第5章 动态规划法

《人工智能算法》

清华大学出版社
2022年7月

提纲

- 引例
- 动态规划法的基本思想
- 动态规划法的适用条件
- 矩阵连乘问题
- 0-1背包问题
- 总结

引例 (1)

Fibonacci序列1, 1, 2, 3, 5, 8, 13, … 递归定义为：

$$f(n)=\begin{cases}1 & n=1,2\\ f(n-1)+f(n-2) & n\geq 3\end{cases}$$

- **分治算法（递归）：**

<u>计算$f(n)$步骤：</u>

```
if (n=1) or (n=2) then
   return 1
else
   return f(n–1)+f(n–2)
end If
```

- **展开递推式：**

$$\begin{aligned}&f(n)\\&=f(n-1)+f(n-2)\\&=2f(n-2)+f(n-3)\\&=3f(n-3)+2f(n-4)\\&=5(f-4)+3f(n-5)\end{aligned}$$

- 算法简洁明了☺
- 对过程重复调用☹
- 重复调用数量巨大☹
- $T(n)$为n的指数
- 不是有效的算法！

线性时间的算法：从$f(1)$自底向上计算直到$f(n)$？

引例 (2)

步骤1： 用$f(n)$存储Fibonacci数列中第n个数的值；

步骤2：

$$f(n)=\begin{cases}1 & n=1,2\\ f(n-1)+f(n-2) & n\ge 3\end{cases}$$

步骤3： 以自底向上的方法计算

n	0	1	2	3	4	5	6	7	8	9	10
$f(n)$	0	1	1	2	3	5	8	13	21	34	55

步骤4： 在数组中分析构造出问题的解

算法：

$A[0] \leftarrow 0$; $A[1] \leftarrow 1$
for $i \leftarrow 2$ to n do
  $A[i] \leftarrow A[i-1]+A[i-2]$
return $A[n]$

时间复杂度：
$O(n)$

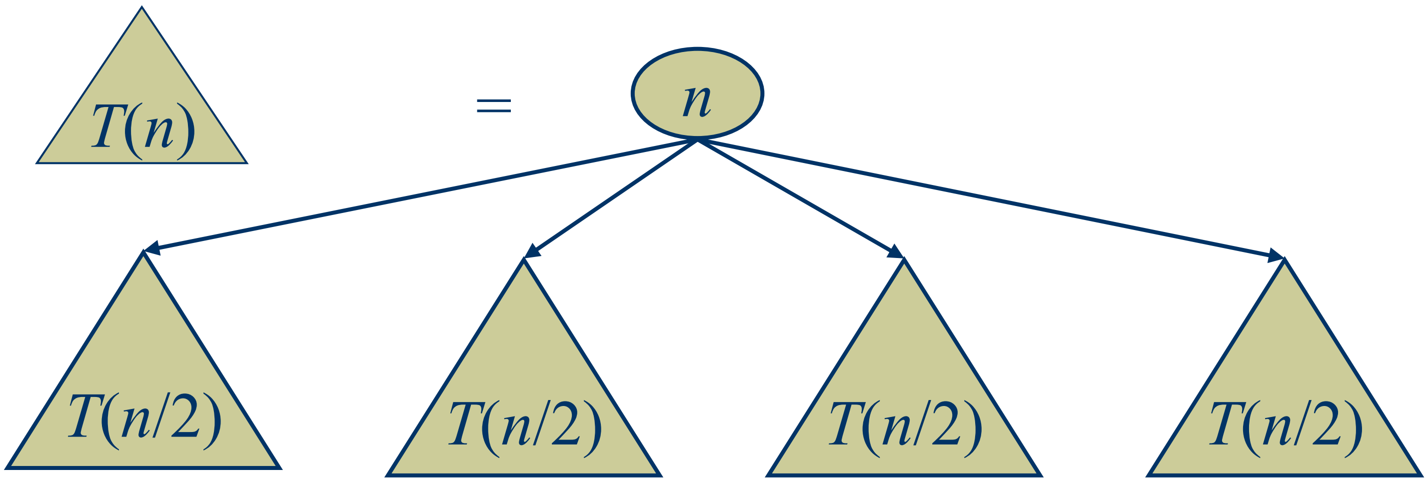
提纲

- 引例
- 动态规划法的基本思想
- 动态规划法的适用条件
- 矩阵连乘问题
- 0-1背包问题
- 总结

动态规划法的基本思想 (1)

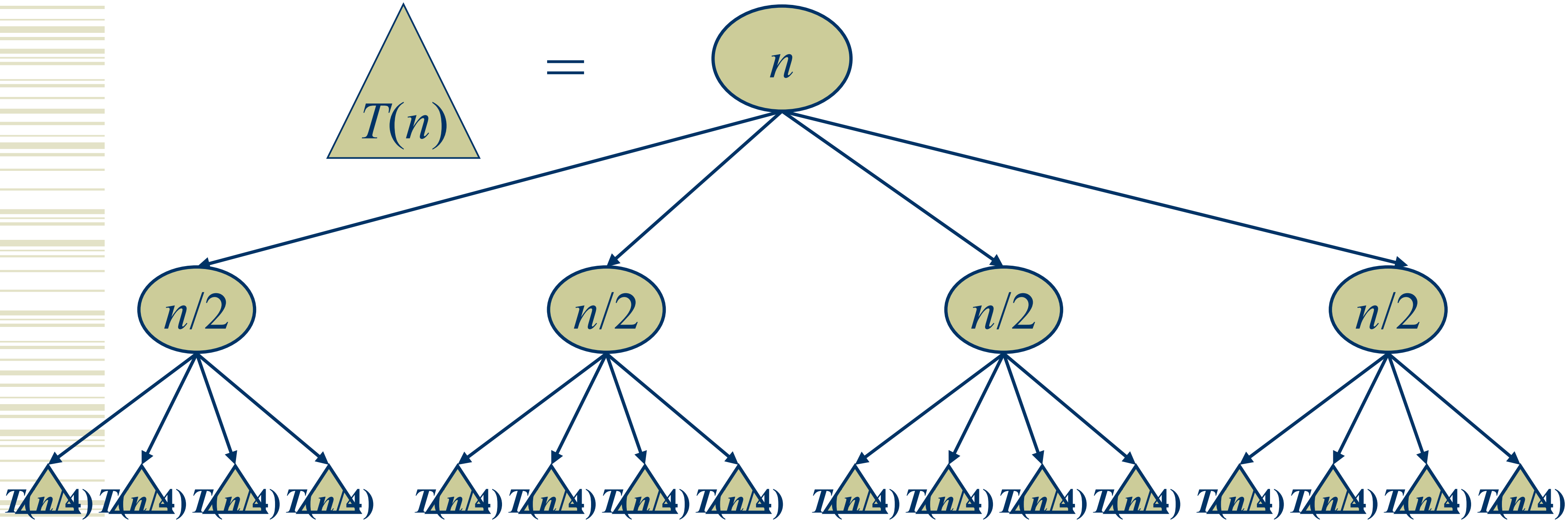
动态规划法（Dynamic Programming）

与分治法类似，其基本思想也是将待求解问题分解成若干个子问题



动态规划法的基本思想 (2)

- 经分解得到的子问题往往不是互相独立的
- 不同子问题的数目常常只有多项式数量级
- 在用分治法求解时，有些子问题被重复计算了许多次



动态规划法的基本思想 (3)

- 保存已解决的子问题的答案，在需要时再找出已求得的答案
- 利用已得到的小规模问题的答案构造待求解的大规模问题的答案
- 可以避免大量重复计算，从而得到多项式时间算法

Those who cannot remember the past are doomed to repeat it.

——George Santayana,
The life of Reason,
Book I: Introduction and
Reason in Common
Sense (1905)

思想，就像幽灵一样……在它自己解释自己之前，必须先告诉它些什么

——查尔斯.狄更斯《董贝父子》

提纲

- 引例
- 动态规划法的基本思想
- 动态规划法的适用条件
- 矩阵连乘问题
- 0-1背包问题
- 总结

动态规划的适用条件 (1)

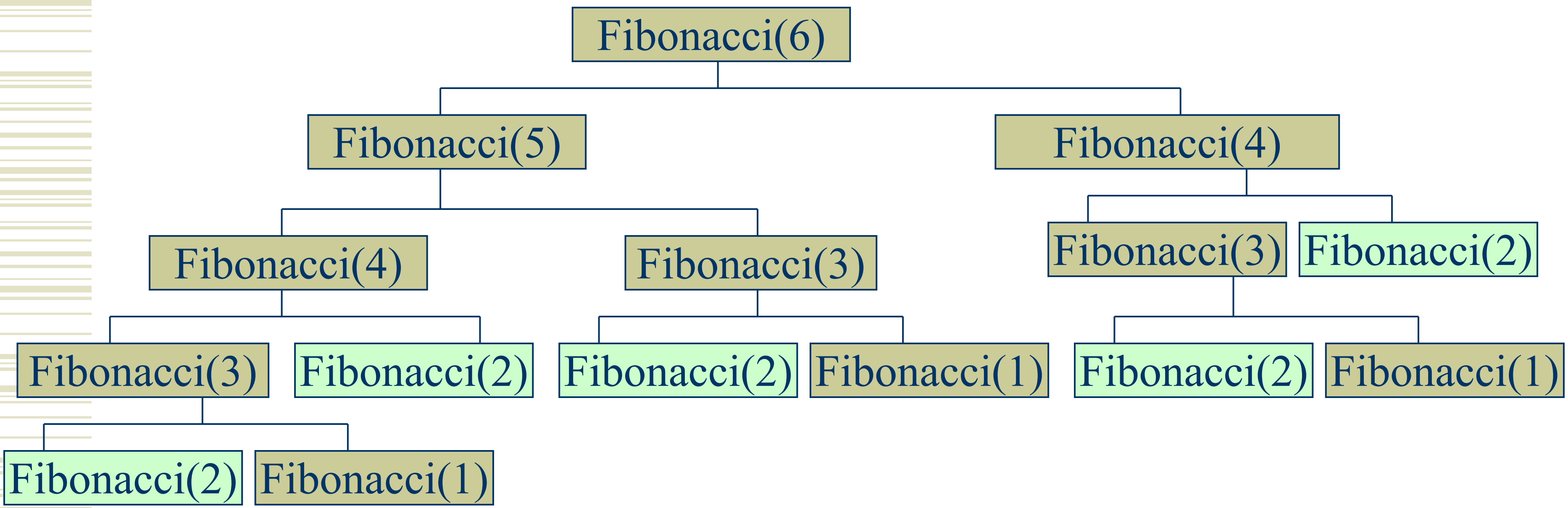
1、最优子结构

- 问题的最优解包含了其子问题的最优解（多阶段决策）
- 最优子结构是问题能用动态规划算法求解的前提
- 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值逐步构造出整个问题的最优值（自顶向下得到最优解）
- 同一个问题可以有多种方式刻划它的最优子结构

2、重叠子问题

- 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
- 对每一个子问题只解一次，自底向上递归求值，并把中间结果存储起来以便以后用来计算所需要的解
- 通常不同的子问题个数随问题的大小呈多项式增长（多项式时间）

动态规划法的适用条件 (2)



动态规划法对许多组合优化问题特别有效！☺

动态规划法的基本步骤

- 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（**填表**）
- 根据计算最优值时得到的信息，构造最优解

提纲

- 引例
- 动态规划法的基本思想
- 动态规划法的适用条件
- 矩阵连乘问题
- 0-1背包问题
- 总结

矩阵连乘问题 (1)

- **引例**

  利用标准的矩阵乘法计算矩阵$M_1(2\times10)$, $M_2(10\times2)$, M_3 (2×10)的乘积

  (1) $(M_1M_2)M_3$: $2\times10\times2+2\times2\times10=80$ 次乘法

  (2) $M_1(M_2M_3)$: $2\times10\times10+10\times2\times10=400$ 次乘法

  结论：不同的乘法执行顺序，乘法次数相差很大！

- **矩阵连乘问题**

  给定n个矩阵$\{A_1, A_2, \ldots, A_n\}$，其中$A_i$和$A_{i+1}$可乘, $i=1, 2, \ldots, n-1$, 确定这n个矩阵乘积的计算次序，使得所需乘法次数最少

  说明：

  - 矩阵乘法满足结合律，连乘的计算次序可由加括号方式确定

  - 计算次序完全确定——完全加括号——按此次序进行2个矩阵相乘

矩阵连乘问题 (2)

- **穷举搜索法**

$$(A_1A_2\cdots A_k)\times(A_{k+1}A_{k+2}\cdots A_n)$$

设前k个矩阵有$P(k)$种加括号方式，对每一个k，有$P(k)P(n-k)$种加括号方式

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

$$P(n)=\frac{1}{n}C_{n-1}^{2n-2}=\frac{(2n-2)!}{n((n-1)!)^2}\approx\frac{4^n}{4\sqrt{\pi}n^{1.5}}$$

1,1,2,5,14,42,132,429,1430,4862,16796, …

$P(n)$随n呈指数增长！ ☹

矩阵连乘问题 (3)

- **递推关系式**

  $A_iA_{i+1}\ldots A_j$记为$A[i{:}\,j]$, 最少乘法次数记为$m(i,j)$, A_i的维数为$p_{i-1}\times p_i$

  计算次序:$(A_iA_{i+1}\ldots A_k)\times(A_{k+1}A_{k+2}\ldots A_j)$

  - **计算量**

    计算$A[i{:}\,k]$的耗费 + 计算$A[k+1{:}\,j]$的耗费 + $A[i{:}\,k]$乘$A[k+1{:}\,j]$的耗费

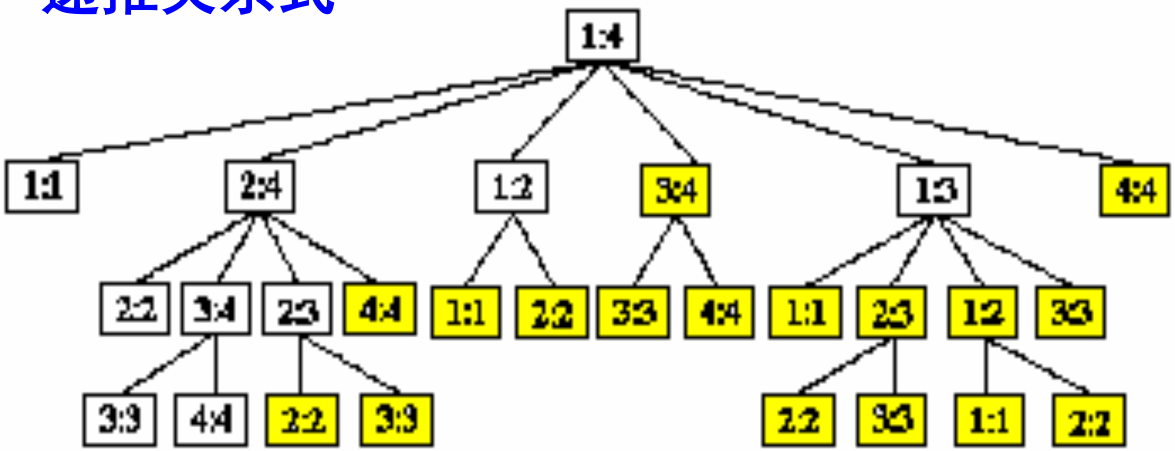
  - **最优子结构性质**

    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

  - **重叠子问题性质?**

矩阵连乘问题 (4)

- **递推关系式**



子问题：

i, j的不同组合:

最多$\Theta(n^2)$个

$m[i, j]$的递推关系式:

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

$$m[1,n]=\min_{1\le k<n}\{m[1,k]+m[k+1,n]+p_0p_kp_n\}$$

矩阵连乘问题 (5)

- 例如：若n=6, $m(2,5)$为以下三个耗费的最小值：
  - $m(2,2)+m(3,5)+p_1 \times p_2 \times p_5$
  - $m(2,3)+m(4,5)+p_1 \times p_3 \times p_5$
  - $m(2,4)+m(5,5)+p_1 \times p_4 \times p_5$

m(1,1)	m(1,2)	m(1,3)	m(1,4)	m(1,5)	m(1,6)
	m(2,2)	m(2,3)	m(2,4)	**m(2,5)**	m(2,6)
		m(3,3)	m(3,4)	m(3,5)	m(3,6)
			m(4,4)	m(4,5)	m(4,6)
				m(5,5)	m(5,6)
					m(6,6)

- 考虑两个方向：
  - $m(i, i) \rightarrow m(i, j\text{-}1)$
  - $m(i+1, j) \rightarrow m(j, j)$
- 计算：
  - $m(i, i), m(i+1, j) \rightarrow$
    $m(i, j\text{-}1), m(j, j)$
- 决策：
  $\min\{m(i, j)\}, i \le k < j$

矩阵连乘问题 (6)

- **依据其递归式以自底向上的方式进行计算（最优值）**

matrixChain ($p[1..n+1]$, $m[1..n][1..n]$, $s[1..n][1..n]$)

```
n←p.length−1
for i=1 to n do
   m[i][i]←0
end for
for r=2 to n do
  for i=1 to n−r+1 do
     j←i+r−1
     m[i][j]←m[i+1][j]+p[i−1]*p[i]*p[j]
     s[i][j]←i
     for k=i+1 to j do
        t←m[i][k]+m[k+1][j]+p[i−1]*p[k]*p[j]
        if t<m[i][j] then
          m[i][j]←t
          s[i][j]←k
        end if
      end for
    end for
end for
```

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

$(A_i\ldots A_k)(A_{k+1}\ldots A_j),\ i<k<j$

矩阵连乘问题 (7)

A_1	A_2	A_3
30×35	35×15	15×5
A_4	A_5	A_6
5×10	10×20	20×25

$\{m(i,j)\}$

	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

- **计算时间：** 设一次乘法的代价为c. 那么

$$T(n)=\sum_{r=2}^{n}\sum_{i=1}^{n-r-1}\sum_{k=1}^{r-1}c=\Theta(n^3)$$

- **所需空间：** $\Theta(n^2)$

提纲

- 引例
- 动态规划法的基本思想
- 动态规划法的适用条件
- 矩阵连乘问题
- 0-1背包问题
- 总结

0-1背包问题 (1)

- **问题**
  - 给定n种物品和一背包。物品 i的重量是w_i，其价值为v_i，背包容量为C。问应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?
  - 特殊的整数规划问题：求一个n元0-1向量$\{x_1, x_2, \ldots, x_n\}$

目标：

$$\max \sum_{i=1}^{n} v_i x_i$$

约束条件：

$$\begin{cases} \sum_{i=1}^{n} w_i x_i \le C \\ x_i \in \{0,1\}, 1 \le i \le n \end{cases}$$

0-1背包问题 (2)

- **最优子结构性质**

  - 设$m(i, j)$为背包剩余容量为j时考虑装入1~i种物品的最大价值

  - $m(i, j)$是下面两个量的最大值 (考虑物品 i)：

  (1) $m(i–1, j)$: 在容量为j的背包中装入1~i–1的物品，不装入物品 i价值最大

  (2) $m(i–1, j–w_i)+v_i$:必装入物品 i，在容量为$j–w_i$的背包中装入1~i–1的物品的最大价值，再加上物品 i的价值v_i ($j \geq w_i$)

- **递推式**

$$m(i, j) = \begin{cases} \max\{m(i-1, j), m(i-1, j-w_i)+v_i\} & j \geq w_i \\ m(i-1, j) & 0 \leq j < w_i \\ 0 & i = 0 \text{或} j = 0 \end{cases}$$

0-1背包问题 (3)

- 用一个$(n+1)\times(C+1)$的矩阵(表)来计算$m(i, j)$, 逐行填表
- 算法: knapsack

  输入:

  n种物品的重量和价值:

  $\{w_1, w_2, \ldots, w_n\}$,

  $\{v_1, v_2, \ldots, v_n\}$;

  背包容量C

  输出: $m(n, C)$

计算时间: ***O(nC)***

伪多项式时间!

算法:

```
for  i=0 to n do
  m[i, 0]←0
end for
for  j=0 to C do
  m[0, j]←0
end for
for i=1 to n do
  for j=1 to C do
    m[i, j]←m[i−1, j]
  if w_i≤j then
   m[i, j]←max{m[i, j], m[i−1, j−w_i]+v_i}
  end if
 end for
end for
return m[n, C]
```

0-1背包问题 (4)

- 例如：若背包容量C=9, 4种物品的重量和价值分别为{2,3,4,5}和{3,4,5,7}, 尽可能将物品装入背包，并使总价值最大。

5×10的表：

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	3	3	3	3	3	3	3	3
2	0	0	3	4	4	7	7	7	7	7
3	0	0	3	4	5	7	8	9	9	12
4	0	0	3	4	5	7	8	10	11	12

最优值：最大价值为 12

最优解：装入物品1,2,3; 装入物品3,4

$m(i, j)$的计算只与$m(i–1,0)$~$m(i–1, j)$的值相关

0-1背包问题 (5)

- **注意到：**

- 求解给定问题时，有些较小子问题的解通常并不需要

  （填表时，i和j都以1递增）

- 自底向上：只有背包容量增加到能装入一个物品时，价值才增加（跃变）

  自顶向下：递归求解，子问题重复，效率低

- **带记忆功能：** 自顶向下递归求解+自底向上表格

```
MFKnapsack(i, j)  //调用MFKnapsack(n, C)
  // 数组w[1..n]、v[1..n]、表m[0..n, 0..C]是全局变量
   初始化 m[0..n, 0..C]←–1；  m[0, 0]←0
  if  m[i, j] < 0 then // 未计算，递归计算m[i, j]；否则，查表得m[i, j]
      if j<w_i then m[i, j]←MFKnapsack(i-1, j)
      else
         m[i, j]←max(MFKnapsack(i–1, j),  v_i+MFKnapsack(i–1,  j–w_i) )
  return m[i,  j]  // 直接返回结果（>=0,查表）或计算结果(<0)
```

0-1背包问题 (6)

- **讨论**

- 带记忆功能算法的效率与自底向上算法效率类型一样，提高效率不会超过一个常数因子

- 填表的空间开销较大，如何优化？

- $O(nC)$伪线性时间复杂度，人们不希望复杂度与C有关，如何处理？

- 考虑近似求解，如何设计算法（贪心算法）？

提纲

- 引例
- 动态规划法的基本思想
- 动态规划法的适用条件
- 矩阵连乘问题
- 0-1背包问题
- 总结

总结

- 动态规划的基本思想、适用条件，所解决问题的主要特征
- 动态规划方法解决问题的一般方法和步骤、多阶段决策问题的特征和最优化原理
- 动态规划的重要算法实例：

  \- 矩阵连乘问题的动态规划算法

  \- 0-1背包问题的动态归划算法

结语

谢谢！