利用动态规划解决01背包问题

孙宇兴　谭芬

（中国人民解放军通信指挥学院　湖北 武汉　430010）

［摘要］在计算机越来越普及的今天，人们对计算机的求知欲日渐增长，对计算速度的要求也越来越高，因此算法也越来越受人重视，好的算法可以加快计算速度和减少系统资源。

动态规范是算法里一种非常重要的方法，是求解决策过程最优化的数学方法。动态规划自问世以来，在经济管理、生产调度、工程技术和最优控制等方面得到了广泛的应用。例如最短路线、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他方法求解更为方便。

本文通过对经典的01背包问题的求解，从动态规划的角度进行阐述，通过案例对该算法的计算过程进行了直观的描述，并对该算法进行了一定的优化，最后指出该算法的优缺点。

［关键词］背包 动态规划 时间复杂度 空间复杂度

［中图分类号］TP31　［文献标识码］A　［文章编号］1009-5349（2010）05－0121－03

前言

背包问题是一个经典的动态规划模型，很多关于算法的教材都把它作为一道例题，该问题既简单又容易理解，而且在某种程度上还能够揭示动态规划的本质。

将具有不同重量和价值的物体装入一个有固定载重量的背包，以获取最大价值，这类问题被称为背包问题。

背包问题可以扩展出很多种问题，而01背包问题是最常见、最有代表性的背包问题。

一、问题描述

给定一个载重量为M的背包及n个物体，物体i的重量为w_i、价值为p_i，$1\leqslant i\leqslant n$，要求把这些物体装入背包，使背包内的物体价值总量最大。此处我们讨论的物体是不可分割的，通常称这种物体不可分割的背包问题为01背包问题。

二、基本思路

01背包问题的特点是：每种物体只有一件，可以选择放或者不放。假设：x_i表示物体i被装入背包的情况，x_i=0，1。当x_i=0时，表示物体没有被装入背包；当x_i=1时，表示物体被装入背包。根据问题的要求，有如下的约束方程(1)和目标函数(2)：

$$\sum_{i=1}^{n} w_i x_i \leq M \qquad (1)$$

$$optp = \max \sum_{i=1}^{n} p_i x_i \qquad (2)$$

于是，该问题归结为寻找一个满足上述约束方程，并使目标函数达到最大的解向量：$X=(x_1,x_2,\cdots,x_n)$(3)的问题。

三、利用动态规划法求解01背包问题

（一）动态规划算法的基本思想

动态规划算法通常用于求解具有某种最优性质的问题。在这类问题中，可能会有许多可行解。每一个解都对应于一个值，我们希望找到具有最优值的解。动态规划算法与分治法类似，其基本思想也是将待求解问题分解成若干个子问题，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，适合于用动态规划求解的问题，经分解得到子问题往往不是互相独立的。若用分治法来解这类问题，则分解得到的子问题数目太多，有些子问题被重复计算很多次。如果我们能够保存已解决的子问题的答案，而在需要时再找出已求得的答案，这样就可以避免大量的重复计算，节省时间。我们可以用一个表来记录所有已解的子问题的答案。不管该子问题以后是否被用到，只要它被计算过，就将其结果填入表中，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格式。

（二）算法设计

假定背包的载重量范围为0～m。类似于资源分配那样，令$optp_i(j)$表示在前i个物体中，能够装入载重量为j的背包所得的最大价值，j=1，2，……，m。显然，此时在前i个物体中，有些物体可以装入背包，有些物体不能装入背包。于是，可以得到下面的动态规划函数：

$$optp_i(0)=optp_0(j)=0 \qquad (4)$$

$$optp_i(j)=\begin{cases} optp_{i-1}(j) & j<w_i \quad (5) \\ \max\{optp_{i-1}(j),optp_{i-1}(j-w_i)+p_i\} & j\geq w_i \quad (6) \end{cases}$$

(4)式表明：把前面i个物体装入载重量为0的背包，或者把0个物体装入载重量为j的背包，得到的价值都为0。(5)式表明：当第i个物体的重量大于背包的载重量时，装入前i个物体得到的最大价值，与装入前i-1个物体得到的最大价值一样，即第i个物体没有装入背包。(6)式表明：当第i个物体的重量小于背包的载重量时，如果第i个物体装入背包，背包中物体的价值，等于把前i-1个物体装入载重量为$j-w_i$的背包所得到的价值与第i个物体的价值p_i之和；如果第i个物体没有装入背包，则背包中物体的价值，等于把前i-1个物体装入载重量为j的背包，却不装入第i个物体所取得的价值。显然，这两种装入方法，在背包中所取得的价值不一定相同，因此，取这两者中的最大值，作为把前面i个物体装入载重量为j的背包所取得的最优价值。

该算法可分为n阶段：

第一阶段，只装入一个物体，计算在不同载重量的背包情况下，所取得的最大价值；

第二阶段，装入前两个物体，按(5)式和(6)式计算在不同载重量的背包情况下，取得的最大价值；

……

第n阶段，装入前n个物体，按(5)式和(6)式计算在不同载重量的背包情况下，取得的最大价值，而在背包载重量为M时，所得结果就是我们想要的结果。

为了确定具体哪个物体装入背包，从$optp_n(m)$的值向前倒推。有如下递推关系式：

收稿日期：2010年5月15日　责任编辑：张丽

孙宇兴（1985－），男，汉族，黑龙江克山农场人，中国人民解放军通信指挥学院信息化作战训练中心，助教。

谭芬（1980－），女，汉族，湖北天门人，中国人民解放军通信指挥学院信息化作战训练中心，讲师。

若$optp_i(j) > optp_{i-1}(j)$则$x_i = 1, j = j - w_i$ （7）

若$optp_i(j) = optp_{i-1}(j)$则$x_i = 0$ （8）

(7)式表明：如果$optp_i(j)$大于$optp_{i-1}(j)$，则物体i被装入背包；如果$optp_i(j)$等于$optp_{i-1}(j)$，表明i物体未被装入背包。按照该关系式，i从n到1依次类推，直到确定第一个物体是否被装入背包为止，就能确定装入背包的具体物体。

（三）存储结构

该算法需要将每一个物体的重量和价值分别存储起来，并针对不同载重量的背包对物体分别计算，故考虑使用数组来实现。对于物体i，用weight[i]来表示重量、用p[i]表示价值，解向量用x[i]表示物体i是否被放入，上面提到的$optp_i(j)$则用二维数组相应的optp[i][j]来表示，物体个数n，背包载重量为m。

（四）算法实现

```
initial knapsack_dynamic(initial optp[][],weight[],p[],x[],n,m)
{
initial i,j,value;
//根据(4)式初始化第0列及解向量X
//解向量初始值设置为false，表示起初所有物体均未放入背包
for(i = 0; i <= n; i++)
{
optp[i][0] = 0;
x[i] = FALSE;
}
//根据(4)式初始化第0行
for(i = 0; i <= m; i++)
{
optp[0][i] = 0;
}
//利用(5)(6)式计算optp[i][j]
for(i = 1; i <= n; i++)
{
for(j = 1; j <= m; j++)
{
optp[i][j] = optp[i-1][j];
if((j >= weight[i]) && (optp[i-1][j - weight[i]]+p[i]) > optp[i-1][j])
optp[i][j] = optp[i-1][j-weight[i]] + p[i];
}
}
//确定装入背包的具体物体
j = m;
for(i = n; i > 0; i--)
{
if(optp[i][j] > optp[i-1][j])
{
x[i] = TRUE;
j = j - weight[i];
}
}
//value就是所要求的值
value = optp[n][m];
return value;
}
```

1.案例分析

现有载重量为10的背包，有四个物体A、B、C、D，其重量分别为4、2、5、3，价值分别为4、3、5、8，要求放入物体使背包所获价值最大。

用optp[i][j]来存储这四个物体在不同载重量的背包下所获的价值，计算过程如下表所示：

	j=10	j=9	j=8	j=7	j=6	j=5	j=4	j=3	j=2	j=1	j=0
optp[0][j]	0	0	0	0	0	0	0	0	0	0	0
optp[1][j]	4	4	4	4	4	4	4	0	0	0	0
optp[2][j]	7	7	7	7	7	4	4	3	3	0	0
optp[3][j]	9	9	8	8	7	5	4	3	3	0	0
optp[4][j]	16	15	13	12	11	11	8	8	3	0	0

相应的，对于optp[i][j]，物体的放入情况及重量如下表所示：

	j=10		j=9		j=8		j=7		j=6		j=5		j=4		j=3		j=2		j=1		j=0	
	放入的物体	重量	放入的物体	重量	放入的物体	重量	放入的物体	重量	放入的物体	重量	放入的物体	重量	放入的物体	重量	放入的物体	重量	放入的物体	重量	放入的物体	重量	放入的物体	重量
0个物体	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
前1个物体(A)	A	4	A	1	A	4	A	4	A	4	A	4	A	4	0	0	0	0	0	0	0	0
前2个物体(AB)	AB	6	AB	6	AB	6	AB	6	AB	6	A	4	A	4	B	2	B	2	0	0	0	0
前3个物体(ABC)	AC	9	AC	9	BC	7	BC	7	AB	6	C	5	A	4	B	2	B	2	0	0	0	0
前4个物体(ABCD)	BCD	10	ABD	9	CD	8	AD	7	BD	5	BD	5	D	3	D	3	B	2	0	0	0	0

2.时间空间复杂度

该算法中，矩阵optp的大小为(m+1)×(n+1)，物体的重量、价值和解向量大小都等于物体个数n，故该算法的空间复杂度为O(nm)。对物体重量、价值的初始化(算法实现略)所需时间都为n，解向量和矩阵第0行初始化时间为n，矩阵第0列初始化时间为m，对矩阵optp的计算所需时间为n×m，解向量X的确定时间为n，故整个算法的时间复杂度为O(nm)。

（五）算法优化

在上述算法中，时间复杂度不能再继续优化了，但是空间复杂度是可以继续优化的。

上述算法中，存储optp使用的是二维数组，其大小为(m+1)×(n+1)，但是仔细观察发现，optp[i][j]只与optp[i-1][j]和optp[i-1][j - weight[i]]有关，与optp[k][l](k=1，2，……，i-2，i+1，……n，j=1，2，……，m)无关，故可考虑只用一维数组_optp来存储，_optp[j]相当于optp[i][j]。而考虑到optp[i][j]是由optp[i][j]和optp[i-1][j - weight[i]]共同计算得到的，故该算法中的j循环要从后往前计算，_optp[]的计算算法设计如下所示：

```
for(i = 1; i <= n; i++)
{
for(j = m; j >= 1; j--)
{
if((j >= weight[i]) && (_optp[j - weight[i]]+p[i]) > _optp[j])
_optp[j] = optp[j-weight[i]] + p[i];
```

（上接120页）

对于消防工作有责无权，也在一定程度上挫伤了公安派出所对消防工作的主动性和积极性。

二、加强公安派出所消防监督工作的建议

（一）组建派出所消防监督工作领导机构，完善派出所消防监督网络体系

要建立派出所消防监督工作的领导机构。各地消防大队对于派出所消防监督工作，能够起到的仅仅是业务指导作用、而非领导作用，市、县公安局也仅仅能对派出所的消防监督从大局上进行部署和协调。因此，在日常工作中，必须在市、县公安局中建立一个领导派出所开展消防监督工作的专门机构，如派出所消防工作指导处或办公室，以减轻基层消防大队的工作压力。这一点，长春市公安局已经走在了全省的前列。该机构具体负责辖区内消防工作开展、履行政府消防工作职能，组织开展消防监督检查和专项整治，督促整改火灾隐患，负责对全地区公安派出所的消防监督工作进行领导、组织、协调和管理，年终参与对各派出所消防监督工作的绩效考核，保证派出所日常消防监督工作的正常、有序开展。

（二）增加专职消防民警数量，逐步建立相对完善的派出所消防民警管理制度

增加派出所消防监督人员编制不是一朝一夕可以解决的问题。在《消防法》明确派出所消防监督检查法定主体的基础上，要根据派出所管辖区域的特点，在一些辖区场所较多、单位较多的派出所设专职消防民警，一些以社区住宅为主的派出所设兼职消防民警，还可由政府出资招收地方编制的消防监督辅助人员，通过设专职消防民警、辅助消防民警的方式，首先保证建立一支相对稳定的派出所消防民警队伍。同时逐步建立完善相应的管理制度，明确人员职责。重要的是建立激励制度，充分调动派出所民警开展消防工作的积极性，逐步改变派出所在消防工作中被动应付的工作局面，推动派出所消防监督工作的进行和开展。

（三）加强对派出所民警的业务培训，提高派出所民警消防业务水平

在建立一支相对稳定的派出所民警队伍的基础上，各级公安消防机构要加大对派出所民警的业务培训力度，定期开展理论学习、现场施教、检查考核等培训、指导工作，提高派出所民警消防业务能力，适应日常消防监督管理工作的需要。对于专（兼）职消防民警，除了要求他们熟练掌握和运用消防法律、法规外还应参加考核颁发证书。公安消防机构还可将常用的消防法律法规、消防技术规范进行摘录，印制成小册子，发到派出所民警手上，以利于他们随用随查，方便工作。指导派出所及专职消防民警严格按照工作程序来开展工作，从而使派出所消防监督管理逐步法制化、规范化。

（四）明确派出所考核机制，尽快修订派出所消防监督管理相关配套规定

新《消防法》及107号令在派出所消防监督管理工作规定上留给各省级公安机关的一定权限，需要根据实际情况进一步明确派出所监督管理范围，细化完善本地派出所消防监督工作规定内容。从组织机构、管辖范围、工作职责、监督检查、火警火灾处置、业务培训、档案和台账、行政处罚、监督考评和奖惩等方面制定规定和具体工作措施，按照抓硬件与抓软件结合、抓治安防范与抓消防监督相结合、职能与任务相结合、考评与奖惩相结合的基本原则认真贯彻落实，增加消防监督工作考核所占比重，从而使各级公安机关特别是基层公安派出所能够思路清晰、目标明确、标准科学地组织和开展消防监督工作，充分发挥公安派出所“点多面广”的消防监督优势，防止消防安全监督工作失控漏管，全面提高社会消防安全控制能力。

（上接122页）

```
}
}
```

上述例子中，相应的_optp[j]计算结果如下表所示：

	j=10	j=9	j=8	j=7	j=6	j=5	j=4	j=3	j=2	j=1	j=0
i=0，_optp[j]	0	0	0	0	0	0	0	0	0	0	0
i=1，_optp[j]	4	4	4	4	4	4	4	0	0	0	0
i=2，_optp[j]	7	7	7	7	7	4	4	3	3	0	0
i=3，_optp[j]	9	9	8	8	7	5	4	3	3	0	0
i=4，_optp[j]	16	15	13	12	11	11	8	8	3	0	0

显然，对于物体i，_optp[j]的计算不会影响到_optp[0，1，……，weight[i]-1]，故上述算法可继续优化为：

```
for(i = 1; i <= n; i++)
{
for(j = m; j >= weitht[i]; j--)
{
if((_optp[j - weight[i]]+p[i]) > _optp[j])
_optp[j] = optp[j-weight[i]] + p[i];
}
}
```

对于01背包问题，常见的要求有两种：一种是恰好装满背包、另一种则没有要求，针对这两种问法，可从初始化上来区分。

当要求恰好装满背包时，初始化时将_optp[0]设为0，其他_optp[1，2，……，m]全部设为－∞，这样就可保证最终得到的_optp[n]是一种恰好装满的最优解，此时可以把初始化理解成没有任何物体可放时，只有容量为0的背包能被价值为0的nothing“恰好装满”；当没有这个限制时，初始化时应将_optp[0，1，……，m]都设为0，此时可以理解为任何一个背包都有一个合法的解“什么都不装”，这个解的价值为0。

当优化后，空间复杂度变为O(m)，计算所需时间为：

$$m - \sum_{i=1}^{n} weight[i]$$

当weitht[i]较大时，可以节省很多时间。

动态规划的缺点：对于01背包问题，用动态规划方法解很容易理解，但是这种方法有一些弊端，从上述算法可以看出：当物体的重量较大时，所需的物理空间较大；当物体的重量不是整数时，无法利用数组来存储该结构。

四、结束语

在日常生活中，01背包问题有很多应用，比如如何利用有限空间的手提包，装入外出旅行的各种必需品。

01背包问题还有很多种解法，每种解法也有各种不同的实现，比如回溯法可以利用结构或类来表示状态空间树，每一本关于算法的书籍都把它当成必讲的题目，而由01背包问题引申出来的各种题目也层出不穷，例如完全背包问题就要求每种物体都无限制使用。01背包问题是最基本的背包问题，它包含了背包问题中设计状态、方程的最基本思想，但是无论哪种类型的背包问题，都可以转换为01背包问题。所以读者们要仔细体会01背包问题基本思路的得出方法，状态转移方程的意义，以及关于时间空间复杂度的优化等问题。

动态规划法是解题的一种思路，它的存在并不仅仅用于解决01背包问题，而是对分而治之和递推的一种有效结合，这才是本文的真正意义所在。