算法设计与分析—入门篇

第四讲 动态规划

哈尔滨工业大学
王宏志
wangzh@hit.edu.cn
http://homepage.hit.edu.cn/pages/wang/

本讲内容

4.1 动态规划的原理

4.2 矩阵乘法问题

4.3 最长公共子序列问题

Why?

- 分治技术的问题
  - 子问题是相互独立的
  - 如果子问题不是相互独立的，分治方法将重复计算公共子问题，效率很低
- 优化问题
  - 给定一组约束条件和一个代价函数，在解空间中搜索具有最小或最大代价的优化解
  - 很多优化问题可分为多个子问题，子问题相互关联，子问题的解被重复使用

What?

- 动态规划的特点
  - 把原始问题划分成一系列子问题
  - 求解每个子问题仅一次，并将其结果保存在一个表中，以后用到时直接存取，不重复计算，节省计算时间
  - 自底向上地计算
- 适用范围
  - 一类优化问题：可分为多个相关子问题，子问题的解被重复使用

How?

- 使用动态规划的条件
  - 优化子结构
    - 当一个问题的优化解包含了子问题的优化解时，我们说这个问题具有优化子结构。
    - 缩小子问题集合，只需那些优化问题中包含的子问题，降低实现复杂性
    - 优化子结构使得我们能自下而上地完成求解过程
  - 重叠子问题
    - 在问题的求解过程中，很多子问题的解将被多次使用

- 动态规划算法的设计步骤
  - 分析优化解的结构
  - 递归地定义最优解的代价
  - 自底向上地计算优化解的代价保存之，并获取构造最优解的信息
  - 根据构造最优解的信息构造优化解

本讲内容

4.1 动态规划的原理

4.2 矩阵乘法问题

4.3 最长公共子序列问题

问题的定义

- 输入：$<A_1, A_2, \ldots, A_n>$，A_i是矩阵
- 输出：计算$A_1 \times A_2 \times \ldots \times A_n$的最小代价方法

矩阵乘法的代价/复杂性：乘法的次数

若A是$p \times q$矩阵，B是$q \times r$矩阵，则$A \times B$的代价是$O(pqr)$

动机

- 矩阵链乘法的实现
  - 矩阵乘法满足结合率。
  - 计算一个矩阵链的乘法可有多种方法:

例如，（$A_1 \times A_2 \times A_3 \times A_4$）

$$= (A_1 \times (A_2 \times (A_3 \times A_4)))$$

$$= ((A_1 \times A_2) \times (A_3 \times A_4))$$

....

$$= ((A_1 \times A_2) \times A_3) \times A_4)$$

- 矩阵链乘法的代价与计算顺序的关系
  - 设A_1=10×100矩阵，A_2=100×5矩阵，A_3=5×50矩阵

  $T((A_1 \times A_2) \times A_3) = 10 \times 100 \times 5 + 10 \times 5 \times 50 = 7500$

  $T(A_1 \times (A_2 \times A_3)) = 100 \times 5 \times 50 + 10 \times 100 \times 50 = 750000$

结论: 不同计算顺序有不同的代价

- 矩阵链乘法优化问题的解空间
  - 设$p(n)$=计算n个矩阵乘积的方法数
  - $p(n)$的递归方程

如此之大的解空间是无法用枚举方法求出最优解的！

$$p(n)=\sum_{k=1}^{n-1} p(k)\,p(n-k) \qquad \textbf{if } n>1$$

$$p(n)=C(n-1)=\text{Catalan数}=\frac{1}{n}\binom{2(n-1)}{n-1}$$

$$=\Omega(4^n/n^{3/2})$$

下边开始设计求解矩阵链乘法问题的动态规划算法

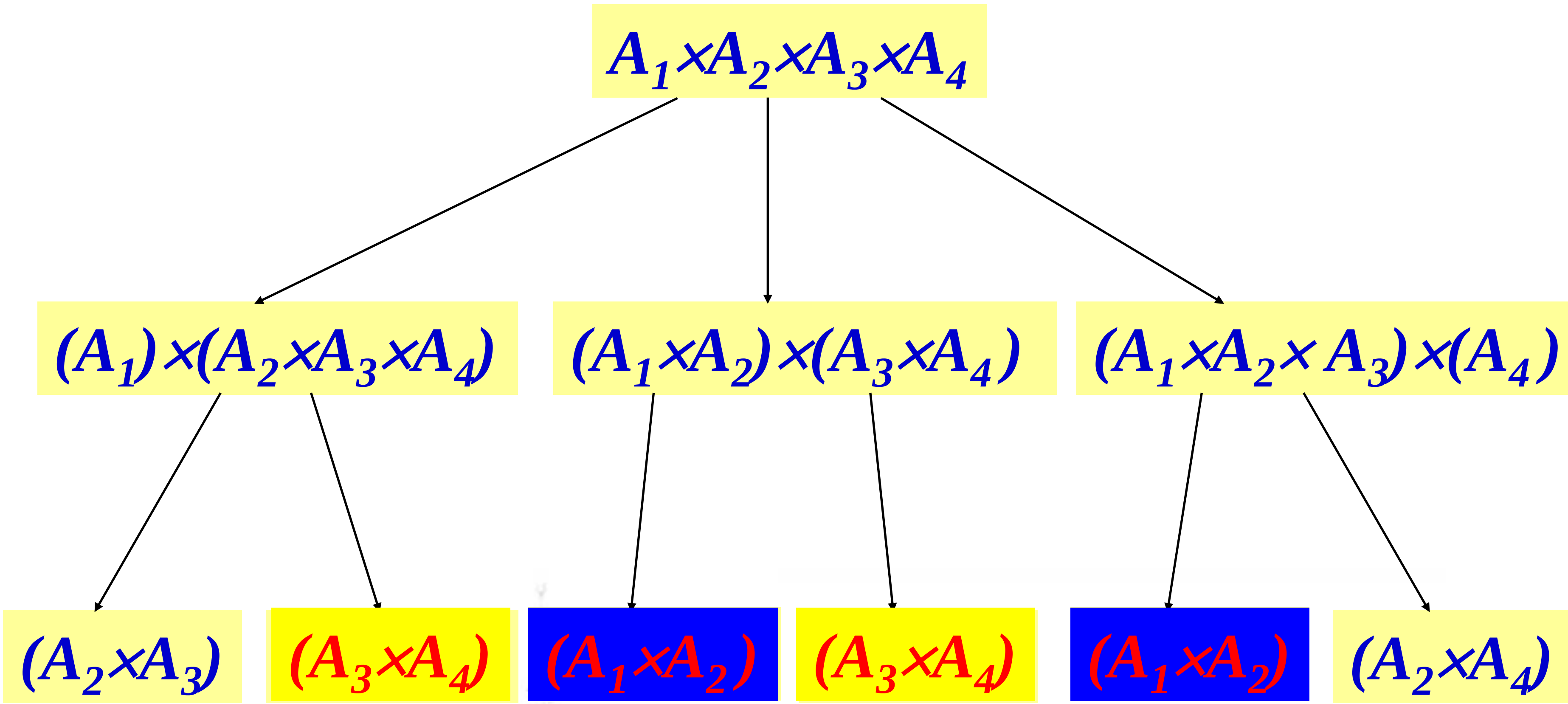
- 分析优化解的结构
- 递归地定义最优解的代价
- 自底向上地计算优化解的代价保存之, 并获取构造最优解的信息
- 根据构造最优解的信息构造优化解

分析优化解的结构

- 两个记号
  - $A_{i\text{-}j}=A_i \times A_{i+1} \times .... \times A_j$
  - $cost(A_{i\text{-}j})$=计算$A_{i\text{-}j}$的代价
- 优化解的结构
  - 若计算A 的优化顺序在k处断开矩阵链，
    序
    优
    化解，对应于子问题$A_{k+1\sim n}$的解必须是$A_{k+1\sim n}$的优化解

具有优化子结构：
问题的优化解包括子问题优化解

- 子问题重叠性



具有子问题重叠性

递归地定义最优解的代价

- 假设
  - $m[i, j]$ = 计算$A_{i\sim j}$的最小乘法数
  - $m[1, n]$ = 计算$A_{1\sim n}$的最小乘法数
  - $A_1 \ldots A_k A_{k+1} \ldots . A_n$ 是优化解(k实际上是不可预知)
- 代价方程

$m[i, i]$ = 计算$A_{i\sim i}$的最小乘法数$= 0$

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

其中, $p_{i-1}p_kp_j$是计算$A_{i\sim k}\times A_{k+1\sim j}$所需乘法数,

$A_{i\sim k}$和$A_{k+1\sim j}$分别是$p_{i-1}\times p_k$和$p_k\times p_j$矩阵.

考虑到所有的k，优化解的代价方程为

$m[i, j] = 0$ if $i=j$

$m[i, j] = \min_{i \le k < j}\{ m[i, k]+m[k+1, j]+p_{i-1} p_k p_j \}$ if $i<j$

自底向上计算优化解的代价

$$m[i, j]= \min_{i\leq k <j}\{ m[i, k] + m[k+1, j] + p_0 p_k p_5 \}$$

$m[1,1]$	$m[1,2]$	$m[1,3]$	$m[1,4]$	$m[1,5]$
	$m[2,2]$	$m[2,3]$	$m[2,4]$	$m[2,5]$
		$m[3,3]$	$m[3,4]$	$m[3,5]$
			$m[4,4]$	$m[4,5]$
				$m[5,5]$

$$m[2,4] = \min\begin{cases} m[2,2]+m[3,4] \\ m[2,3]+m[4,4] \end{cases}$$

$$m[i, j]= \min_{i\le k<j}\{\, m[i, k] + m[k+1, j] + p_{i-1}\, p_k\, p_j \,\}$$

m[1,1]	m[1,2]	m[1,3]	m[1,4]	m[1,5]
	m[2,2]	m[2,3]	m[2,4]	m[2,5]
		m[3,3]	m[3,4]	m[3,5]
			m[4,4]	m[4,5]
				m[5,5]

```
Matrix-Chain-Order(p)
n=length(p)-1;
FOR i=1 TO n DO
    m[i, i]=0;
FOR l=2 TO n DO /* 计算地l对角线 */
    FOR i=1 TO n-l+1 DO
        j=i+l-1;
        m[i, j]=∞;
        FOR k←i To j-1 DO /* 计算m[i,j] */
            q=m[i, k]+m[k+1, j]+p_{i-1}p_k p_j
            IF q<m[i, j] THEN m[i,j]=q;
Return m.
```

获取构造最优解的信息

$S[i,j]$记录$A_iA_{i+1}\ldots A_j$的最优划分处在A_k与A_{k+1}之间

```
Matrix-Chain-Order(p)
n=length(p)-1;
FOR i=1 TO n DO
      m[i, i]=0;
FOR l=2 TO n DO
      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] THEN m[i,j]=q ,
   s[i,j]=k;
Return m and s.
```

构造最优解

$S[i, j]$记录$A_i \ldots A_j$的最优划分处；
$S[i, S[i,j]]$记录$A_i \ldots A_{s[i,j]}$的最优划分处；
$S[S[i,j]+1, j]$记录$A_{s[i,j]+1} \ldots A_j$的最优划分处.

Print-Optimal-Parens(*s*, *i*,

IF $j=i$

THEN Print "*A*"*i*;

ELSE Print "("

Print-Optimal-Parens(*s*, *i*, *s[i, j]*)

Print-Optimal-Parens(*s*, *s[i, j]*+1, *j*)

Print ")"

调用**Print-Optimal-Parens(*s*, *1*, *n*)**

即可输出$A_{1\sim n}$的优化计算顺序

算法复杂性

- 时间复杂性
  - 计算代价的时间
    - *(l, i, k)*三层循环, 每层至多*n-1*步
    - $O(n^3)$
  - 构造最优解的时间: $O(n)$
  - 总时间复杂性为：$O(n^3)$
- 空间复杂性
  - 使用数组*m*和*S*
  - 需要空间$O(n^2)$

本讲内容

4.1 动态规划的原理

4.2 矩阵乘法问题

4.3 最长公共子序列问题

问题的定义

- 子序列
  - $X=(A, B, C, B, D, B)$
  - $Z=(B, C, D, B)$是X的子序例
  - $W=(B, D, A)$不是X的子序例
- 公共子序列
  - Z是序列X与Y的公共子序列如果Z是X的子序也是Y的子序列。

最长公共子序列（*LCS*）问题

输入： $X = (x_1, x_2, \ldots, x_n)$，$Y = (y_1, y_2, \ldots y_m)$

输出： Z = X与Y的最长公共子序列

最长公共子序列结构分析

- 第i前缀
  - 设$X=(x_1, x_2, ..., x_n)$是一个序列，X的第i前缀X_i是一个序列，定义为$X_i=(x_1, ..., x_i)$

例. $X=(A, B, D, C, A)$, $X_1=(A)$, $X_2=(A, B)$, $X_3=(A, B, D)$

- 优化子结构

定理1（优化子结构） 设$X=(x_1, \ldots, x_m)$、$Y=(y_1, \ldots, y_n)$ 是两个序列，$Z=(z_1, \ldots, z_k)$是X与Y的LCS，我们有：

(1) 如果$x_m=y_n$，则$z_k=x_m=y_n$，Z_{k-1}是X_{m-1}和Y_{n-1}的LCS，即，$LCS_{XY} = LCS_{X_{m-1}Y_{n-1}} + <x_m=y_n>$.

(2) 如果$x_m \neq y_n$，且$z_k \neq x_m$，则Z是X_{m-1}和Y的LCS，即 $LCS_{XY}= LCS_{X_{m-1}Y}$

(3) 如果$x_m \neq y_n$，且$z_k \neq y_n$，则Z是X与Y_{n-1}的LCS，即 $LCS_{XY}= LCS_{XY_{n-1}}$

证明:

(1). $X=<x_1, \ldots, x_{m-1}, x_m>$, $Y=<y_1, \ldots, y_{n-1}, x_m>$，则

$$LCS_{XY} = LCS_{X_{m-1}Y_{n-1}} + <x_m=y_n>.$$

设$z_k \neq x_m$，则可加$x_m=y_n$到Z，得到一个长为$k+1$的X与Y的公共序列，与Z是X和Y的LCS矛盾。于是$z_k=x_m=y_n$。

现在证明Z_{k-1}是X_{m-1}与Y_{n-1}的LCS。显然Z_{k-1}是X_{m-1}与Y_{n-1}的公共序列。我们需要证明Z_{k-1}是LCS。设不然，则存在X_{m-1}与Y_{n-1}的公共子序列W，W的长大于$k-1$。增加$x_m=y_n$到W，我们得到一个长大于k的X与Y的公共序列，与Z是LCS矛盾。于是，Z_{k-1}是X_{m-1}与Y_{n-1}的LCS.

（2）$X=<x_1, \ldots, x_{m-1}, x_m>$, $Y=<y_1, \ldots, y_{n-1}, y_n>$，$x_m \neq y_n$，$z_k \neq x_m$，则 $LCS_{XY}= LCS_{X_{m-1}Y}$

由于$z_k \neq x_m$，Z是X_{m-1}与Y的公共子序列。我们来证Z是X_{m-1}与Y的LCS。设X_{m-1}与Y有一个公共子序列W，W的长大于k, 则W也是X与Y 的公共子序列，与Z是LCS矛盾。

（3）同(2)可证。

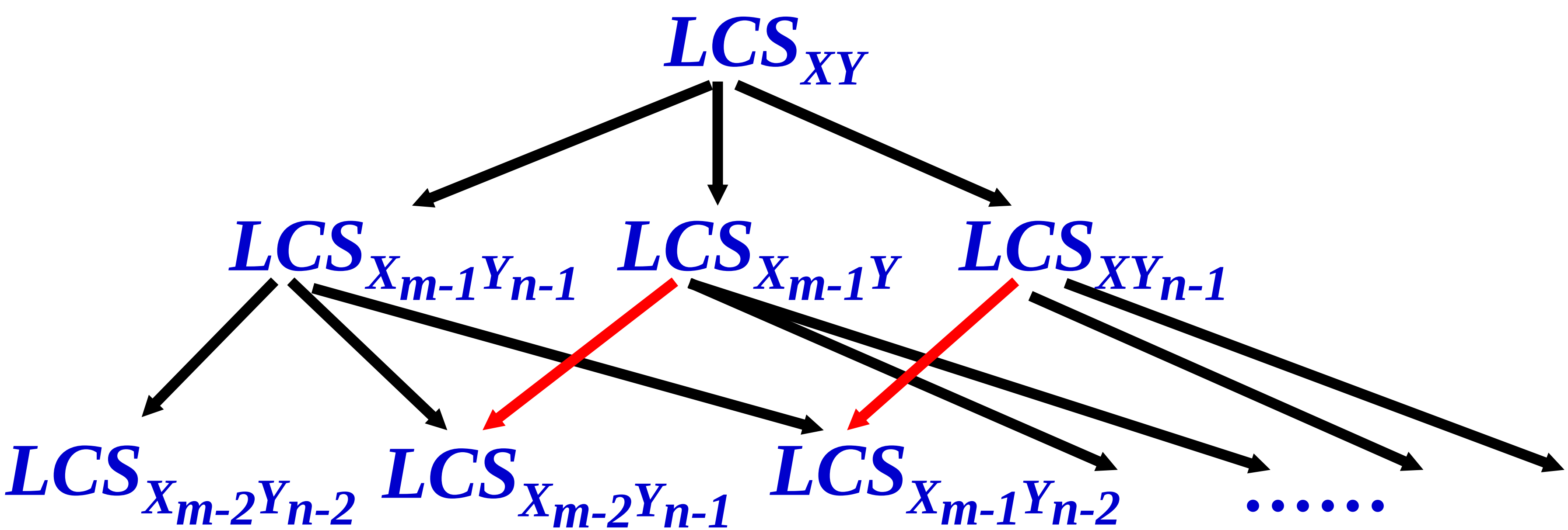
***X*和*Y*的*LCS*的优化解结构为**

$$LCS_{XY}=LCS_{X_{m-1}Y_{n-1}}+\langle x_m=y_n\rangle \quad \text{if } x_m=y_n$$

$$LCS_{XY}=LCS_{X_{m-1}Y} \quad \text{if } x_m\neq y_n,\ z_k\neq x_m$$

$$LCS_{XY}=LCS_{XY_{n-1}} \quad \text{if } x_m\neq y_n,\ z_k\neq y_n$$

- 子问题重叠性



LCS问题具有子问题重叠性

建立LCS长度的递归方程

- $C[i, j]$ = X_i与Y_j 的*LCS*的长度
- *LCS*长度的递归方程

$$
\begin{aligned}
&C[i, j] = 0 && \text{if } i=0 \text{ 或 } j=0 \\
&C[i, j] = C[i-1, j-1] + 1 && \text{if } i, j>0 \text{ 且 } x_i = y_j \\
&C[i, j] = Max(C[i, j-1], C[i-1, j]) && \text{if } i, j>0 \text{ 且 } x_i \neq y_j
\end{aligned}
$$

自底向上计算LCS的长度

- 基本思想

C[i-1, j-1]	*C[i-1,j]*
C[i, j-1]	*C[i, j]*

- 计算过程

C[0,0]	*C[0,1]*	*C[0,2]*	*C[0,3]*	*C[0,4]*
C[1,0]	*C[1,1]*	*C[1,2]*	*C[1,3]*	*C[1,4]*
C[2,0]	*C[2,1]*	*C[2,2]*	*C[2,3]*	*C[2,4]*
C[3,0]	*C[3,1]*	*C[3,2]*	*C[3,3]*	*C[3,4]*

- 计算*LCS*长度的算法
  - 数据结构

    C[0:m,0:n]: *C[i,j]*是X_i与Y_j的*LCS*的长度

    B[1:m,1:n]: *B[i,j]*是指针，指向计算*C[i,j]*时所选择的子问题的优化解所对应的*C*表的表项

```
LCS-length(X, Y)
 m←length(X);  n←length(Y);
 For i←1 To m Do C[i,0]←0;
 For j←1 To n Do C[0,j]←0;
 For i←1 To m Do
     For j←1 To n Do
         If x_i = y_j
         Then C[i,j]←C[i-1,j-1]+1;  B[i,j]←"↖";
         Else If C[i-1,j]≥C[i,j-1]
                 Then C[i,j]←C[i-1,j]; B[i,j]←"↑";
                 Else C[i,j]←C[i,j-1]; B[i,j]←"←";
 Return C and B.
```

构造优化解

- 基本思想
  - 从*B[m, n]*开始按指针搜索
  - 若*B[i, j]*=“↖”，则x_i=y_j是*LCS*的一个元素
  - 如此找到的“*LCS*”是*X*与*Y*的*LCS*

Print-LCS(B, X, i, j)

IF $i=0$ or $j=0$ THEN Return;

IF $B[i, j]$=“↖”

THEN Print-LCS(B, X, $i-1$, $j-1$);

Print x_i;

ELSE If $B[i, j]$=“↑”

THEN Print-LCS(B, X, $i-1$, j);

ELSE Print-LCS(B, X, i, $j-1$).

Print-LCS(B, X, length(X), length(Y))

可打印出X与Y的LCS。

算法复杂性

- 时间复杂性
  - 计算代价的时间
    - (i, j)两层循环,i循环m步, j循环n步
    - $O(mn)$
  - 构造最优解的时间: $O(m+n)$
  - 总时间复杂性为：$O(mn)$
- 空降复杂性
  - 使用数组C和B
  - 需要空间$O(mn)$