课程回顾

- 递归式（举例：阶乘、斐波那契数列、Ackermann函数、*n*阶Hanoi塔问题、全排列问题）
- 递归式和分治法关系
- 递归式求解（代入法、递归树法、主方法）
- 分治法求解适用条件
  - 小规模问题易解、可分解为规模较小的相同子问题、子问题解可合并、子问题相互独立
- 分治法求解举例：
  - 归并排序
  - 二分搜索
  - 最大子数组问题
  - 矩阵乘法的Strassen算法
  - 大整数乘法
  - 快速排序

第3章 动态规划

苏州大学 计算机科学与技术学院
汪笑宇
Email: xywang21@suda.edu.cn

本章内容

■动态规划（教材Chapter 15）

- 动态规划概述：算法步骤、两种实现方法
- 动态规划原理
- 动态规划求解实例：
  - 钢条切割
  - 矩阵链乘法的最优括号化
  - 多边形的最佳三角剖分
  - 最长公共子序列
  - 最优二叉搜索树
  - 0-1背包

动态规划概述

- 动态规划（Dynamic Programming）作为一种使**多阶段决策过程最优**的通用方法，在20世纪50年代由美国数学家理查德·贝尔曼提出。
- 动态规划是在应用数学中用来解决某类最优问题的重要工具，而且在计算机领域中被当做一种通用的算法设计技术来使用。

贝尔曼，R.

动态规划概述 (续)

- 动态规划主要用于优化问题求解，即求出问题的最优(最大/小)解，当有多个最优解时一般求一个即可
- 与分治法对比（教材p204）
  - 分治法：将问题划分为互不相交的子问题，递归地求解子问题，再将它们的解组合起来求出原问题的解
  - 动态规划：应用于子问题重叠的情况，即不同的子问题具有公共的子子问题

动态规划概述 (续)

- 当分解问题非独立时，即它们共享子子问题时，可采用动态规划
- 此时分治法将重复地解这些共同的子子问题，形成重复计算，而动态规划对每一子子问题只做一次计算，然后将答案存储在一表中（这就是programming含义，像节目单一样），故可避免重复计算
- 回忆斐波那契数递归算法

动态规划概述 (续)

- 设计动态规划算法步骤：
  1. 刻画一个最优解的结构特征
  2. 递归地定义最优解的值
  3. 计算最优解的值，通常采用自底向上的方法
  4. 利用计算出的信息构造一个最优解
- 步骤1-3是基础
- 有时需要执行步骤3的过程中维护一些额外信息，以便用来构造一个最优解

动态规划概述 (续)

- 两种等价的实现方法（教材p207）：
  - 带备忘的自顶向下法 (top-down with memoization)：按自然的递归形式编写过程，过程中保存每个子问题的解。当需要一个子问题的解时，首先检查是否已经保存过此解，若是则直接返回保存的值，否则按通常方式计算这个子问题
  - 自底向上法 (bottom-up method)：将子问题按规模排序，按由小至大的顺序进行求解。当求解某个子问题时，它所依赖的更小的子问题都已求解完毕，结果已经保存
- 由于没有频繁的递归函数调用开销，自底向上方法的时间复杂性函数通常具有更小的系数

动态规划问题

- 钢条切割
- 矩阵链乘法的最优括号化
- 多边形的最佳三角剖分
- 最长公共子序列
- 最优二叉搜索树
- 0-1背包

动态规划问题

- ■钢条切割
- ■矩阵链乘法的最优括号化
- ■多边形的最佳三角剖分
- ■最长公共子序列
- ■最优二叉搜索树
- ■0-1背包

钢条切割问题

■Serling公司购买长钢条将其切割为短钢条出售

■切割工序本身没有成本

■出售长度为i英寸的钢条价格为p_i (i=1, 2, ..., 10)美元

长度i	1	2	3	4	5	6	7	8	9	10
价格p_i	1	5	8	9	10	17	17	20	24	30

■钢条切割问题定义：给定一段长度为n英寸的钢条和一个价格表p_i (i=1, 2, ..., n)，求切割钢条方案，使得销售收益r_n最大

钢条切割问题 (续)

长度i	1	2	3	4	5	6	7	8	9	10
价格p_i	1	5	8	9	10	17	17	20	24	30

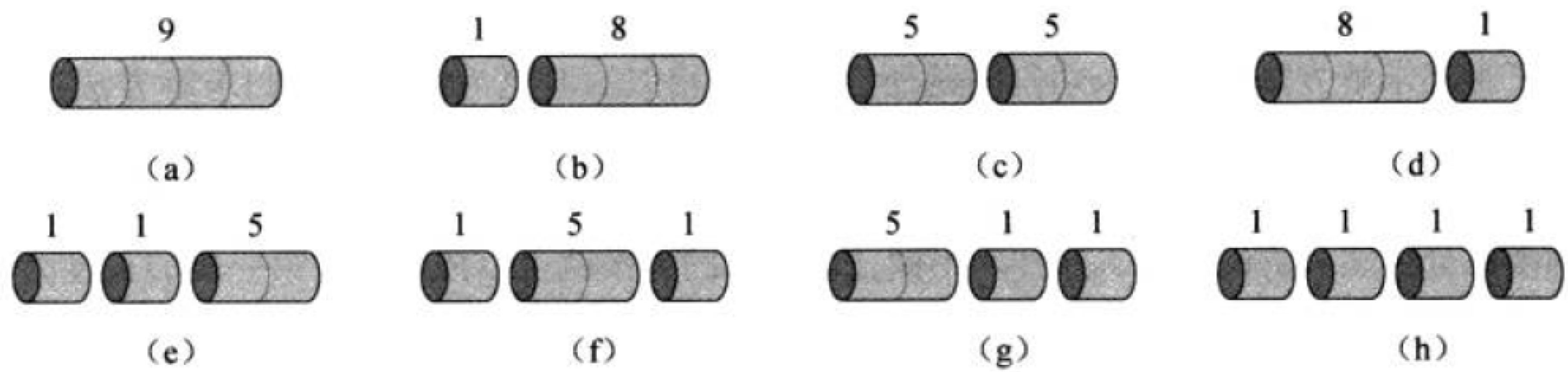
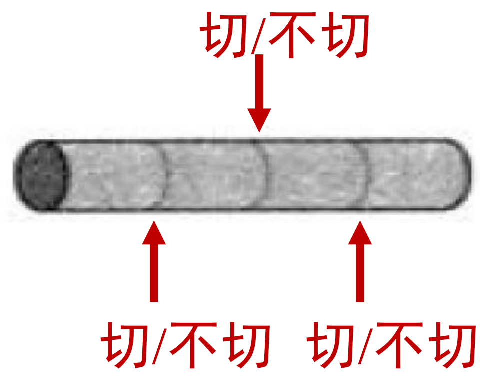


图 15-2 4 英寸钢条的 8 种切割方案。根据图 15-1 中的价格表，在每段钢条之上标记了它的价格。最优策略为方案(c)——将钢条切割为两段长度均为 2 英寸的钢条——总价值为 10

钢条切割问题 (续)

■长度为n的钢条切割方案共有2^{n-1}种



■设一个最优解将钢条切割为k段（$1\le k\le n$），最优切割方案：$n=i_1+i_2+\ldots+i_k$，最大收益为 $r_n=p_{i_1}+p_{i_2}+\ldots+p_{i_k}$

■$r_n=\max(p_n, r_1+r_{n-1}, r_2+r_{n-2}, \ldots, r_{n-1}+r_1)$

钢条切割问题 (续)

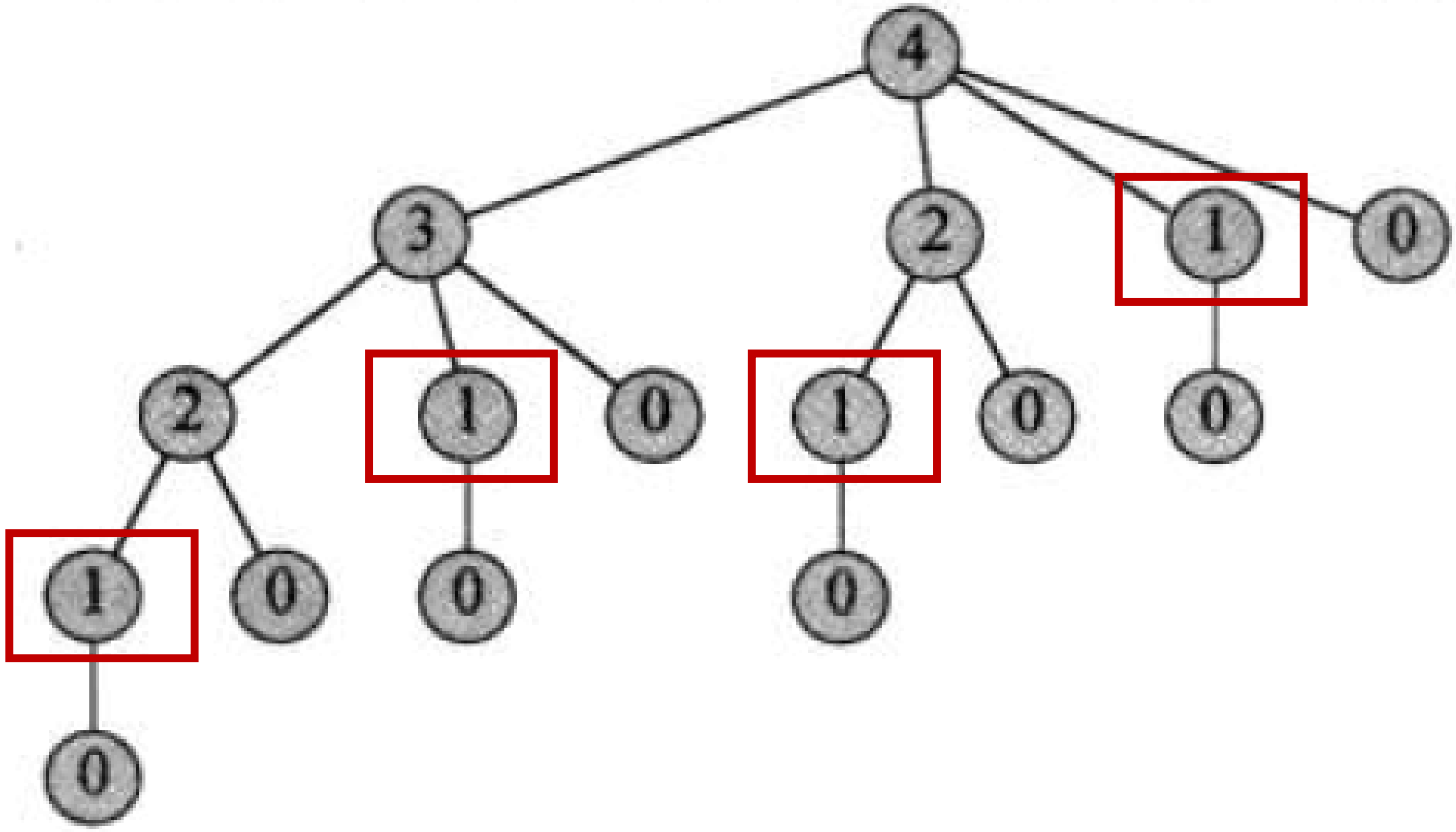
■最优子结构：问题的最优解由相关子问题的最优解组合而成，而这些子问题可以独立求解

■简化求解：从左边切割下长度为*i*的一段不再继续切割，只对右边剩下的*n*-*i*段继续切割（递归求解）：$r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$

■自顶向下递归：

```
CUT_ROD(p, n)
if n = 0
    return 0
q ← -∞
for i ← 1 to n do
    q ← max(q, p[i] + CUT_ROD(p, n-i))
return q
```

钢条切割问题 (续)



钢条切割问题 (续)

```
CUT_ROD(p, n)
if n = 0
    return 0
q ← -∞
for i ← 1 to n do
    q ← max(q, p[i] + CUT_ROD(p, n-i))
return q
```

■时间复杂度分析：

$$T(n) = 1 + \sum_{i=1}^{n} T(n-i)$$

$$= 1 + T(n-1) + T(n-2) + \ldots + T(0)$$

$$T(n-1) = 1 + T(n-2) + T(n-3) + \ldots + T(0)$$

$$T(n) - T(n-1) = T(n-1) \;\Rightarrow\; T(n) = 2T(n-1)$$

$$\Rightarrow\; T(n) = 2^n$$

指数时间复杂度！

钢条切割问题 (续)

■带备忘的自顶向下法：

```
MEMOIZED_CUT_ROD(p, n)
let r[0..n] be a new array
for i ← 0 to n do
    r[i] ← -∞
return MEMOIZED_CUT_ROD_AUX(p, n, r)
```

```
MEMOIZED_CUT_ROD_AUX(p, n, r)
if r[n] ≥ 0
    return r[n]
if n = 0
    q ← 0
else q ← -∞
    for i ← 1 to n
        q ← max(q, p[i] + MEMOIZED_CUT_ROD_AUX(p, n-i, r))
r[n] ← q
return q
```

钢条切割问题 (续)

■自底向上法：

```
BOTTOM_UP_CUT_ROD(p, n)
let r[0..n] be a new array
r[0] ← 0
for j ← 1 to n do
    q ← -∞
    for i ← 1 to j do
        q ← max(q, p[i] + r[j-i])
    r[j] ← q
return r[n]
```

➢依次求解规模为$j = 0, 1, ..., n$的子问题

钢条切割问题 (续)

- 重构解：输出最大收益及切割方案
- 对长度为j的钢条计算最大收益r_j及第一段钢条切割长度s_j

```
EXTENDED_BOTTOM_UP_CUT_ROD(p, n)
let r[0..n] be a new array
r[0] ← 0
for j ← 1 to n do
    q ← -∞
    for i ← 1 to j do
        if q < p[i] + r[j-i]
            q ← p[i] + r[j-i]
            s[j] ← i
        r[j] ← q
return r and s
```

钢条切割问题 (续)

■输出最优切割方案

```
PRINT_CUT_ROD_SOLUTION(p, n)
(r, s) ← EXTENDED_BOTTOM_UP_CUT_ROD(p, n)
while n > 0 do
    print s[n]
    n ← n – s[n]
```

动态规划问题

- ■钢条切割
- ■矩阵链乘法的最优括号化
- ■多边形的最佳三角剖分
- ■最长公共子序列
- ■最优二叉搜索树
- ■0-1背包

矩阵链乘法

- 给定n个矩阵的序列（矩阵链）$<A_1, A_2, \ldots, A_n>$，计算乘积$A_1A_2\cdots A_n$
- 计算多个矩阵连乘积可用括号来决定计算次序，每一个括号内的矩阵相乘调用标准的矩阵乘法
- 矩阵积的完全括号化
  - 它是单一矩阵
  - 或者是两个完全括号化的矩阵链的积

递归定义
计算次序无二义性

$(A_1(A_2(A_3A_4)))$

$(A_1((A_2A_3)A_4))$

$((A_1A_2)(A_3A_4))$

$((A_1(A_2A_3))A_4)$

$(((A_1A_2)A_3)A_4)$

矩阵链乘法 (续)

■不同的括号化方式产生不同的计算成本

MATRIX_MULTIPLY(A, B)
1 **if** A.*columns* $\neq$ B.*rows*
2   **error** “incompatible dimensions”
3 **else** let C be a new A.*rows*×B.*columns* matrix
4   **for** $i \leftarrow 1$ **to** A.*rows* **do**
5   **for** $j \leftarrow 1$ **to** B.*columns* **do**
6             $c_{ij} \leftarrow 0$
7   **for** $k \leftarrow 1$ **to** A.*columns* **do**
8                 $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$
9 **return** C

- 第8行执行次数：A.*rows*×B.*columns*×A.*columns*(或B.*rows*)
- 设A是$p \times q$矩阵、B是$q \times r$矩阵，则计算C=A·B共需pqr次标量乘法

矩阵链乘法 (续)

- 不同的括号化方式产生不同的计算成本
  - 例：以矩阵链$<A_1, A_2, A_3>$相乘为例，三个矩阵规模分别为10×100、100×5、5×50
  - $(A_1A_2)A_3$：A_1A_2：10×100×5=5000, 得到10×5矩阵

    $(A_1A_2)A_3$：10×5×50=2500

    共计5000+2500=7500次标量乘法
  - $A_1(A_2A_3)$：A_2A_3：100×5×50=25000, 得到100×50矩阵

    $A_1(A_2A_3)$：10×100×50=50000

    共计25000+50000=75000次标量乘法

相差10倍！

矩阵链乘法 (续)

- 矩阵链乘法问题实质上是一个最优括号化问题：
  - 给定n个矩阵的链<$A_1, A_2, \ldots, A_n$>，矩阵A_i的规模为$p_{i-1}\times p_i$ $(1\le i\le n)$，求完全括号化方案，使得计算乘积$A_1A_2\cdots A_n$所需标量乘法次数最少
  - 计算括号化方案数量：设$P(n)$表示一个n个矩阵的链中可选括号化方案数量，则穷举法产生数量：

$$P(n)=\begin{cases}1, & n=1,\\ \displaystyle\sum_{k=1}^{n-1}P(k)P(n-k), & n\ge 2.\end{cases}$$

Catalan数，指数阶：$\Omega(4^n/n^{1.5})$，不如直接求解矩阵乘积！

矩阵链乘法 (续)

■应用动态规划方法可获得多项式时间求解方法

1. 刻画一个最优解的结构特征
2. 递归地定义最优解的值
3. 计算最优解的值，通常采用自底向上的方法
4. 利用计算出的信息构造一个最优解

矩阵链乘法 (续)

1. 刻画一个最优解的结构特征——最优括号化方案的结构特征

- ➢$A_{i..j}$ $(1\le i\le j\le n)$：$A_iA_{i+1}\cdots A_j$
- ➢设$A_iA_{i+1}\ldots A_j$的最优括号化是在A_k和A_{k+1}之间划分开 $(i\le k<j$且$i<j)$
- ➢对某个k，先计算$A_{i..k}$和$A_{k+1..j}$，再计算两者乘积得到$A_{i..j}$
- ➢计算代价：

  $A_{i..k}$计算代价 + $A_{k+1..j}$计算代价 + 两者乘积计算代价

矩阵链乘法 (续)

- 关键：$A_iA_{i+1}\cdots A_j$的最优括号化亦要求分割开的两个子链$A_{i..k}$和$A_{k+1..j}$是最优括号化。
- 可用反证法证明：若$A_{i..k}$括号化不是最优，则可找到一个成本更小的方法将其括号化，代入到$A_{i..j}$的最优括号化表示中，得到的计算成本比最优解小，矛盾！

矩阵链乘法 (续)

2. 递归地定义最优解的值

- 怎样用子问题的最优解递归地定义原问题的最优解(一般是最优解的值)?
- 子问题：对所有的$1 \le i \le j \le n$确定$A_{i..j}$最优括号化代价
- $m[i, j]$：计算$A_{i..j}$所需标量乘法次数的最小值
  - 若$i=j$，矩阵链只有A_i，无需乘法，$m[i, i]=0$
  - 若$i<j$，利用步骤1最优子结构计算代价（k是最优分割点）：
    $A_{i..k}$计算代价 + $A_{k+1..j}$计算代价 + 两者乘积计算代价
    即：$m[i, j] = m[i, k] + m[k+1, j] + p_{i-1}p_kp_j$
- 原问题最优解：计算$A_{1..n}$所需的最小代价为$m[1, n]$

矩阵链乘法 (续)

- $m[i, j]$：计算$A_{i..j}$所需标量乘法次数的最小值
  - 若$i<j$，利用步骤1最优子结构计算代价（k是最优分割点）：
    $A_{i..k}$计算代价 + $A_{k+1..j}$计算代价 + 两者乘积计算代价
    即：$m[i, j] = m[i, k] + m[k+1, j] + p_{i-1}p_kp_j$
  - 以上公式成立需要k是最优分割点
  - 检查所有$j-i$种可能的k即可：

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

  - 定义$s[i, j]$保存$A_{i..j}$最优括号化方案分割点位置k

矩阵链乘法 (续)

- 例：求矩阵链<A_1, A_2, A_3, A_4, A_5>乘积，在k=3处分割：
  - ($A_1\ A_2\ A_3\ A_4\ A_5$)

$$
\begin{array}{ccccc}
(A_1 & A_2 & A_3) & (A_4 & A_5) \\
p_0 \times p_1 & p_1 \times p_2 & p_2 \times p_3 & p_3 \times p_4 & p_4 \times p_5 \\
 & \Downarrow & & \Downarrow & \\
 & p_0 \times p_3 & & p_3 \times p_5 &
\end{array}
$$

  - 标量乘法次数：$m[1, 5] = m[1, 3] + m[4, 5] + p_0 p_3 p_5$
    $m[i, j] = m[i, k] + m[k+1, j] + p_{i-1} p_k p_j$

矩阵链乘法 (续)

3. 计算最优解的值，通常采用自底向上的方法

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

- 递归算法为指数时间复杂度：递归调用树不同分支中多次计算同一个子问题
- 需求解的不同子问题数目为n^2阶：$C_n^2 + n = \Theta(n^2)$
  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
- 矩阵A_i规模为$p_{i-1} \times p_i$ ($i=1,2,\ldots,n$)，辅助表$m[1..n, 1..n]$保存代价$m[i, j]$，辅助表$s[1..n-1, 2..n]$记录$m[i, j]$对应的最优分割点k

矩阵链乘法 (续)

■例：计算矩阵链$\langle A_1, A_2, A_3, A_4\rangle$乘积。以下计算所有$m[i, j]$的值，其中$1 \le i \le j \le n$

- $i = j$：$m[1, 1] = m[2, 2] = m[3, 3] = m[4, 4] = 0$
- $i < j$：$m[1, 2]$：k=1时为$m[1, 1] + m[2, 2] + p_0p_1p_2$

  $m[2, 3]$：k=2时为$m[2, 2] + m[3, 3] + p_1p_2p_3$

  $m[3, 4]$：k=3时为$m[3, 3] + m[4, 4] + p_2p_3p_4$

  $m[1, 3]$：k=1时为$m[1, 1] + m[2, 3] + p_0p_1p_3$

  k=2时为$m[1, 2] + m[3, 3] + p_0p_2p_3$

  $m[2, 4]$：k=2时为$m[2, 2] + m[3, 4] + p_1p_2p_4$

  k=3时为$m[2, 3] + m[4, 4] + p_1p_3p_4$

矩阵链乘法 (续)

- 例：计算矩阵链<A_1, A_2, A_3, A_4>乘积。以下计算所有$m[i, j]$的值，其中$1 \le i \le j \le n$
  - $i = j$：$m[1, 1] = m[2, 2] = m[3, 3] = m[4, 4] = 0$
  - $i < j$：$m[1, 2], m[2, 3], m[3, 4]$

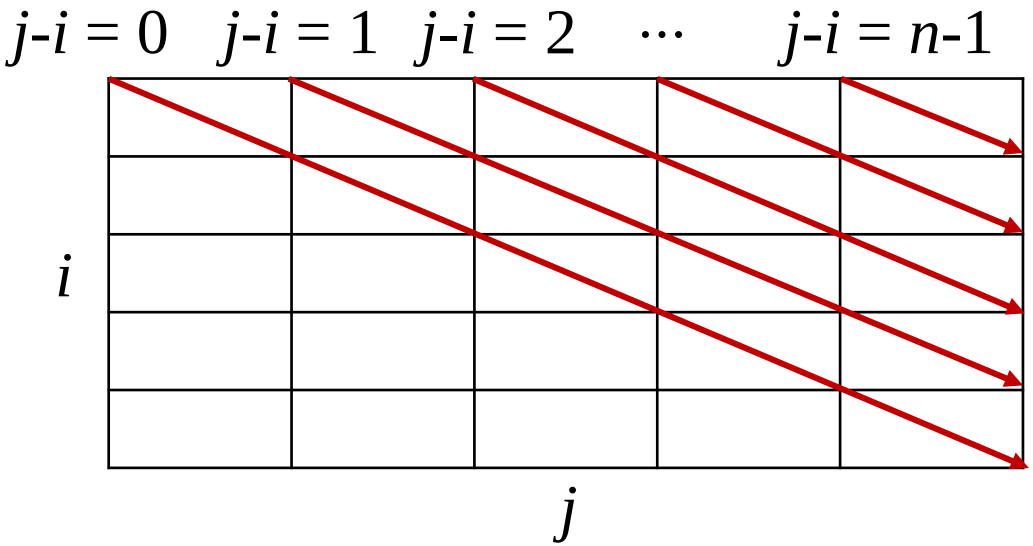
    $m[1, 3], m[2, 4]$

    $m[1, 4]$：k=1时为$m[1, 1] + m[2, 4] + p_0p_1p_4$
    k=2时为$m[1, 2] + m[3, 4] + p_0p_2p_4$
    k=3时为$m[1, 3] + m[4, 4] + p_0p_3p_4$

矩阵链乘法 (续)

- $m[i, j]$ 计算顺序：

  $j\text{-}i = 0,\ j\text{-}i = 1,\ j\text{-}i = 2,\ \ldots,\ j\text{-}i = n\text{-}1$



矩阵链乘法 (续)

```
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 do
    m[i, i] ← 0    // i = j
for l ← 2 to n do  // l：矩阵链长度，i ≠ j 时 1 ≤ j-i = l-1 ≤ n-1
    for i ← 1 to n-l+1 do
        j ← i + l - 1
        m[i, j] ← ∞
        for k ← i to j-1 do
            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
```

矩阵链乘法 (续)

- 时间复杂度：$O(n^3)$，三层循环
- 空间复杂度：$O(n^2)$，保存表m和s
- 较穷举方法指数阶高效得多

矩阵链乘法 (续)

4. 构造最优解

- MATRIX_CHAIN_ORDER求出了计算矩阵链乘积所需的最少标量乘法次数，但并未指出如何进行这种最优代价矩阵链乘法计算
- 表s记录了构造最优解的最优分割信息，可递归求出其中最外层划分位置：$k = s[1, n]$，则进一步求$s[1, k]$和$s[k+1, n]$，直到$s[i, j]$中$i=j$为止

```
PRINT_OPTIMAL_PARENS(s, i, j)
if i = j
    print "A"_i
else print "("
    PRINT_OPTIMAL_PARENS(s, i, s[i, j])
    PRINT_OPTIMAL_PARENS(s, s[i, j]+1, j)
    print ")"
```

矩阵链乘法 (续)

■按照最优括号化计算矩阵链乘积

```
MATRIX_CHAIN_MULTIPLY(A, s, i, j)
if i = j
    return A_i
else
    X ← MATRIX_CHAIN_MULTIPLY(A, s, i, s[i, j])
    Y ← MATRIX_CHAIN_MULTIPLY(A, s, s[i, j]+1, j)
    return MATRIX_MULTIPLY(X, Y)
```