动态规划

陈长建

计算机科学系

关于第二次实验

- **院楼103**，**11月3日**上午8:30-12:00（现场验收）
- 第二次实验离线题（离线准备）
  - 1. 用动态规划法实现0-1背包
  - 2. 半数集问题（实现题2-3）
  - 3. 集合划分问题（实现题2-7）
- 在线题
  - acm.hnu.edu.cn

小班讨论课安排

- 讨论主题分为3类：
  - 算法分析练习
  - 算法实现练习
  - 《数学之美》等课外资料阅读
- 分组选题（每班分6组）
  - 每组选择1位组长，实行组长负责制
  - 每次讨论课，每组选1类题，即每次课参与讨论分析题、实现题与《 数学之美》三类题的各有2组
  - 每组在2次讨论课中，必须轮流选择3类题目

关于小班讨论

- 以组为单位,每组一题,每组人人参与,合理分工,ppt中标记分工,尽量都有代码演示,第8周上台报告

第一次选题
算法分析练习
《数学之美》
算法实现练习
算法分析练习
算法分析练习

第一次选题
算法分析练习
算法实现练习
算法实现练习
算法分析练习

关于小班讨论

- 时间：**本周四（10月31日）**

星期四
算法设计与分析*(课程指导) 陈长建 指导学时:4 40102(8,14周) 中314 上课人数:31 计科2205 重修人数:
算法设计与分析*(课程指导) 陈长建 指导学时:4 40304(8,14周) 中314 上课人数:32 计科2204 重修人数:
算法设计与分析*(课程指导) 陈长建 指导学时:4 40506(8,14周) 中307 上课人数:29 计科2206 重修人数:

关于期中考试

- 时间：第九周周六（**11月9日**）晚上，7点到8点半
- 地点：院楼401
- 范围：
  - 前三部分（绪论、递归与分治、动态规划）
- 题型：
  - 简答题、应用题、算法题
- 形式：笔试

回顾-最长公共子序列

定义 **一个给定序列的子序列是在该序列中删除若干元素后得到的序列。** 即

若给定序列 X={x1,x2,…,xm}，则另一序Z={z1,…,zk}是X的子序列是指:

存在一个严格递增下标序列{i1,i2,…,ik}使得对于所有j=1,2,…,k有：$Z_j=X_{ij}$

例如 序列 X={A，B，C，B，D，A，B}

子序列 Z={B，C，D，B}

相应的递增下标序列为{2，3，5，7}

。

最长公共子序列

公共子序列

给定2个序列 X 和 Y，当另一序列 Z 既是X的子序列又是Y的子序列时，称Z是序列X和Y的公共子序列。

最长公共子序列问题

给定2个序列 X={x1,x2,...,xm}和 Y={y1,y2,...,yn}，找出 X 和 Y 的最长公共子序列。

例如：字符串13455和245576的最长公共子序列为455

字符串acdfg和adfc的最长公共子序列为adf

最长公共子序列

LCS（Longest Common Subsequence）的应用

- 求两个序列中最长的公共子序列算法，广泛的应用在图形相似出路、媒体流的相似比较、计算生物学方面。生物学家常常利用该算法进行基因序列比对，由此推测序列的结构、功能和演化过程。
- LCS可以描述两段文字之间的“相似度”，即它们的雷同程度，从而能够用来辨别抄袭。另一方面，对一段文字进行修改之后，计算改动前后文字的最长公共子序列，将除此子序列外的部分提取出来，这种方法判断修改的部分，往往十分准确。简而言之，百度知道，百度百科都用得上。

回顾 – 最大子段和

给定由N个整数（可能有负整数）组成的序列$a_1,a_2,\ldots,a_n$,求该序列形如 $a_i+a_{i+1}+\ldots+a_j$的子段和的最大值。

当所有整数均为负整数时，定义其最大子段和为0

例如：

当$\{a_1,a_2,\ldots,a_6\}=\{-1，11，-4，13，-5，-2\}$时

其最大子段和为 20

算法

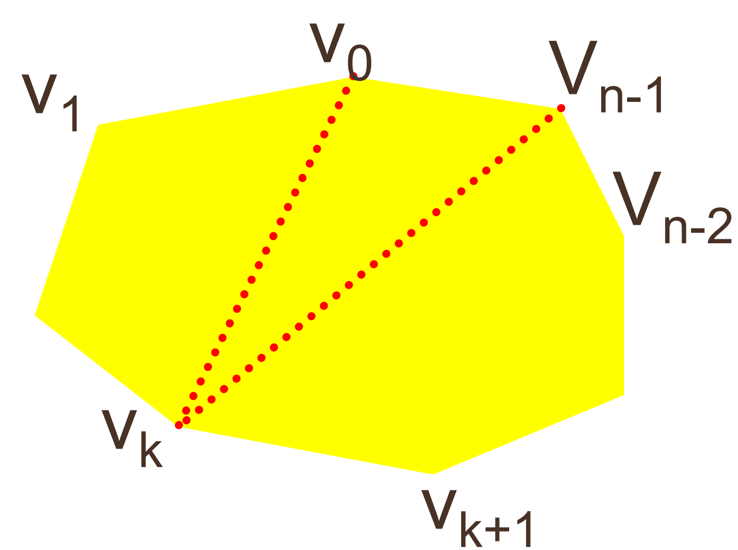
算法1：对所有的(i,j)对，顺序求和$a_i+\ldots+a_j$并比较出最大的和

算法2：分治策略，将数组分成左右两半，分别计算左边的最大和、右边的最大和、跨边界的最大和，然后比较其中最大者

算法3：动态规划

回顾 – 凸多边形最优三角剖分

- **凸多边形：** 用多边形顶点的逆时针序列表示凸多边形，即$P=\{v_0,v_1,\ldots,v_{n-1}\}$表示具有n条边的凸多边形。
- **弦：** 若v_i与v_j是多边形上不相邻的2个顶点，则线段v_iv_j称为多边形的一条弦。弦将多边形分割成2个多边形$\{v_i,v_{i+1},\ldots,v_j\}$和$\{v_j,v_{j+1},\ldots v_i\}$。



概念

- **多边形的三角剖分：**将多边形分割成互不相交的三角形的弦的集合T。

- **凸多边形最优三角剖分：**给定凸多边形P，以及定义在由多边形的边和弦组成的三角形上的权函数w。要求确定该凸多边形的三角剖分，使得该三角剖分中诸三角形上权之和为最小。

本章要点:

- 理解动态规划算法的概念
- 掌握动态规划算法的基本要素
- 掌握设计动态规划算法的步骤
- 通过范例学习动态规划算法设计策略
  - 最大子段和、最长公共子序列、矩阵链连乘、凸多边形最优三角剖分、0-1背包问题

例6 – 流水作业调度

- 问题描述：n个作业{1，2，…，n}要在由2台机器M1和M2组成的流水线上完成加工。每个作业加工的顺序都是先在M1上加工，然后在M2上加工。M1和M2加工作业i所需的时间分别为a_i和b_i。
- 流水作业调度问题：要求确定这n个作业的最优加工顺序，使得从第一个作业在机器M1上开始加工，到最后一个作业在机器M2上加工完成所需的时间最少。

- 一般流水作业调度问题： n个作业{1，2，…，n}要在由m台机器M1、M2、…、Mm组成的流水线上完成加工。要求确定这n个作业的最优加工顺序，使m台机器上作业所需的时间最少。

加工时间矩阵

- n个作业{1，2，…，n}，m台机器M1、M2、…、Mm，设Mi加工作业j所需的时间分别为t_{ij}。
- 加工时间矩阵：T=（t_{ij}）$_{m\times n}$。
- n个作业{1，2，…，n}，2台机器M1、M2的时间矩阵

$$T=\begin{pmatrix} t_{11} & t_{12} & \cdots & t_{1n} \\ t_{21} & t_{22} & \cdots & t_{2n} \end{pmatrix}$$

记为：

$$T=\begin{pmatrix} a_1 & a_2 & \cdots & a_n \\ b_1 & b_2 & \cdots & b_n \end{pmatrix}$$

算法分析：m=2的情况

- 为叙述方便起见，以下设
- 作业集合为$J=\{J_1，J_2，\ldots，J_n\}$

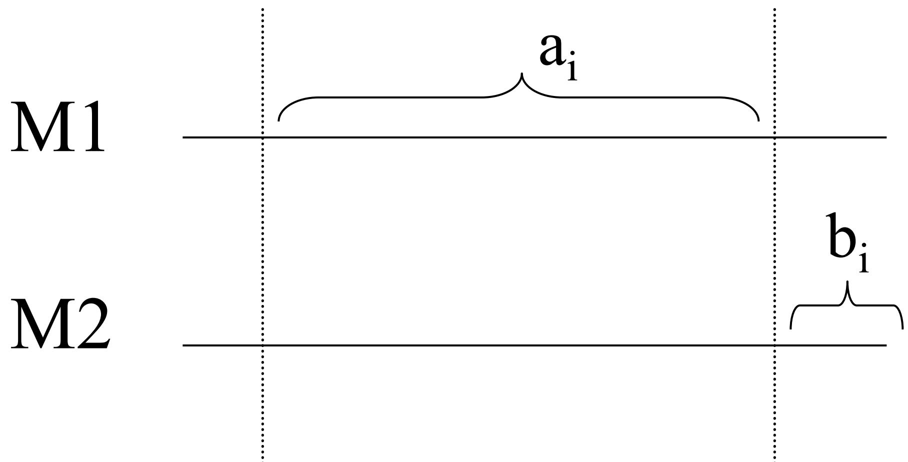
又设 一个子集$S \subseteq J$，$S_0=J$

约定：为叙述方便。有时将作业集J仍用{1，2，3，…，n}表示

分析

- **一个最优调度应使机器M1没有空闲时间，且机器M2的空闲时间最少。(为什么？)**
- **在一般情况下，机器M2上会有机器空闲和作业积压2种情况**

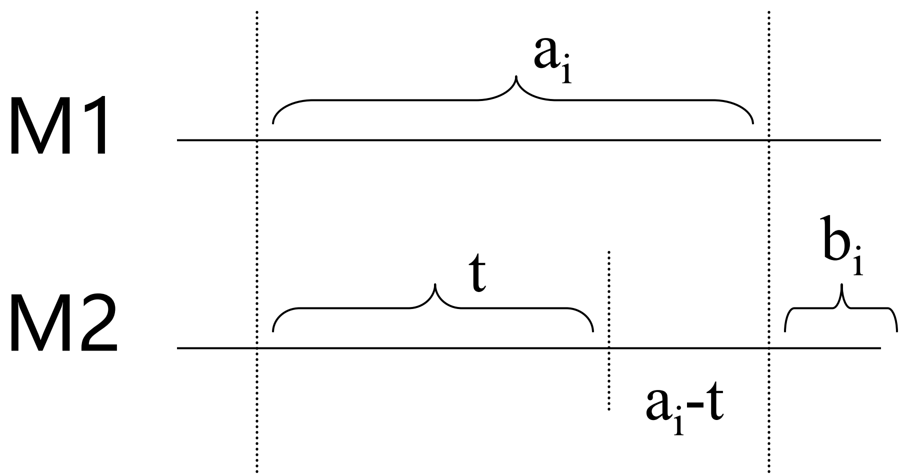
  - 情况



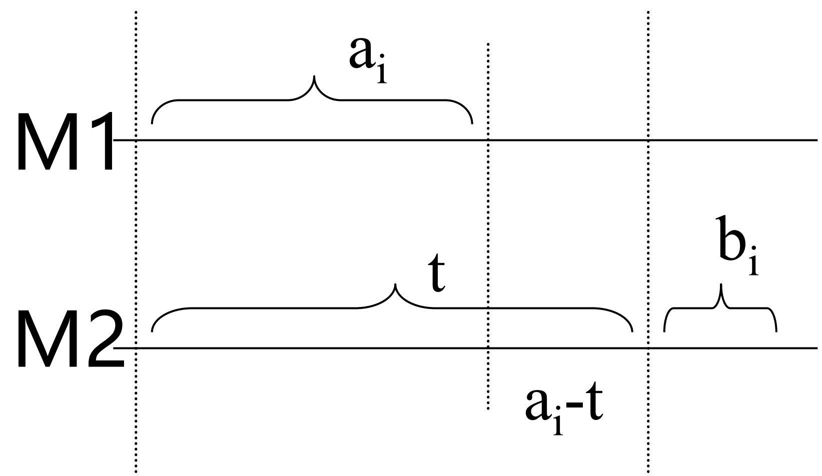
一般情况-在M1做作业i的情况

在M2上未完成前面作业时，M1上做作业i的情况

- 情况1



- 情况2



两个特殊情况

- 如果每个作业在M1上的时间都是小于M2上的时间
  - a=[1,6,8], b=[2,7,9]
  - 最优调度是按照a升序排列

- 如果每个作业在M1上的时间都是大于M2上的时间
  - a=[2,7,9], b=[1,6,8]
  - 最优调度是按照b降序排列

算法分析

- 一般情况
  - 设机器M1开始加工S中作业时，机器M2还在加工其他作业，要等时间 t 后才可使用,完成S中作业（两道工序）所需的最短时间记为T(S,t)。
- **流水作业调度问题变为：求最优值为T(J,0)。**

开始时的情况

- 设π是所给n个流水作业的一个最优调度，即已排好作业调度：$\pi(1)$，$\pi(2)$，…，$\pi(n)$，其中$\pi(k)\in\{1,2,\ldots,n\}$，记$i=\pi(1)$

最优子结构性质

- 设π是所给n个流水作业的一个最优调度，即已排好作业调度：
  $\pi(1)$， $\pi(2)$， …， $\pi(n)$，其中$\pi(k)\in\{1,2,\ldots,n\}$，记$i=\pi(1)$

- 设机器M1开始加工J中第一个作业$J_{\pi(1)}$时，机器M2可能在等待，它所需的加工时间为 $a_{\pi(1)}+T'$，其中T′ 是在机器M2的等待时间为$b_{\pi(1)}$时、安排作业$J_{\pi(2)}$，…，$J_{\pi(n)}$所需的时间，这是最好的时间。

- 记$S=J-\{J_{\pi(1)}\}$，则可证明：

$$T'=T(S,b_{\pi(1)})。$$

$T'=T(S,b_{\pi}(1))$的证明

- 注：$T(S,b_{\pi(1)})$是完成S中所有作业的最少时间
- 证明：由T '的定义知对特定安排π，$T'\geq T(S,b_{\pi(1)})$。
- 若真$T'>T(S,b_{\pi(1)})$，设π'是作业集$S=J-\{J_{\pi(1)}\}$在机器M2的等待时间为$b_{\pi(1)}$情况下的一个最优调度。则π(1)，π' (2)，…，π' (n)是J的一个调度π'，且该调度π'所需的时间为$a_{\pi(1)}+T(S,b_{\pi(1)})<a_{\pi(1)}+T'$。
- 这与π是J的最优调度矛盾。故$T'\leq T(S,b_{\pi(1)})$。从而$T'=T(S,b_{\pi(1)})$。
- 这就证明了流水作业调度问题具有最优子结构的性质。

结论

- $T(J,0)=a_{\pi(1)}+T(J-\{J_{\pi(1)}\},b_{\pi(1)})$

如末知作业$J_{\pi(1)}$是否应该排在第一位，则应考虑所有任务

$$T(J,0)=\min_{\{1\le i\le n\}}\{a_i+T(J-\{J_i\},b_i)\}$$

递归关系

- $T(J,0)=\min_{\{1\le i\le n\}}\{a_i+T(J-\{J_i\},b_i)\}$
- **需要计算：$T(J-\{J_i\},b_i)$**

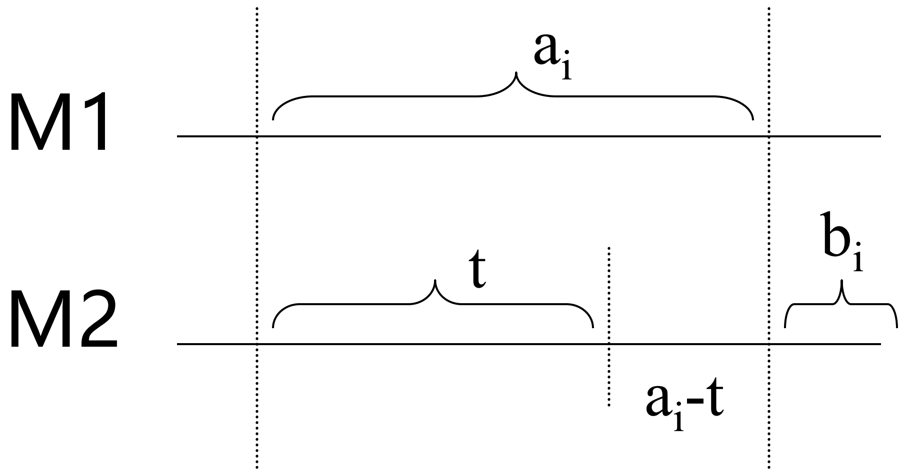
一般情况：设$S\subseteq J$，$S_0=J$，则可证

$$T(S,t)=\min_{J_i\in S}\{a_i+T(S-\{J_i\},b_i+\max(t-a_i,0))\}$$

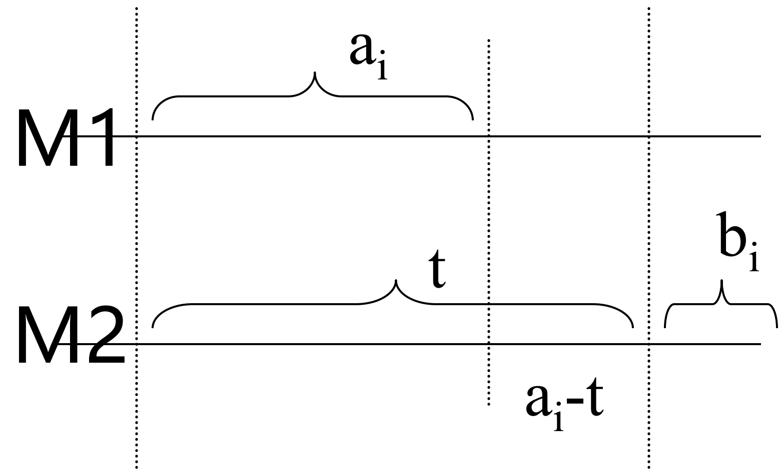
$$T(S,t)=\min\{a_i+T(S-\{J_i\}, b_i+\max(t-a_i,0))\}$$

图例

- 情况1



- 情况2



Johnson不等式

对递归式的深入分析表明，算法可进一步得到简化。

设π是作业集S在机器M2的等待时间为t时的任一最优调度。

若π(1)=i, π(2)=j。则由动态规划递归式可得：

$T(S,t)=a_i+T(S-\{J_i\},b_i+\max\{t-a_i,0\})=a_i+a_j+T(S-\{J_i,J_j\},t_{ij})$

其中，

$$
\begin{aligned}
t_{ij} &= b_j + \max\{b_i + \max\{t-a_i,0\} - a_j,0\} \\
&= b_j + b_i - a_j + \max\{\max\{t-a_i,0\}, a_j - b_i\} \\
&= b_j + b_i - a_j + \max\{t-a_i, a_j - b_i, 0\} \\
&= b_j + b_i - a_j - a_i + \max\{t, a_i + a_j - b_i, a_i\}
\end{aligned}
$$

t_{ij}演算

$$t_{ij} = b_j + b_i - a_j - a_i + \max\{t, a_i + a_j - b_i, a_i\}$$

$$= \begin{cases} t + b_i + b_j - a_i - a_j & 若 \ max(t, a_i, a_i + a_j - b_i) = t \\ b_i + b_j - a_i & 若 \ max(t, a_i, a_i + a_j - b_i) = a_i \\ b_j & 若 \ max(t, a_i, a_i + a_j - b_i) = a_i + a_j - b_i \end{cases}$$

如果作业J_i和J_j满足$\min\{a_j, b_i\} \geq \min\{a_i, b_j\}$，则称作业$J_i$和$J_j$满足Johnson不等式。

当作业i和j满足Johnson不等式时

- 如交换作业J_i和作业J_j的加工顺序，得到作业集S的另一调度，它所需的加工时间为

$$T'(S,t)=a_i+a_j+T(S-\{J_i,J_j\},t_{ji})$$

其中，

$$t_{ji}=b_j+b_i-a_j-a_i+\max\{t,a_i+a_j-b_j,a_j\}$$

当作业J_i和J_j满足Johnson不等式时，有

$$\max\{-b_i,-a_j\}\le\max\{-b_j,-a_i\}$$

$$a_i+a_j+\max\{-b_i,-a_j\}\le a_i+a_j+\max\{-b_j,-a_i\}$$

$$\max\{a_i+a_j-b_i,a_i\}\le\max\{a_i+a_j-b_j,a_j\}$$

$$\max\{t,a_i+a_j-b_i,a_i\}\le\max\{t,a_i+a_j-b_j,a_j\}$$

当作业i和j满足Johnson不等式时

$$T\,(S,t) \leq T'(S,t)$$

此时，作业J_i排在作业J_j的前面

即：当作业J_i和作业J_j不满足Johnson不等式时，只要交换它们的加工顺序后，不增加加工时间。

流水作业调度的Johnson法则

结论：对于流水作业调度问题，必存在最优调度π，使得作业$J_{\pi(i)}$和$J_{\pi(i+1)}$满足Johnson不等式。

Johnson法则：当调度π，对任何i，作业$J_{\pi(i)}$和$J_{\pi(i+1)}$满足Johnson不等式

$$\min\{b_{\pi(i)}, a_{\pi(i+1)}\} \geq \min\{b_{\pi(i+1)}, a_{\pi(i)}\}$$

称调度π满足Johnson法则

可证明：调度π满足Johnson法则，当且仅当，对任意$i<j$有

$$\min\{b_{\pi(i)}, a_{\pi(j)}\} \geq \min\{b_{\pi(j)}, a_{\pi(i)}\}$$

所有满足Johnson法则的调度均为最优调度。

算法描述

流水作业调度问题的Johnson算法

(1)令 $N_1 = \{i \mid a_i < b_i\}, N_2 = \{i \mid a_i \geq b_i\};$

(2)将N_1中作业依a_i的升序排序；将N_2中作业依b_i的降序排序；

(3)N_1中作业接N_2中作业构成满足Johnson法则的最优调度。

算法举例

	J_1	J_2	J_3	J_4	J_5	J_6
印刷	3	12	5	2	9	12
装订	8	10	9	6	3	1

- N_1={1，3，4}， N_2={2，5，6}
- **N_1按a_i升序：J_4， J_1， J_3，**
- **N_2按b_i降序：J_2， J_5， J_6**
- **合并： J_4， J_1， J_3，J_2， J_5， J_6**

算法复杂度分析

- 算法的主要计算时间花在对作业集的排序。因此，在最坏情况下算法所需的计算时间为O(nlogn)。所需的空间为O(n)。

例7 – 0-1背包问题

- 假设给定n个物体和一个背包，物体i的重量为w_i，价值为v_i（i=1,2,……,n），背包能容纳的物体重量为c，要从这n个物体中选出若干件放入背包，使得放入物体的总重量小于等于c，而总价值达到最大
- 如果用x_i=1表示将第i件物体放入背包，用x_i=0表示未放入，则问题变为选择一组x_i（i=0,1）使得

$$w_x = \sum_{i=1}^{n} w_i x_i \le c, \quad v_x = \sum_{i=1}^{n} v_i x_i$$

，并且达到最大

0−1背包问题的数学表示

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

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

- **0-1背包问题是一个特殊的整数规划问题**

0-1背包问题的最优子结构性质

- 设$(y_1, y_2, \ldots, y_n)$是所给0-1背包问题的一个最优解，满足：

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

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

- 则$(y_2, \ldots, y_n)$是下面相应子问题的一个最优解：

$$\max \sum_{i=2}^{n} v_i x_i$$

为什么？

$$\begin{cases} \sum_{i=2}^{n} w_i x_i \le c - w_1 y_1 \\ x_i \in \{0, 1\}, \quad 2 \le i \le n \end{cases}$$

递归关系

- **设所给0-1背包问题的子问题**

$$\max \sum_{k=i}^{n} v_k x_k \qquad \begin{cases} \sum_{k=i}^{n} w_k x_k \le j \\ x_k \in \{0,1\}, i \le k \le n \end{cases}$$

的最优值为m(i，j)，即m(i，j)是背包容量为j，可选择物品为i，i+1，…，n时0-1背包问题的最优值。

由0-1背包问题的最优子结构性质，有计算m(i，j)的递归式：

$$m(i,j) = \begin{cases} \max\{m(i+1,j), m(i+1,j-w_i)+v_i\} & j \ge w_i \\ m(i+1,j) & 0 \le j < w_i \end{cases}$$

$$m(n,j) = \begin{cases} v_n & j \ge w_n \\ 0 & 0 \le j < w_n \end{cases}$$

-0-1背包问题

例：给定 n= 4 (物品种类)　c= 8 (最大容量)

物品 (weight, value) = { (1, 2), (4, 1), (2, 4), (3, 3) }

w[]={1, 4, 2, 3}, v[]={2, 1, 4, 3}

j / i	j=1	j=2	j=3	j=4	j=5	j=6	j=7	j=8
i=4	0	0	3	3	3	3	3	3
i=3	0	4	4	4	7	7	7	7
i=2	0	4	4	4	7	7	7	7
i=1								9

绿色框为方法 Traceback 的回溯过程 x[]={1,0,1,1}

0-1背包问题的动态规划法

−0−1背包问题

```
void Knapsack(int v[], int w[], int c, int n, int m[][10])
{
    int jMax = min(w[n]-1,c); //背包剩余容量上限范围[0~w[n]-1]
    for(int j=0; j<=jMax;j++)
        m[n][j]=0;

    for(int j=w[n]; j<=c; j++) //限制范围[w[n]~c]
        m[n][j] = v[n];

    for(int i=n-1; i>1; i--)
    {
      jMax = min(w[i]-1,c);
      for(int j=0; j<=jMax; j++)//背包不同剩余容量j<=jMax<c
            m[i][j] = m[i+1][j]; //没产生任何效益
```

0-1背包问题的动态规划法

-0-1背包问题

```
        for(int j=w[i]; j<=c; j++) //背包不同剩余容量j-wi >c
    {
        m[i][j] = max(m[i+1][j],m[i+1][j-w[i]]+v[i]);//效益值增长vi
    }
  }
  m[1][c] = m[2][c];
  if(c>=w[1])
  {
    m[1][c] = max(m[1][c],m[2][c-w[1]]+v[1]);
  }
}
```

算法复杂度分析

- 从m(i，j)的递归式容易看出，程序有两次循环，一次关于i (<=n)，一次关于j (<=c)。算法需要O(nc)计算时间。
- 当背包容量c很大时，算法需要的计算时间较多。例如，当$c>2^n$时，算法需要$\Omega(n2^n)$计算时间。