

## 算法设计与分析实验报告

实验名称：0/1 背包问题(回溯算法)

### 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  $w_i$ ，其价值为  $v_i$ ，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

0-1 背包问题的解空间可用子集树表示。在搜索解空间树时，只要其左儿子节点是一个可行结点，搜索就进入其左子树。当右子树有可能包含最优解时才进入右子树搜索；否则将右子树减去。

计算右子树上界函数的方法是：将剩余物品质量依其单位重量价值排序，依次装入物品，直到装不下为止，此时再装入物品的一部分而装满背包，由此得到的价值是右子树中解的上界。

利用一个类来记录物品的重量、价值等信息，同时用一个数组来记录最后解的信息。实现过程中可以先用 sort 算法将物品按单位重量价值由大到小排好序，方便后续操作。

### 三、时间和空间复杂性分析

计算上界需要  $O(n)$  时间，在最坏情况下有  $O(2^n)$  个右儿子节点需要计算上界，故解 0-1 背包问题的回溯算法所需的计算时间为  $O(n2^n)$

时间复杂度：  $O(n2^n)$

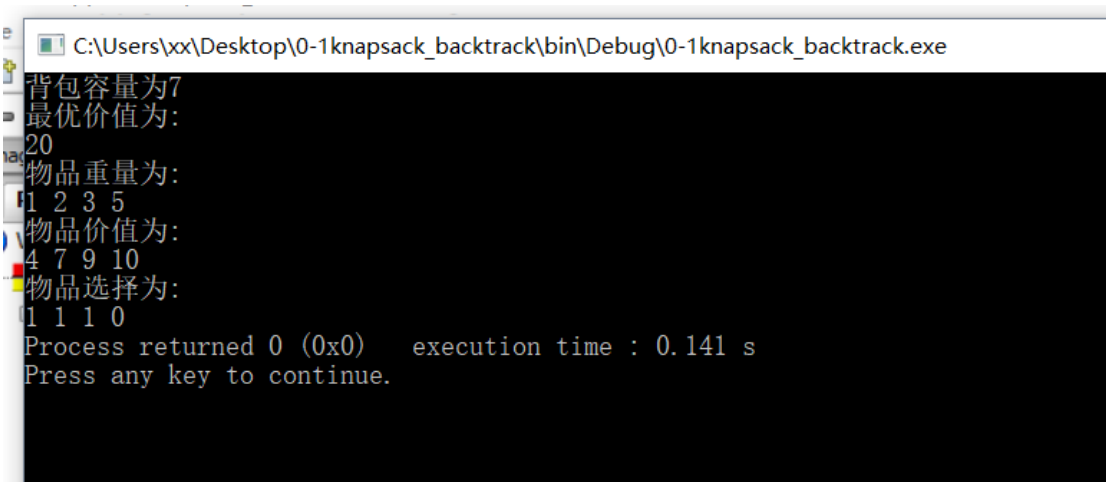
空间复杂度：

物品的重量信息和价值信息都是提前申请数组存储的，在递归时只需要申请一个 bound 来返回上界，这也只需要常数级的空间。因此空间复杂度为  $O(1)$

## 四、程序实现和实验测试过程

源程序见 0-1knapsack\_backtrack.cpp

测试过程：



```
C:\Users\xx\Desktop\0-1knapsack_backtrack\bin\Debug\0-1knapsack_backtrack.exe
背包容量为7
最优价值为:
20
物品重量为:
1 2 3 5
物品价值为:
4 7 9 10
物品选择为:
1 1 1 0
Process returned 0 (0x0) execution time : 0.141 s
Press any key to continue.
```

## 五、总结

0-1 背包问题是子集选取问题。一般情况下是 NP 难的，不论是动态规划还是回溯法，都较难找到多项式级别时间复杂度的算法。