算法设计与分析

蒋婷婷

上节课回顾

- 分治策略
  - 一般性选择问题
  - 卷积

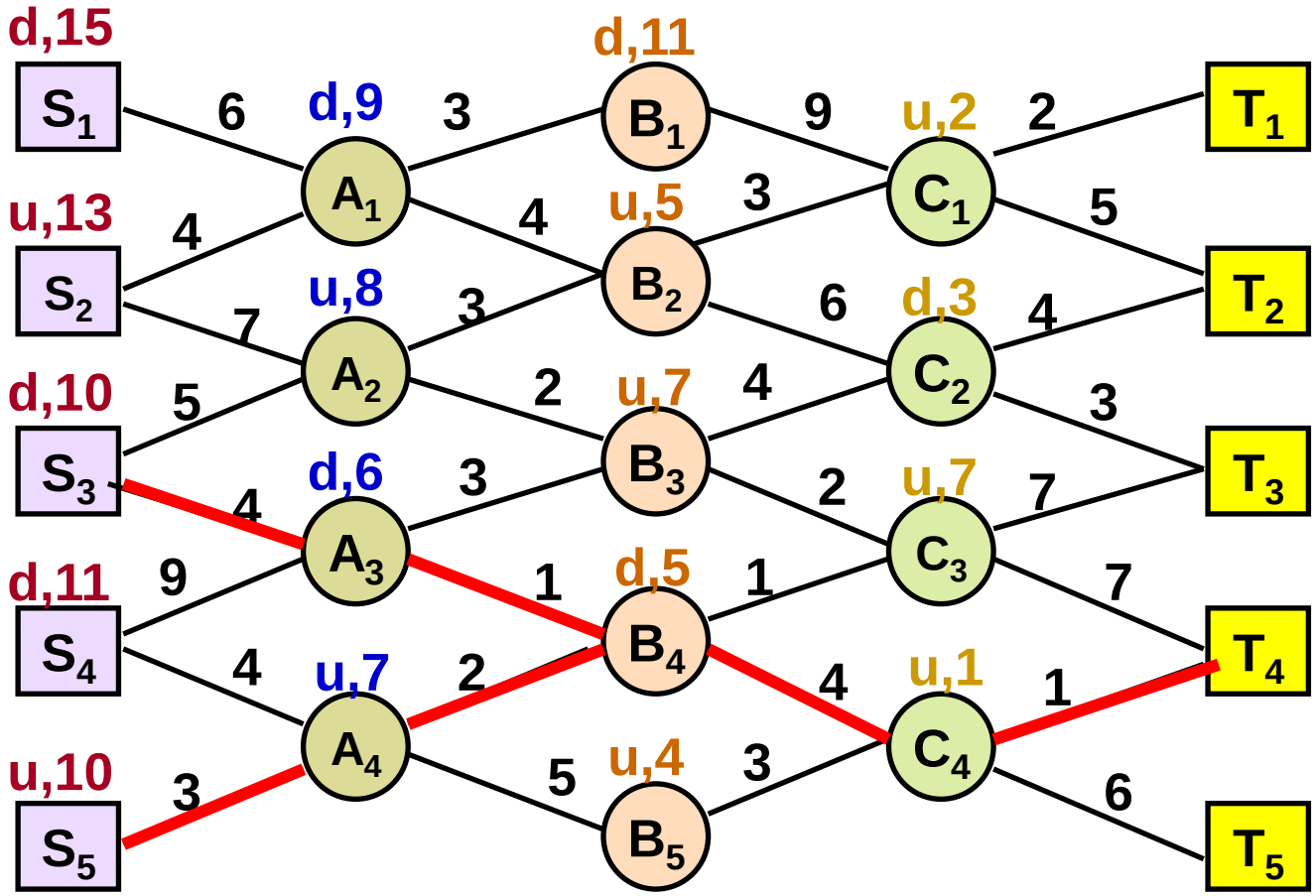
动态规划

动态规划 **(Dynamic Programming)**

- 基本思想和使用条件
- 动态规划算法的设计步骤
- 应用实例
- 小结

基本思想和使用条件

例1　求从始点到终点的最短路径



基本思想

解：判断序列

$$F(C_l)=\min_m\{C_lT_m\}$$

$$F(B_k)=\min_l\{B_kC_l+F(C_l)\}$$

$$F(A_j)=\min_k\{A_jB_k+F(B_k)\}$$

$$F(S_i)=\min_j\{S_iA_j+F(A_j)\}$$

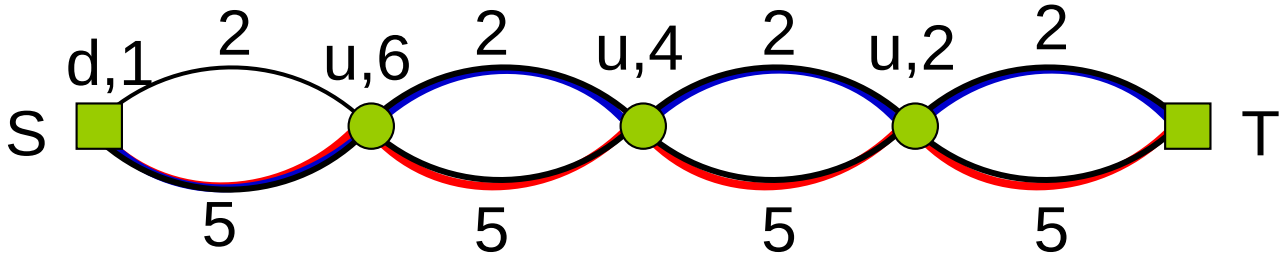
任何最短路径的子路径都是相对于子路径的始点和终点的最短路径

为找一条最短路径只需从T_j开始进行多步判断

使用条件:优化原则

一个最优决策序列的任何子序列本身一定是相对于子序列的初始和结束状态的最优的决策序列

例2 求总长模10的最小路径



最优解：下、下、下、下

动态规划求解：下、上、上、上

不满足优化原则，不能使用动态规划设计技术

算法设计步骤

例3 矩阵乘法： 设$A_1,A_2,\ldots,A_n$为矩阵序列，A_i为$P_{i-1}\times P_i$阶矩阵，$i = 1,2,\ldots,n$. 确定乘法顺序使得元素相乘的总次数最少.

输入：向量$P = <P_0, P_1, \ldots, P_n>$

实例： $P = <10, 100, 5, 50>$

A_1: 10×100, A_2: 100×5, A_3: 5×50,

乘法次序

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

$A_1(A_2\ A_3)$: $10 \times 100 \times 50 + 100 \times 5 \times 50 = 75000$

搜索空间的规模

一般算法：n次运算加括号方法有 $\frac{1}{n+1}\binom{2n}{n}$ 种，Catalan数

$$W(n)=\Omega\left(\frac{1}{n+1}\frac{(2n)!}{n!n!}\right)=\Omega\left(\frac{1}{n+1}\frac{\sqrt{2\pi 2n}\left(\frac{2n}{e}\right)^{2n}}{\sqrt{2\pi n}\left(\frac{n}{e}\right)^{n}\sqrt{2\pi n}\left(\frac{n}{e}\right)^{n}}\right)$$

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

动态规划算法

输入$P=<P_0, P_1, \ldots, P_n>$，$A_{i..j}$ 表示乘积$A_iA_{i+1}\ldots A_j$ 的结果，其最后一次相乘是

$$A_{i..j} = A_{i..k}\, A_{k+1..j},$$

$m[i,j]$ 表示得到 $A_{i..j}$ 的最少的相乘次数

递推方程

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

为了确定加括号的次序，设计表 $s[i,j]$,记录求得最优时最后一次运算的位置

算法1：递归实现

算法1 RecurMatrixChain(*P*,*i*,*j*)

1. *m*[*i*,*j*]←∞

2. *s*[*i*,*j*]←*i*

3. for *k*←*i* to *j*–1 do

4. *q* ← RecurMatrixChain(*P*,*i*,*k*)

+ RecurMatrixChain(*P*,*k*+1,*j*) + $p_{i-1} p_k p_j$

5. if *q*<*m*[*i*,*j*]

6. then *m*[*i*,*j*]←*q*

7. *s*[*i*,*j*]←*k*

8. Return *m*[*i*,*j*]

这里没有写出算法的全部描述（递归调用的初值等）

递归实现的复杂性

复杂性满足递推关系

$$T(n) \geq \begin{cases} O(1) & n = 1 \\ \sum_{k=1}^{n-1} (T(k) + T(n-k) + O(1)) & n > 1 \end{cases}$$

$$T(n) \geq O(n) + \sum_{k=1}^{n-1} T(k) + \sum_{k=1}^{n-1} T(n-k) = O(n) + 2\sum_{k=1}^{n-1} T(k)$$

数学归纳法证明 $T(n) \geq 2^{n-1}$

$n=2$, 显然为真

假设对于任何小于n 的 k 命题为真，则

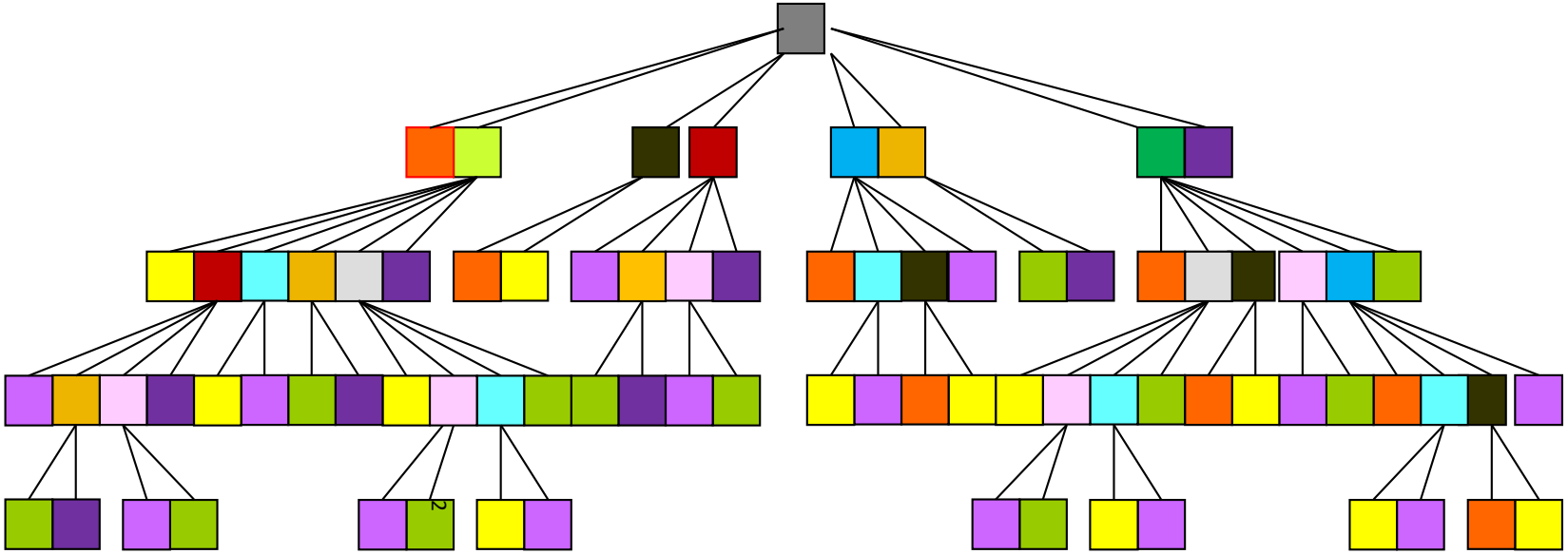
$$T(n) \geq O(n) + 2\sum_{k=1}^{n-1} T(k) \geq O(n) + 2\sum_{k=1}^{n-1} 2^{k-1}$$

$$= O(n) + 2(2^{n-1} - 1) \geq 2^{n-1}$$

复杂性高的原因：子问题重复计算

***n*=5，计算子问题：81个；不同的子问题：15个**

子问题	1-1	2-2	3-3	4-4	5-5	1-2	2-3	3-4	4-5	1-3	2-4	3-5	1-4	2-5	1-5
数	8	12	14	12	8	4	5	5	4	2	2	2	1	1	1



算法2：迭代实现

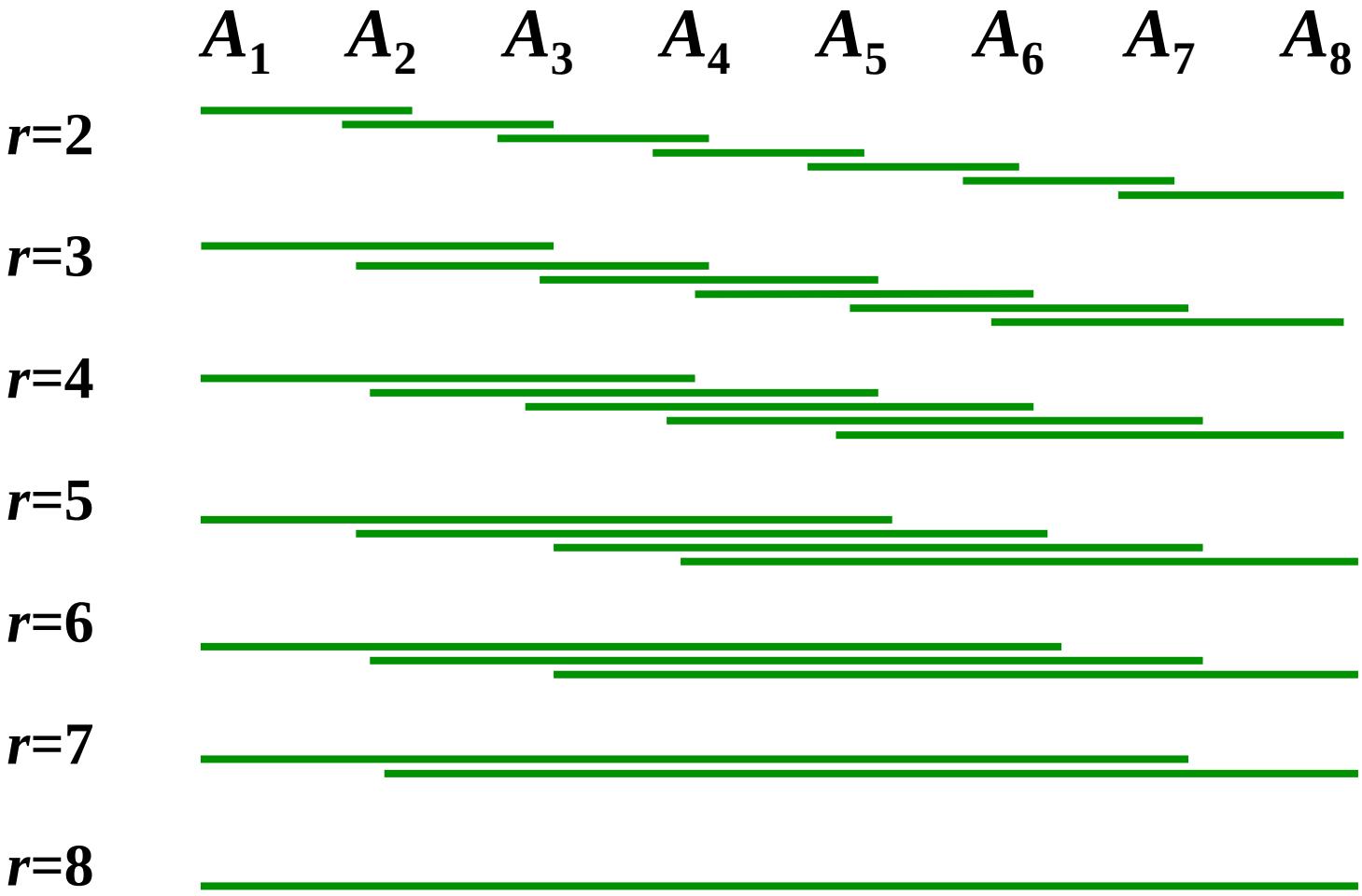
算法2 MatrixChain(P,n)

1. 令所有的 $m[i,i]$初值为0
2. for $r \leftarrow 2$ to n do　　// r为计算的矩阵链长度//
3. 　for $i \leftarrow 1$ to $n-r+1$ do　　//$n-r+1$为最后r链的始位置//
4. 　　$j \leftarrow i+r-1$　　// 计算链i—j //
5. 　　$m[i,j] \leftarrow m[i+1,j] + p_{i-1}*p_i*p_j$　　// $A_i(A_{i+1}..A_j)$//
6. 　　$s[i,j] \leftarrow i$　　//记录分割位置//
7. 　　for $k \leftarrow i+1$ to $j-1$ do
8. 　　$t \leftarrow m[i,k]+m[k+1,j]+ p_{i-1}*p_k*p_j$　　//$(A_i..A_k)(A_{k+1}..A_j)$//
9. 　　if $t<m[i,j]$
10. 　　then $m[i,j] \leftarrow t$
11. 　　$s[i,j] \leftarrow k$

复杂性：行2,3,7循环进行都是$O(n)$，循环内为$O(1)$，

$W(n)=O(n^3)$

迭代过程的一个实例



实例

输入 P= <30, 35, 15, 5, 10, 20>, n=5，

矩阵链：$A_1 A_2 A_3 A_4 A_5$，其中

A_1:30×35，A_2:35×15，A_3:15×5，A_4:5×10，A_5:10×20

r=1	m[1,1]=0	m[2,2]=0	m[3,3]=0	m[4,4]=0	m[5,5]=0
r=2	m[1,2]=15750	m[2,3]=2625	m[3,4]=750	m[4,5]=1000	
r=3	m[1,3]=7875	m[2,4]=4375	m[3,5]=2500		
r=4	m[1,4]=9375	m[2,5]=7125			
r=5	m[1,5]=11875				

r=2	s[1,2]=1	s[2,3]=2	s[3,4]=3	s[4,5]=4	
r=3	s[1,3]=1	s[2,4]=3	s[3,5]=3		
r=4	s[1,4]=3	s[2,5]=3			
r=5	s[1,5]=3				

解：　$(A_1 (A_2 A_3)) (A_4 A_5)$

两种算法的比较

递归算法：复杂性高

非递归算法：复杂性较低

原因

- 递归动态规划算法的子问题被多次重复计算
- 子问题计算次数呈指数增长
- 非递归动态规划算法每个子问题只计算一次
- 子问题的计算随问题规模成多项式增长

动态规划算法设计步骤

- 将问题表示成多步判断，确定子问题的边界
  - 整个判断序列就对应问题的最优解
  - 每步判断对应一个子问题，子问题类型与原问题一样
- 确定优化函数，以函数的极大(或极小)作为判断的依据
- 确定是否满足优化原则-----动态规划算法的必要条件
- 确定子问题的重叠性
- 列出关于优化函数的递推方程(或不等式)和边界条件
- 自底向上计算子问题的优化函数值
- 备忘录方法(表格)存储中间结果
- 设立标记函数$s[i,j]$求解问题的解

投资问题

m 元钱，n 项投资，$f_i(x)$: 将 x 元投入第 i 项项目的效益

目标函数　$\max\{f_1(x_1) + f_2(x_2) + \ldots + f_n(x_n)\}$

约束条件　$x_1 + x_2 + \ldots + x_n = m$，$x_i \in \mathbf{N}$

实例：5万元钱，4个项目，效益函数如下表所示

x	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$
0	0	0	0	0
1	11	0	2	20
2	12	5	10	21
3	13	10	30	22
4	14	15	32	23
5	15	20	40	24

算法设计

设$F_k(x)$ 表示 x元钱投给前k 个项目的最大效益

多步判断

假设知道 p 元钱（$p \le x$）投给前 $k-1$个项目的最大效益，决定 x 元钱投给前 k 个项目的分配方案

递推方程和边界条件

$$F_k(x) = \max_{0 \le x_k \le x} \{ f_k(x_k) + F_{k-1}(x - x_k) \} \quad k > 1$$

$$F_1(x) = f_1(x)$$

算法设计

x	$F_1(x)$ $x_1(x)$	$F_2(x)$ $x_2(x)$	$F_3(x)$ $x_3(x)$	$F_4(x)$ $x_4(x)$
1	11 1	11 0	11 0	20 1
2	12 2	12 0	13 1	31 1
3	13 3	16 2	30 3	33 1
4	14 4	21 3	41 3	50 1
5	15 5	26 4	43 4	61 1

解 $x_1=1,\ x_2=0,\ x_3=3,\ x_4=1\quad F_4(5)=61$

算法的复杂度分析

表中有 m 行 n 列, 共计 mn 项

$$F_k(x)=\max_{0\le x_k\le x}\{f_k(x_k)+F_{k-1}(x-x_k)\}\quad k>1$$

$$F_1(x)=f_1(x)$$

对第$F_k(x)$项 ($2\le k\le n, 1\le x\le m$) 需要 $x+1$次加法, x 次比较

加法次数 $$\sum_{k=2}^{n}\sum_{x=1}^{m}(x+1)=\frac{1}{2}(n-1)m(m+3)$$

比较次数 $$\sum_{k=2}^{n}\sum_{x=1}^{m}x=\frac{1}{2}(n-1)m(m+1)$$

$$W(n)=O(nm^2)$$

3.3.2 背包问题

一个旅行者准备随身携带一个背包. 可以放入背包的物品有 n 种, 每种物品的重量和价值分别为 w_j, v_j. 如果背包的最大重量限制是 b, 怎样选择放入背包的物品以使得背包的价值最大?

目标函数 $\max \sum_{j=1}^{n} v_j x_j$

约束条件 $\sum_{j=1}^{n} w_j x_j \le b, \quad x_j \in \mathbf{N}$

线性规划问题

由线性条件约束的线性函数取最大或最小的问题

整数规划问题 线性规划问题的变量 x_j 都是非负整数

子问题划分、优化函数、标记函数

$F_k(y)$：装前 k 种物品，总重不超过 y，背包的最大价值

$i(k,y)$：装前 k 种物品，总重不超过 y，背包达最大价值时装入物品的最大标号

递推方程、边界条件、标记函数

$$F_k(y)=\max\{F_{k-1}(y),F_k(y-w_k)+v_k\}$$

$$F_0(y)=0,\quad 0\le y\le b,\quad F_k(0)=0,\quad 0\le k\le n$$

$$F_1(y)=\left\lfloor\frac{y}{w_1}\right\rfloor v_1$$

$$F_k(y)=-\infty\quad y<0$$

$$i_k(y)=\begin{cases} i_{k-1}(y) & F_{k-1}(y) > F_k(y-w_k)+v_k \\ k & F_{k-1}(y) \le F_k(y-w_k)+v_k \end{cases}$$

实例计算

$v_1 = 1,\ \ v_2 = 3,\ \ v_3 = 5,\ v_4 = 9,$

$w_1 = 2,\ \ w_2 = 3,\ w_3 = 4,\ w_4 = 7,$

$b = 10$

$F_k(y)$ 的计算表如下：

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

追踪问题的解

k　y	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	1	1	1	1	1	1
2	0	1	2	2	2	2	2	2	2	2
3	0	1	2	3	3	3	3	3	3	3
4	0	1	2	3	3	3	4	3	4	4

在上例中, 求得

$i_4(10)=4 \Rightarrow x_4 \geq 1$

$i_4(10-w_4)=i_4(3)=2 \Rightarrow x_2 \geq 1,\ x_4=1,\ x_3=0$

$i_2(3-w_2)=i_2(0)=0 \Rightarrow x_2 = 1,\ x_1=0$

解 $x_1=0,\ x_2=1,\ x_3=0,\ x_4=1$

追踪解 x_j 的算法

算法 TrackSolution

输入：$i_k(y)$表，其中$k=1,2,\ldots,n$，$y=1,2,\ldots,b$

输出：$x_1, x_2, \ldots, x_n$，n种物品的装入量

1. for $j \leftarrow 1$ to n do
2. $x_j \leftarrow 0$
3. $y \leftarrow b, j \leftarrow n$
4. $j \leftarrow i_j(y)$,
5. $x_j \leftarrow 1$
6. $y \leftarrow y - w_j$
7. while $i_j(y)=j$ do
8. $y \leftarrow y - w_j$
9. $x_j \leftarrow x_j + 1$
10. if $i_j(y) \neq 0$ goto 4