算法设计与分析

动态规划

Dynamic Programming

15.1 方法概述

- 一些术语和概念
- 最优性原理
- 方法的基本求解步骤
- 动态规划法的适用条件
- 最优性原理判别举例
- 设计技巧——阶段划分和状态表示
- 存在的问题

概念（1）

- **动态规划(dynamic programming)**是运筹学的一个分支，20世纪50年代初美国数学家R.E.Bellman等人在研究优化问题求解时，提出了著名的**最优化原理**(*principle of optimality*)，把多阶段过程问题求解转化为一系列单阶段问题求解，创立了解决这类过程优化问题的新方法—动态规划。

概念（2）

- **多阶段决策问题：**求解的问题可以划分为一系列相互联系的阶段，在每个阶段都需要作出决策，且上一个阶段决策的选择会影响下一个阶段的决策，从而影响整个过程的活动路线，
- 求解的目标是选择各个阶段的决策使整个过程达到最优。

概念（3）

- **动态规划**主要用于求解以阶段划分的动态过程的优化问题。

- 动态规划可应用于多种**优化问题**，如最短路线、库存管理、资源分配、设备更新、排序、装载等问题，用动态规划方法比用其它方法求解更为方便。

切杆问题

- 问题描述及举例
- 问题满足最优子结构
- 递归实现
- 动态规划算法

问题描述及举例（1）

- 问题描述

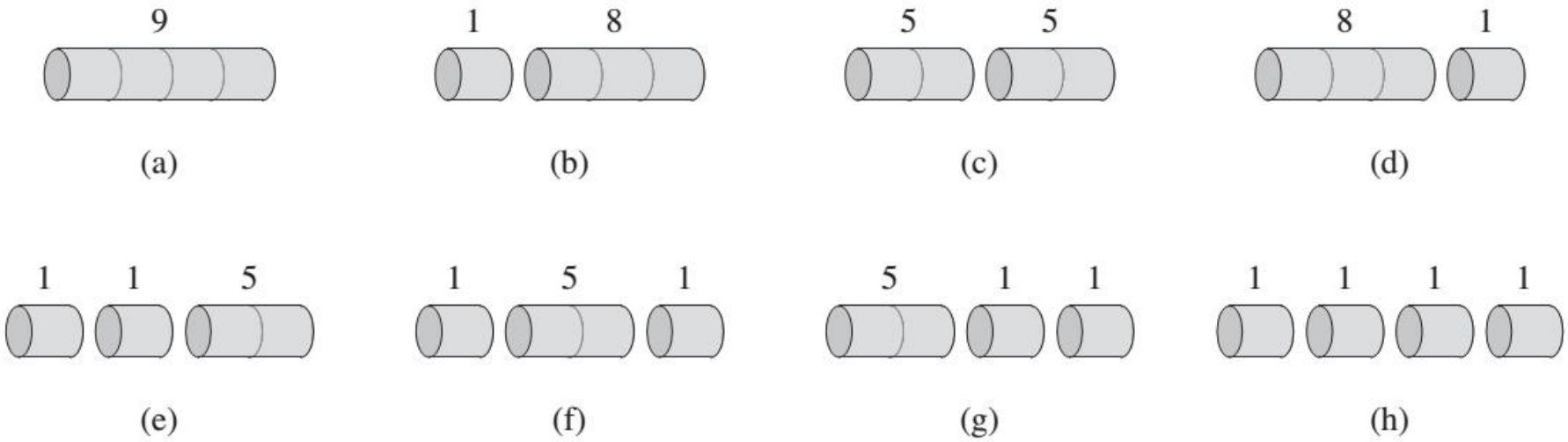
钢条切割问题：给定一段长度为n英寸的钢条和一个价格表p_i（i=1, 2, …, n），求切割钢条方案，使得销售收益r_n最大。

切割工序本身没有成本支出。 Serling公司管理层希望知道最佳的切割方案。假定我们知道Serling公司出售一段长度为i英寸的钢条的价格为p_i（i=1, 2, …, 单位为美元）。钢条的长度均为整英寸。如下为一个价格表的样例。

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

问题描述及举例（2）

下图给出了4英寸钢条所有可能的切割方案，包括根本不切割的方案。



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

最优子结构

定义：r_k　　是长度为k的钢条的最优切割代价

递推关系：

$$r_n = \max_{1 \le i \le n}(p_i + r_{n-i})$$

•问题分解的方式为：将长度为n的钢条分解为左边开始一段，以及剩余部分继续分解的结果。

最优子结构和递归实现

- 自顶向下递归实现

$$r_n = \max_{1 \le i \le n}(p_i + r_{n-i})$$

在此公式中，原问题的最优解只包含一个相关子问题(右端剩余部分)的解，而不是两个。

```
CUT-ROD(p, n)
1  if n == 0
2       return 0
3  q = -∞
4  for i = 1 to n
5       q = max(q, p[i] + CUT-ROD(p, n - i))
6  return q
```

递归实现的效率

- CUT-ROD的效率差

反复地用相同的参数值对自身进行递归调用，即它反复求解相同的子问题。

下面显示了n=4时的调用过程： CUT-ROD(p, n)对i=1,2,…,n调用：CUT-ROD(p, n-i)，等价于对j=0,1,…,n-1调用CUT-ROD(p, j)。当这个过程递归展开时，它所做的工作量(用n的函数的形式描述)会爆炸性地增长。

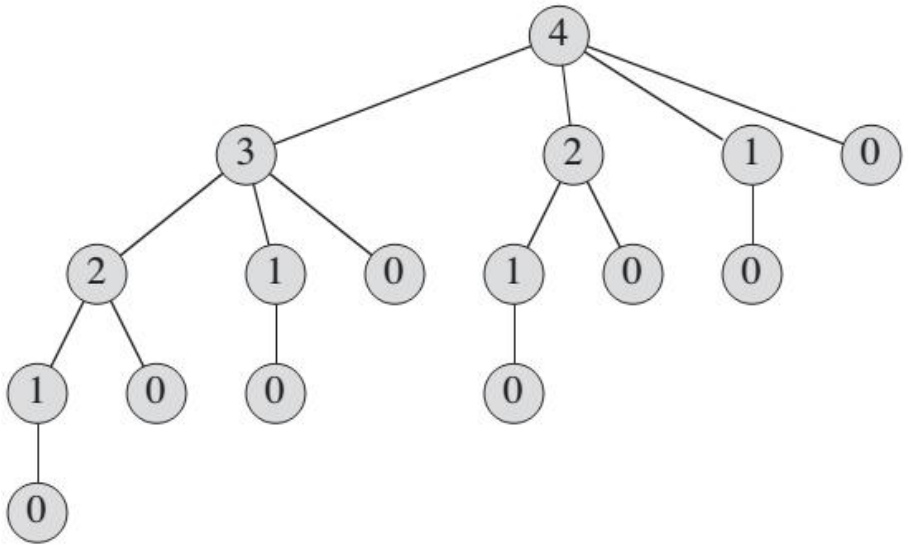


图15-1

动态规划-带备忘的自顶向下法

保存每个子问题的解，当需要一个子问题的解时，过程首先检查是否已经保存过此解。如果是，则直接返回保存的值，从而节省了计算时间；否则，按通常方式计算这这子问题。

```
MEMOIZED-CUT-ROD-AUX(p, n, r)
1  if r[n] ≥ 0
2      return r[n]
3  if n == 0
4      q = 0
5  else q = −∞
6      for i = 1 to n
7          q = max(q, p[i] + MEMOIZED-CUT-ROD-AUX(p, n − i, r))
8  r[n] = q
9  return q
```

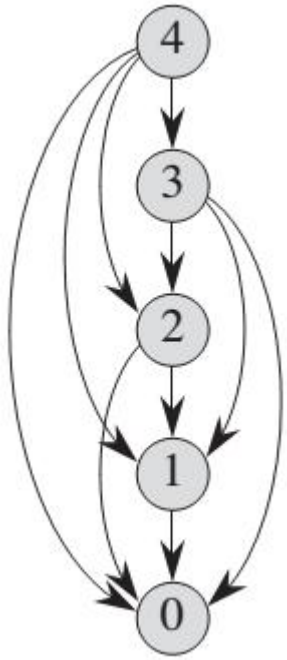
```
MEMOIZED-CUT-ROD(p, n)
1  let r[0..n] be a new array
2  for i = 0 to n
3      r[i] = −∞
4  return MEMOIZED-CUT-ROD-AUX(p, n, r)
```

动态规划-自底向上法

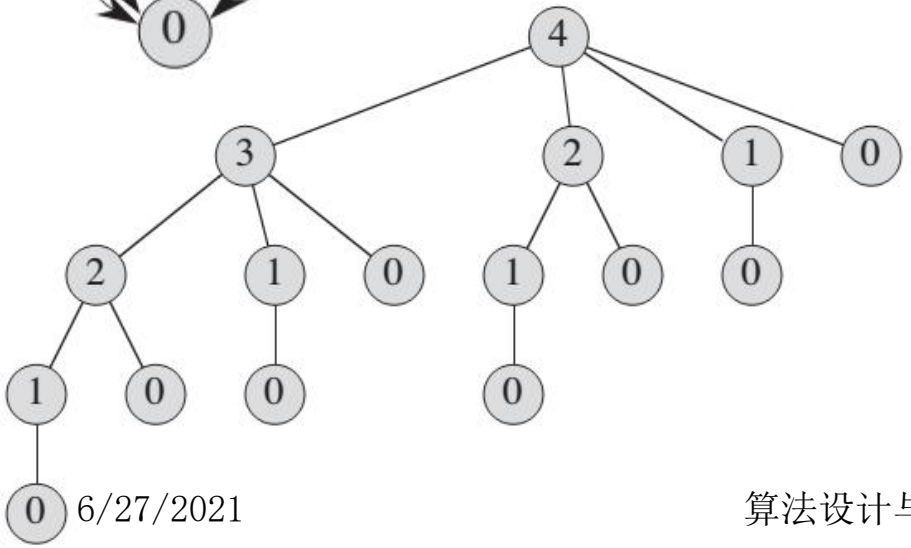
将子问题按规模排序，按由小到大的顺序解。当求解某个子问题时，它所依赖的些更小的子问题都已求解完毕，结果已经保存。每个子问题只需求解一次，当我们求解它(也是第一次遇到它)时，它的所有前提子问题都已求解完成。

```
BOTTOM-UP-CUT-ROD(p, n)
1  let r[0..n] be a new array
2  r[0] = 0
3  for j = 1 to n
4      q = -∞
5      for i = 1 to j
6          q = max(q, p[i] + r[j - i])
7      r[j] = q
8  return r[n]
```

子问题图与重构解



n=4时，钢条切割问题的子问题图。顶点的标号给出了子问题的规模。有向边(x, y)表示当求解子问题x时需要子问题y的解。此图实际上是图15-1中递归调用树的简化版—树中标号相同的结点收缩为图中的单一顶点，所有边均从父节点指向子结点。



EXTENDED-BOTTOM-UP-CUT-ROD(p, n)

```
1   let r[0..n] and s[0..n] be new arrays
2   r[0] = 0
3   for j = 1 to n
4       q = -∞
5       for i = 1 to j
6           if q < p[i] + r[j - i]
7               q = p[i] + r[j - i]
8               s[j] = i
9       r[j] = q
10  return r and s
```

切杆问题——分阶段求解

阶段：切短的钢条是前面阶段，切长的钢条是后面阶段 。

- 长钢条的切割是在短钢条切割方案已知的基础上进行的。
- 每个阶段问题的最优求解都是基于前一个阶段的解是最优的基础上。

一些术语和概念

- 阶段：把所给的问题的求解过程恰当地分成若干个相互联系的阶段。

切短钢条是前面阶段，切长钢条的阶段

- 状态：状态表示每个阶段开始时，问题的客观状况。状态既是该阶段的某个起点，又是前一个阶段的某个终点。通常一个阶段有若干个状态。
  - 状态的无后效性：如果某阶段状态给定后，则后续发展不受该阶段以前各阶段状态的影响，称为无后效性。

更短钢条的切割方案对长钢条的切割没有影响

  注：适于动态规划法求解的问题具有状态的无后效性

- 策略：各个阶段决策的确定后，就组成了一个决策序列，该序列称之为一个策略。由某个阶段开始到终止阶段的过程称为子过程，其对应的某个策略称为子策略。

最优性原理

- Bellman的原定义如下：

  An optimal policy has the property that whatever the initial state and initial decision are, then remaining decisions must constitute an optimal policy with regard to the state resulting from first decision.

- Bellman最优性原理：

  求解问题的一个最优策略序列的子策略序列总是最优的，则称该问题满足最优性原理。

  注：对具有最优性原理性质的问题而言，如果有一决策序列包含有非最优的决策子序列，则该决策序列一定不是最优的。

- 钢条切割问题满足最优子结构(optimal substructure)性质：第i阶段是最优的前提是，前面阶段的切割是最优的。

适用条件

动态规划法的有效性依赖于问题本身所具有的两个重要的适用性质

- 最优子结构

如果问题的最优解是由其子问题的最优解来构造，则称该问题具有最优子结构性质。

- 重叠子问题

在用递归算法自顶向下解问题时，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。动态规划算法正是利用了这种子问题的重叠性质，对每一个子问题只解一次，而后将其解保存在一个表格中，在以后该子问题的求解时直接查表。

方法的基本思想

- 动态规划的思想实质是分治思想和解决冗余。
- 与分治法类似的是

  将原问题分解成若干个子问题，先求解子问题，然后从这些子问题的解得到原问题的解。
- 与分治法不同的是

  经分解的子问题往往不是互相独立的。若用分治法来解，有些共同部分（子问题或子子问题）被重复计算了很多次。
- 如果能够保存已解决的子问题的答案，在需要时再查找，这样就可以避免重复计算、节省时间。动态规划法用一个表来记录所有已解的子问题的答案。这就是动态规划法的基本思路。具体的动态规划算法多种多样，但它们具有相同的填表方式。

方法的求解步骤

①找出最优解的性质，并刻画其结构特征；

②递归地定义最优值（写出动态规划方程）；

③以自底向上的方式计算出最优值；

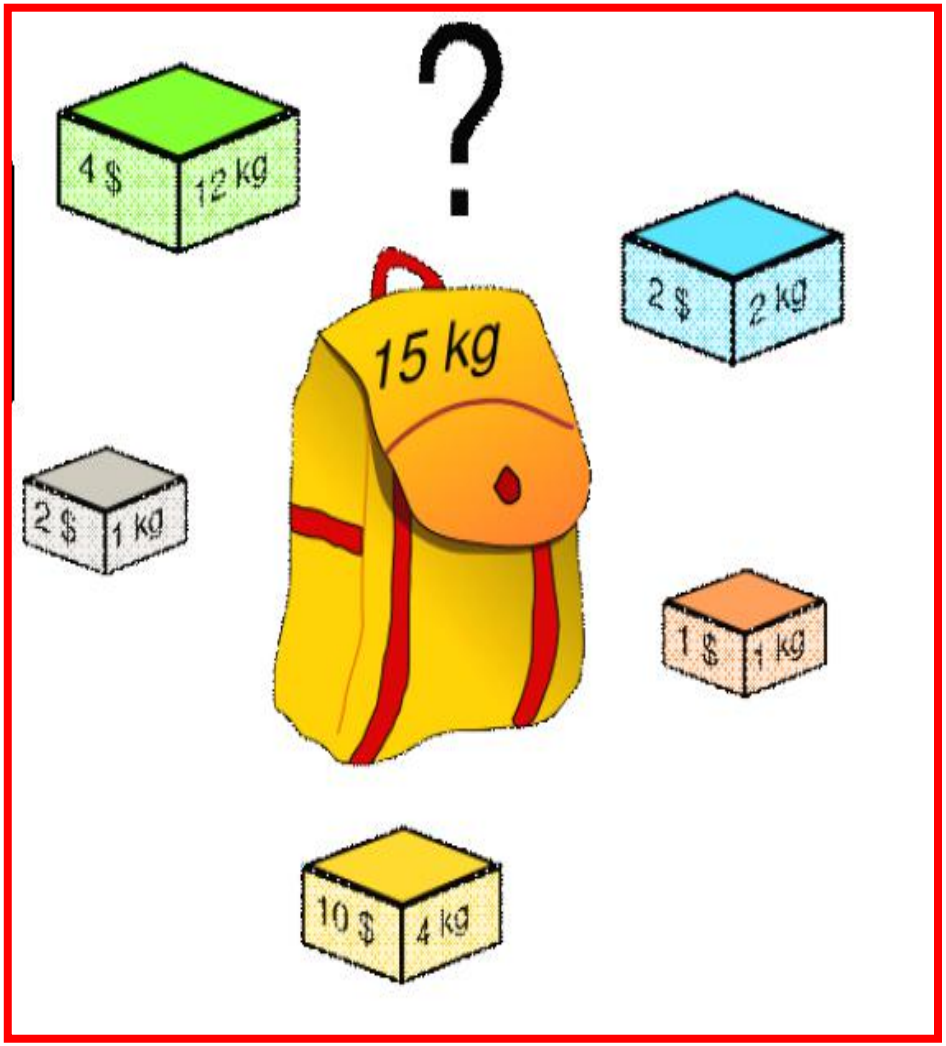
④根据计算最优值时记录的信息，构造最优解。

注：

—步骤①~③是动态规划算法的基本步骤。如果不需要求出最优值的具体表达，步骤④可以省略；

—若需要求出问题的一个最优解，则必须执行步骤④，步骤③中记录的信息是构造最优解的基础

背包问题



给定n种物品和一个背包，物品i的重量是w_i，其价值为v_i，背包的容量为C。如何选择装入背包的物品，使得装入背包中物品的总价值最大? 0/1背包问题。

背包问题

设x_i表示物品i装入背包的情况，则当x_i=0时，表示物品i没有被装入背包，x_i=1时，表示物品i被装入背包。有如下约束条件和目标函数：

$$\begin{cases} \sum_{i=1}^{n} w_i x_i \le C \\ x_i \in \{0,1\} \quad (1 \le i \le n) \end{cases} \qquad \text{（式1）}$$

$$\max \sum_{i=1}^{n} v_i x_i \qquad \text{（式2）}$$

问题归结为寻找一个满足约束条件式1，并使目标函数式2达到最大的解向量$X=(x_1, x_2, \ldots, x_n)$。

背包问题

令$V(i,\ j)$表示在前$i(1\le i\le n)$个物品中能够装入容量为j（$1\le j\le C$）的背包中的物品的最大值，则可以得到如下动态规划函数：

$$V(i,j)=\begin{cases}V(i-1,j) & j<w_i\\ \max\{V(i-1,j),\ V(i-1,j-w_i)+v_i\} & j>w_i\end{cases} \quad \text{（式3）}$$

$$V(i,\ 0)=\ V(0,\ j)=0$$

（式4）

例如，有5个物品，其重量分别是{2，2，6，5，4}，价值分别为{6，3，5，4，6}，背包的容量为10，求装入背包的物品和获得的最大价值。

		0	1	2	3	4	5	6	7	8	9	10
	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
w_1=2 v_1=6	**1**	**0**	**0**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**
w_2=2 v_2=3	**2**	**0**	**0**	**6**	**6**	**9**	**9**	**9**	**9**	**9**	**9**	**9**
w_3=6 v_3=5	**3**	**0**	**0**	**6**	**6**	**9**	**9**	**9**	**9**	**11**	**11**	**14**
w_4=5 v_4=4	**4**	**0**	**0**	**6**	**6**	**9**	**9**	**9**	**10**	**11**	**13**	**14**
w_5=4 v_5=6	**5**	**0**	**0**	**6**	**6**	**9**	**9**	**12**	**12**	**15**	**15**	**15**

x_1=1

x_2=1

x_3=0

x_4=0

x_5=1

背包求解（填表）过程

可以看到，装入背包的物品的最大价值是15，根据式5，装入背包的物品为X={1, 1, 0, 0, 1}。

背包问题——分阶段求解

- 第一阶段，只装入前1个物品，要求能够得到最大价值；
- 第二阶段，只装入前2个物品，要求能够得到最大价值；这个问题求解要在第一阶段最优解的基础上进行；
- 依此类推，直到第n个阶段。
- 最后，$V(n,C)$便是在容量为C的背包中装入n个物品时取得的最大价值。
- 每个阶段问题的求解都是基于前一个阶段的解是最优的基础上。

0/1背包问题满足最优性原理

设$(x_1, x_2, \ldots, x_n)$是所给0/1背包问题的一个最优解，则$(x_2, \ldots, x_n)$是下面一个子问题的最优解：

$$\max \sum_{i=2}^{n} v_i x_i \qquad \begin{cases} \sum_{i=2}^{n} w_i x_i \le C - w_1 x_1 \\ x_i \in \{0,1\} \quad (2 \le i \le n) \end{cases}$$

如若不然，设$(y_2, \ldots, y_n)$是上述子问题的一个最优解则 $\sum_{i=2}^{n} v_i y_i > \sum_{i=2}^{n} v_i x_i$ ，且 $w_1 x_1 + \sum_{i=2}^{n} w_i y_i \le C$ 。因此，

$$v_1 x_1 + \sum_{i=2}^{n} v_i y_i > v_1 x_1 + \sum_{i=2}^{n} v_i x_i = \sum_{i=1}^{n} v_i x_i$$

这说明$(x_1, y_2, \ldots, y_n)$是所给0/1背包问题比$(x_1, x_2, \ldots, x_n)$更优的解，从而导致矛盾。

最优性原理判别举例（1）

例1：设G是一个有向加权图，则G从顶点i到顶点j之间的最短路径问题满足最优性原理。

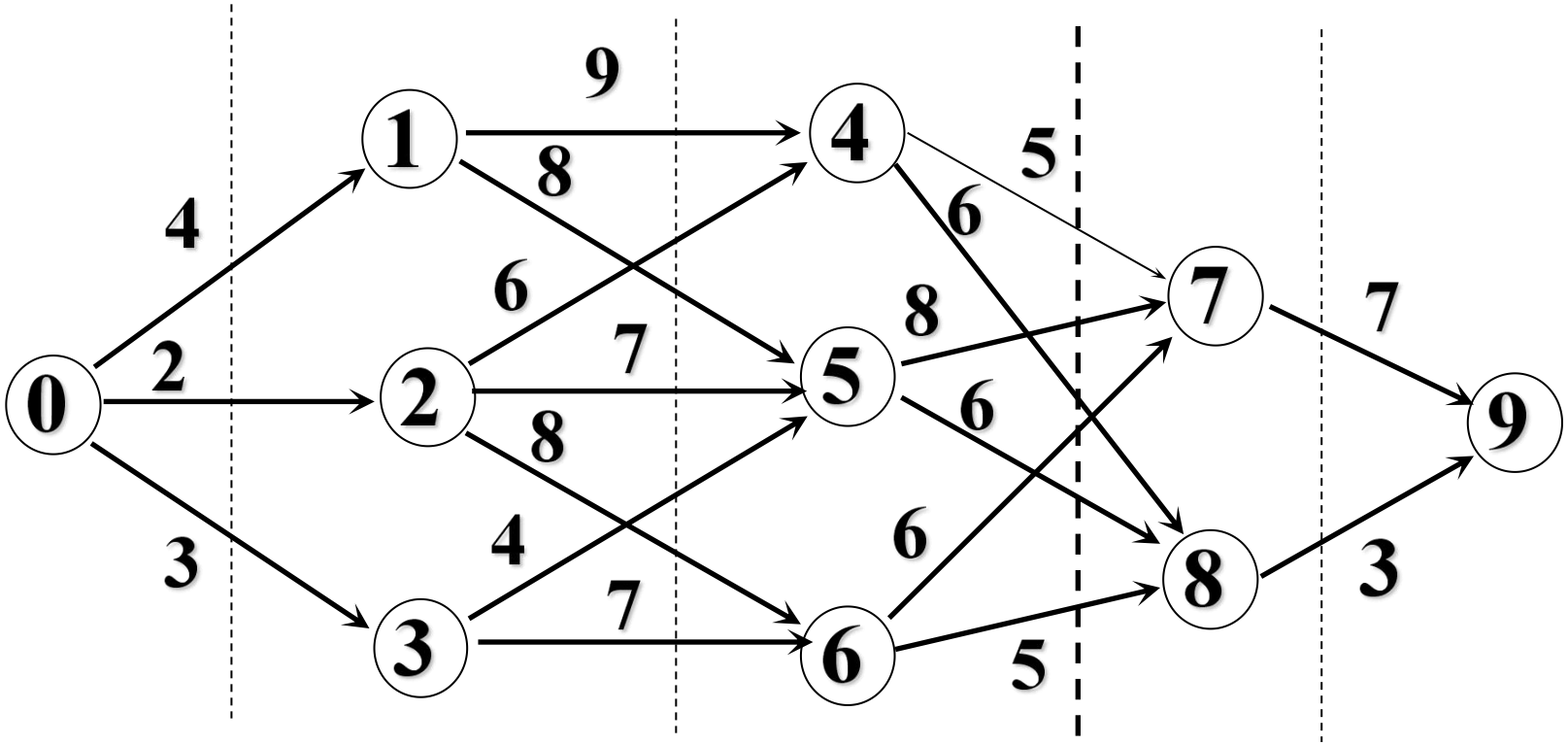


图1 有向图

最优性原理判别举例（1）

证明：（反证）

设$i\sim i_p\sim i_q\sim j$是一条最短路径，但其中子路径$i_p\sim i_q\sim j$不是最优的，

假设最优的路径为$i_p\sim i_q'\sim j$

则我们重新构造一条路径：$i\sim i_p\sim i_q'\sim j$

显然该路径长度小于$i\sim i_p\sim i_q\sim j$，

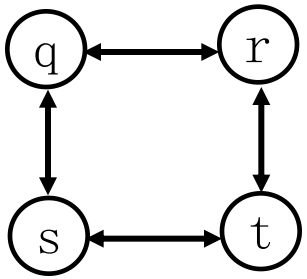
与$i\sim i_p\sim i_q\sim j$是顶点i到顶点j的最短路径相矛盾.

所以，原问题满足最优性原理。

最优性原理判别举例（2）

- 例2：最长路径问题不满足最优性原理。

证明：



q→r→t是q到t的最长路径，

而q→r的最长路径是q→s→t→r

r→t的最长路径是r→q→s→t

但q→r和r→t的最长路径合起来并不是q到t的最长路径。所以，原问题并不满足最优性原理。

□

注：因为q→r和r→t的子问题都共享路径s→t，组合成原问题解时，有重复的路径对原问题是不允许的 。

15.3 矩阵链乘法

- 问题描述
- 加括号的方案数
- 动态规划算法

问题描述

- 问题描述：

  给定n个矩阵$A_1, A_2, \cdots, A_n$，A_i的维数为$p_{i-1} \times p_i (1 \leqslant i \leqslant n)$，

以一种最小化标量乘法次数的方式进行完全括号化。

- Remark：

  1. 设$A_{p \times q}, A_{q \times r}$两矩阵相乘，普通乘法的次数为$p \times q \times r$

  2. 加括号对乘法次数的影响

     如：$A_{10 \times 100} \times B_{100 \times 5} \times C_{5 \times 50}$

     ((AB)C)：7500次　　(A(BC))：75000次

加括号的方案数——蛮力法

用p(n)表示n个矩阵链乘的括号化方案的数量，如果将n个矩阵从第k和第k+1处隔开，对两个子序列再分别加扩号，用p(n)表示则可以得到下面递归式：

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

$$\Rightarrow p(n)=C(n-1)\text{为}Catalan\text{数}$$

$$C(n)=\frac{1}{n+1}\binom{2n}{n}=\Omega\left(\frac{4^n}{n^{3/2}}\right)$$

呈指数增长

1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, …

动态规划算法（1）

- 最优性原理分析

1. 矩阵链乘问题满足最优性原理

记A[i:j]为$A_iA_{i+1}\cdots A_j$链乘的一个最优括号方案，设A[i:j]的最优次序中含有二个子链A[i:k]和A[k+1:j]，则A[i:k]和A[k+1:j]也是最优的。（反证可得）

2. 矩阵链乘的子问题空间：A[i:j]， $1\leqslant i\leqslant j\leqslant n$

A[1:1]， A[1:2]， A[1:3]， … ， A[1:n]

A[2:2]， A[2:3]， … ， A[2:n]

… …

…

A[n-1:n-1]， A[n-1:n]

动态规划算法（2）

- 递归求解最优解的值

记m[i][j]为计算A[i:j]的最少乘法数，则原问题的最优值为m[1][n]，那么有

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

这里，$\underbrace{(A_iA_{i+1}\cdots A_k)}_{p_{i-1}\times p_k}\times\underbrace{(A_{k+1}A_{k+2}\cdots A_j)}_{p_k\times p_j}$

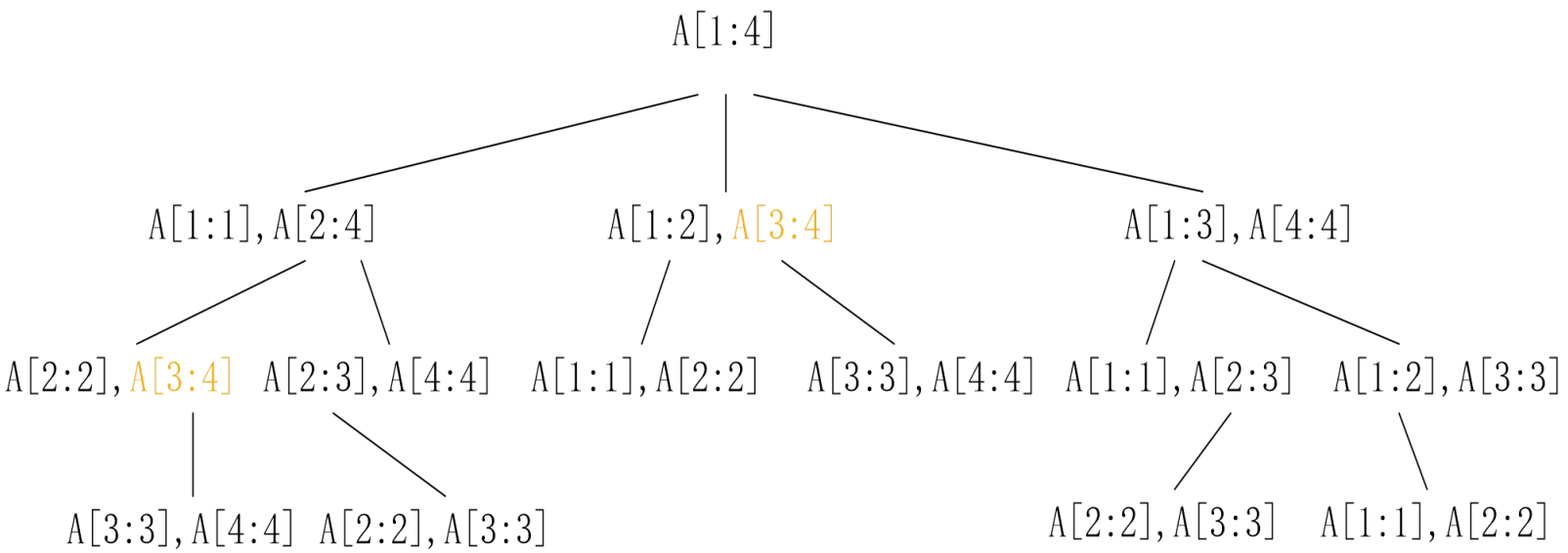
A[i:j]的最优断开位置是k，记录到表s[i][j]中

注：m[i][j]实际是子问题最优解的解值，保存下来避免重复计算。

动态规划算法（3）

- 递归求解最优解的值(Cont.)

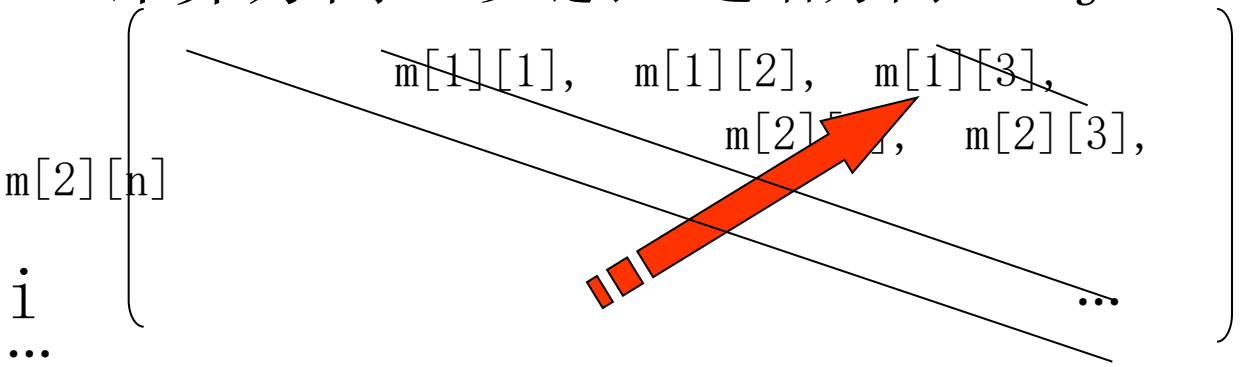
  计算m[1][4]过程如下：



A[3:4]被计算了2次，保存下来可以节省许多时间

动态规划算法（4）

- 自底向上记忆化方式求解m[i][j]
  - 计算方向：以链长*l*递增方向.　　j



  - 计算最优值的算法(Godbole,1973)： P213

$$T(n)=O(n^3),\quad S(n)=O(n^2)$$

  - 注：①如果自顶向下计算(含重复计算)，这样效率较低(为$\Omega(2^n)$)。

②也可以采用自顶向下的记忆型递归算法P220。

动态规划算法（6）

- 计算示例：

	行 x 列
A1	30x35
A2	35x15
A3	15x5
A4	5x10
A5	10x20
A6	20x25

		j					
	s	1	2	3	4	5	6
i	1		1	1	3	3	3
	2			2	3	3	3
	3				3	3	3
	4					4	5
	5						5
	6						

		j					
	m	1	2	3	4	5	
i	1	0	15750	7875	9375	11875	15625
	2		0	2625	4375	7125	10500
	3			0	750	2500	5375
	4				0	1000	3500
	5					0	5000
	6						0

Result

$((A_1(A_2A_3))((A_4A_5)A_6))$

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

动态规划算法（5）

- 构造最优解
  - 利用s[i][j]中保存的k，进行对A[i:j]的最佳划分，加括号为$(A_iA_{i+1}\cdots A_k)\times(A_{k+1}A_{k+2}\cdots A_j)$
  - 输出最优解：P215

```
PrintOptimalParens(s, i, j)
{   if i=j then
        print  “A” i;
    else
    {  print  “(” ;
       PrintOptimalParens(s, i, s[i,j]);
       PrintOptimalParens(s, s[i,j]+1, j);
       print  “)” ;
    }
}
```

i \ S	1	2	3	4	5	6
1		1	1	3	3	3
2			2	3	3	3
3				3	3	3
4					4	5
5						5
6						

最长公共子序列（LCS）

- 问题描述
- 如何求X、Y的LCS
  - —LCS最优解结构特征（step1）
  - —子问题的递归解（step2）
  - —计算最优解值（step3）
  - —构造一个LCS（step4）

问题描述（1）

- 子序列定义

  给定序列X=($x_1, x_2, \cdots, x_m$)，序列Z=($z_1, z_2, \cdots, z_k$)是X的一子序列，必须满足：若X的索引中存在一个严格增的序列$i_1, i_2, \cdots, i_k$，使得对所有的j=1~k，均有$x_{i_j}=z_j$

  例如，序列Z={B，C，D，B}是序列X={A，B，C，B，D，A，B}的子序列，相应的递增下标序列为{2，3，5，7}。

- 两个序列的公共子序列

  Z是X和Y的子序列，则Z是两者的公共子序列CS。

- 最长的公共子序列(LCS)

  在X和Y的CS中，长度最大者为一个最长公共子序列LCS。

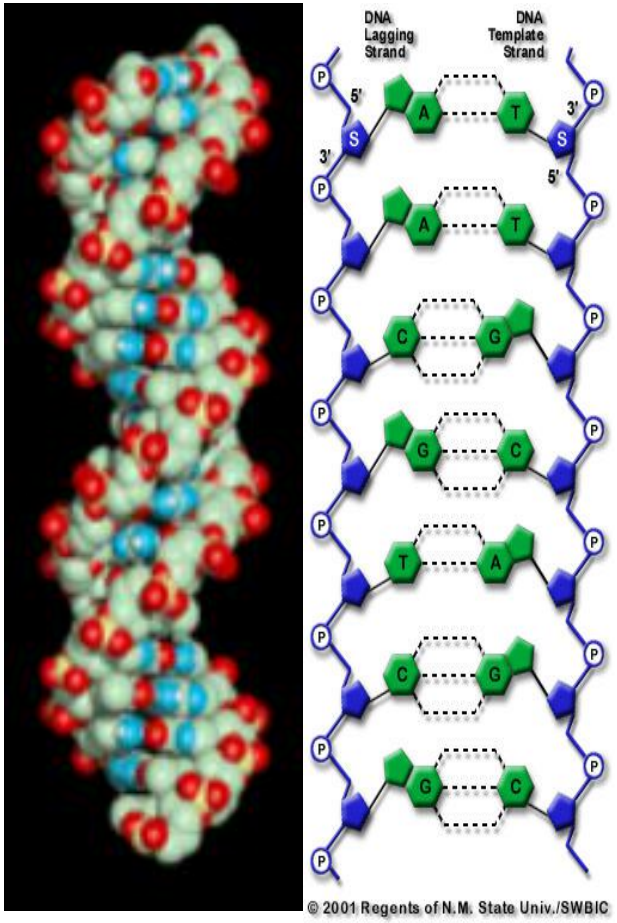
问题描述（2）

- 举个栗子：　　两个DNA序列

S_1 = ACCGGTCGAGTGCGCGGAAGCCGGCCGAA

S_2 = GTCGTTCGGAATGCCGTTGCTCTGTAAA

LCS S_3 = GTCGTCGGAAGCCGGCCGAA.



求LCS的step1（1）

- Step1：LCS最优解的结构特征

  定义X的i^{th}前缀：$X_i=(x_1, x_2, \cdots, x_i)$，$i=1\sim m$

  $X_0=\Phi$　　Φ为空集

- Th15.1（一个LCS的最优子结构）

  设序列$X=(x_1, x_2, \cdots, x_m)$和$Y=(y_1, y_2, \cdots, y_n)$，$Z=(z_1, z_2, \cdots, z_k)$是X和Y的任意一个LCS，则

  (1)若$x_m=y_n$，==> $z_k=x_m=y_n$且Z_{k-1}是X_{m-1}和Y_{n-1}的一个LCS；

  (2)若$x_m<>y_n$且$z_k<>x_m$，==> Z是X_{m-1}和Y的一个LCS；

  (3)若$x_m<>y_n$且$z_k<>y_n$，==> Z是X和Y_{n-1}的一个LCS；

  注：由此可见，2个序列的最长公共子序列可由(1)(2)(3)算出，(2)(3)的解是对应子问题的最优解。因此，最长公共子序列问题具有**最优子结构性质**。

求LCS的step1（2）

- Th15.1 的证明

(1)若$x_m=y_n$，==> $z_k=x_m=y_n$，则Z_{k-1}是X_{m-1}和Y_{n-1}的一个LCS；

（应用反证法）

先证：$z_k=x_m=y_n$。

若$z_k<>x_m$（也有$z_k<>y_n$），则将x_m加到Z后，于是获得X和Y的长度为k+1的CS，与Z是X和Y的LCS矛盾。

==> $z_k=x_m=y_n$

再证：Z_{k-1}是X_{m-1}和Y_{n-1}的一个LCS。

由Z的定义 ==> 前缀Z_{k-1}是X_{m-1}和Y_{n-1}的CS(长度为k-1)

若Z_{k-1}不是X_{m-1}和Y_{n-1}的LCS，则存在一个X_{m-1}和Y_{n-1}的公共子序列W，W的长度>k-1，于是将z_k加入W之后，则产生的公共子序列长度>k，与Z是X和Y的LCS矛盾。

==> Z_{k-1}是X_{m-1}和Y_{n-1}的一个LCS

求LCS的step1（3）

- Th15.1 的证明

(2)若$x_m <> y_n$且$z_k <> x_m$， ==> Z是X_{m-1}和Y的一个LCS；

∵ $z_k <> x_m$，则Z是 X_{m-1}和Y的一个CS

下证：Z是X_{m-1}和Y的LCS

（反证）若不然，则存在长度>k的CS序列W，

显然，W也是X和Y的CS，但其长度>k，矛盾。

(3)若$x_m <> y_n$且$z_k <> y_n$， ==> Z是X和Y_{n-1}的一个LCS；

(3)与(2)对称，类似可证。

综上，定理15.1证毕。

□

求LCS的step2

- Step2：子问题的递归解

  —定理15.1将X和Y的LCS分解为：

  (1)if $x_m=y_n$ then　　//解一个子问题

  找X_{m-1}和Y_{n-1}的LCS；

  (2)if $x_m<>y_n$ then　　//解二个子问题

  找X_{m-1}和Y的LCS 和 找X和Y_{n-1}的LCS；

  取两者中的最大的；

  —c[i, j]定义为X_i和Y_j的LCS长度，i=0~m, j=0~n；

$$c[i,j]=\begin{cases}0 & i=0\ or\ j=0\\ c[i-1,j-1]+1 & i,j>0\ and\ x_i=y_j\\ \max\{c[i,j-1],c[i-1,j]\} & i,j>0\ and\ x_i<>y_j\end{cases}$$

求LCS的step3（1）

- Step3：计算最优解值

  —数据结构设计

  c[0..m, 0..n]　　//存放最优解值，计算时行优先

  b[1..m, 1..n]　　//解矩阵，存放构造最优解信息

  b[i, j]确定：

  ↖　　如果c[i, j]由c[i-1, j-1]确定

  ↑　　如果c[i, j]由c[i-1, j]确定

  ←　　如果c[i, j]由c[i, j-1]确定

  当构造解时，从b[m,n]出发，上溯至i=0或j=0止

  上溯过程中，当b[i,j]包含“↖”时打印出xi(yj)

求LCS的step3（2）

- Step3：计算最优解值

—算法

```
LCS_Length(X, Y)
{    m ←length[X];  n ←length[Y];
     for  i←0  to m  do  c[i,0] ←0;     //0列
     for  j←0  to n  do  c[0,j] ←0;     //0行
     for  i←1  to m  do
          for j←1  to n  do
               if xi=yj  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] ←
"↑"; }    //由Xi-1和Yj确定
                    else
                         {   c[i, j]←c[i, j-1]; b[i, j] ←
"←"; }    //由Xi和Yj-1确定
       return  b and  c;
}
```

时间 θ(mn)

$$c[i,j]=\begin{cases}0 & i=0\ or\ j=0\\ c[i-1,j-1]+1 & i,j>0\ and\ x_i=y_j\\ \max\{c[i,j-1],c[i-1,j]\} & i,j>0\ and\ x_i<>y_j\end{cases}$$

Example

i	j	0 y_j	1 B	2 D	3 C	4 A	5 B	6 A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

求LCS的step4

- Step4：构造一个LCS

一算法

```
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);
 }
```

时间： θ (m+n)

设计技巧

- 动态规划的设计技巧：阶段的划分、状态的表示和存储表的设计；
- 在动态规划的设计过程中，阶段的划分和状态的表示是其中重要的两步，这两步会直接影响该问题的计算复杂性和存储表设计，有时候阶段划分或状态表示的不合理还会使得动态规划法不适用。
- 记忆型递归——动态规划的变种

  —每个子问题的解对应一表项；

  —每表项初值为一特殊值，表示尚未填入；

  —递归时，第一次遇到子问题进行计算并填表，以后查表取值；

存在的问题

- 问题的阶段划分和状态表示，需要具体问题具体分析，没有一个清晰明朗的方法；
- 空间溢出的问题，是动态规划解决问题时一个普遍遇到的问题；
  - 动态规划需要很大的空间以存储中间产生的结果，这样可以使包含同一个子问题的所有问题共用一个子问题解，从而体现动态规划的优越性，但这是以牺牲空间为代价的，为了有效地访问已有结果，数据也不易压缩存储，因而空间矛盾是比较突出的。

- 作业：（1）15.2-3, 15.2-5， 15.4-2

  （2）查Hu和Shing(1980,1982,1984) O(nlogn)算法的资料，写一个该算法的介绍

习题

- 1、 钢条切割问题中，假设现在改变条件，钢条每切割一次，代价为c，那么最佳的决绝方案是什么呢？
- 2、修改MEMORIZED-CUT-ROD，使之不仅返回最优收益，还返回切割方案。
- 3、用动态规划实现斐波那契数F(n) = F(n-1) + F(n-2)
- 4、写一个递归调用的带有记忆功能的LCS算法