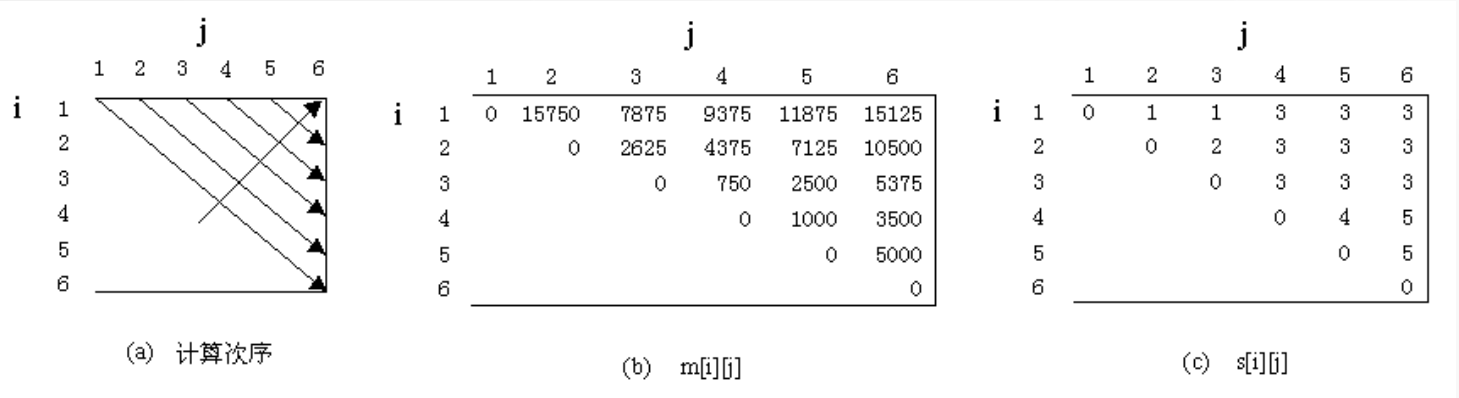
算法复杂度分析：
算法matrixChain的主要计算量取决于算法中对r，i和k的3重循环。循环体内的计算量为O(1)，而3重循环的总次数为O(n3)。因此算法的计算时间上界为O(n3)。算法所占用的空间显然为O(n2)。

用动态规划法求最优解

例子：下表6个矩阵连乘问题

A1	A2	A3	A4	A5	A6
30×35	35×15	15×5	5×10	10×20	20×25

$$m[2][5]=\min\begin{cases} m[2][2]+m[3][5]+p_1p_2p_5=0+2500+35\times15\times20=13000 \\ m[2][3]+m[4][5]+p_1p_3p_5=2625+1000+35\times5\times20=7125 \\ m[2][4]+m[5][5]+p_1p_4p_5=4375+0+35\times10\times20=11375 \end{cases}$$



(a) 计算次序

i \ j	1	2	3	4	5	6
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]

i \ j	1	2	3	4	5	6
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]

练习

A1	A2	A3	A4	A5
3×7	7×8	8×5	5×12	12×10

	1	2	3	4	5
1	0	168	288	468	828
2		0	280	700	1230
3			0	480	1000
4				0	600
5					0

	1	2	3	4	5
1	0	1	2	3	4
2		0	2	3	3
3			0	3	3
4				0	4
5					0

最少乘法次数为828，

相应的最优加括号方式为 $(((A_1A_2)A_3)A_4)A_5$