算法设计与分析实验报告

实验名称：0/1 背包问题(动态规划)

一、问题陈述，相关背景、应用及研究现状的综述分析

1.问题陈述：

给定 **n** 种物品和一背包。物品 **i** 的重量是 **wi**，其价值为 **vi**，背包的容量为 **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**?**在选择装入背包的物品时，对每种物品 **i** 只有两种选择，即装入背包或不装入背包。不能将物品 **i** 装入背包多次，也不能只装入部分的物品 **i;**

二、模型拟制、算法设计和正确性证明

1. 算法设计：

考虑原问题的一部分，设 $V(i,j)$ 表示将前 $i(1\leqslant i\leqslant n)$ 个物品装入容量为 $j(1\leqslant j\leqslant C)$ 的背包获得的最大价值，在决策 x_i 时，已确定了 $(x_1,\cdots,x_{i-1})$，则问题处于下列两种状态之一：

(1) 背包容量不足以装入物品 i，则装入前 i 个物品得到的最大价值和装入前 $i-1$ 个物品得到的最大价值是相同的，即 $x_i=0$，背包不增加价值。

(2) 背包容量可以装入物品 i，如果把第 i 个物品装入背包，则背包中物品的价值等于把前 $i-1$ 个物品装入容量为 $j-w_i$ 的背包中的价值加上第 i 个物品的价值 v_i；如果第 i 个物品没有装入背包，则背包中物品的价值等于把前 $i-1$ 个物品装入容量为 j 的背包中所取得的价值。显然，取二者中价值较大者作为把前 i 个物品装入容量为 j 的背包中的最优解。则得到如下递推式：

$$V(i,j)=\begin{cases}V(i-1,j) & j<w_i\\ \max\{V(i-1,j),V(i-1,j-w_i)+v_i\} & j\geqslant w_i\end{cases}$$

为了确定装入背包的具体物品，从 $V(n,C)$ 的值向前推，如果 $V(n,C)>V(n-1,C)$，表明第 n 个物品被装入背包，前 $n-1$ 个物品被装入容量为 $C-w_n$ 的背包中；否则，第 n 个物品没有被装入背包，前 $n-1$ 个物品被装入容量为 C 的背包中。依此类推，直到确定第 1 个物品是否被装入背包中为止。由此，得到如下函数：

$$x_i=\begin{cases}0 & V(i,j)=V(i-1,j)\\ 1 & j=j-w_i \quad V(i,j)>V(i-1,j)\end{cases}$$

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，动态规划法求解 0/1 背包问题的过程如图 6.17 所示，具体过程如下。

	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	x_1=1
w_2=2 v_2=3 2	0	0	6	6	9	9	9	9	9	9	9	x_2=1
w_3=6 v_3=5 3	0	0	6	6	9	9	9	9	11	11	14	x_3=0
w_4=5 v_4=4 4	0	0	6	6	9	9	9	10	11	13	14	x_4=0
w_5=4 v_5=6 5	0	0	6	6	9	9	9	12	12	15	**15**	x_5=1

图 6.17　0/1 背包的求解过程

三、时间和空间复杂性分析

时间复杂度：

Knapsack 需要 O(nc)计算时间

Traceback 需要 0(n)计算时间

空间复杂度：

没有使用额外空间

S（n）=O(1)

四、程序实现和实验测试过程

源程序见 knapsack_int.cpp

测试过程：

五、总结

算法缺点：

要求所给物品重量为整数；

当背包容量 c 很大时，算法计算时间较长。