浙 江 工 商 大 学
计算机与信息工程学院

《算法分析与设计课程作业》

学　号：18020100006
姓　名：徐晨鸥
班　级：计算机与信息技术18硕
日　期：2019.04.22

一、作业内容

背包问题（Knapsack problem）是一种组合优化的NP完全问题。问题可以描述为：给定一组物品，每种物品都有自己的重量和价格，在限定的总重量内，我们如何选择，才能使得物品的总价格最高。问题的名称来源于如何选择最合适的物品放置于给定背包中。

我们有n种物品，物品j的重量为w_j，价格为p_j。

我们假定所有物品的重量和价格都是非负的。背包所能承受的最大重量为W。

如果限定每种物品只能选择0个或1个，则问题称为**0-1背包问题**。

可以用公式表示为：

最大化 $$\sum_{j=1}^{n} p_j x_j$$

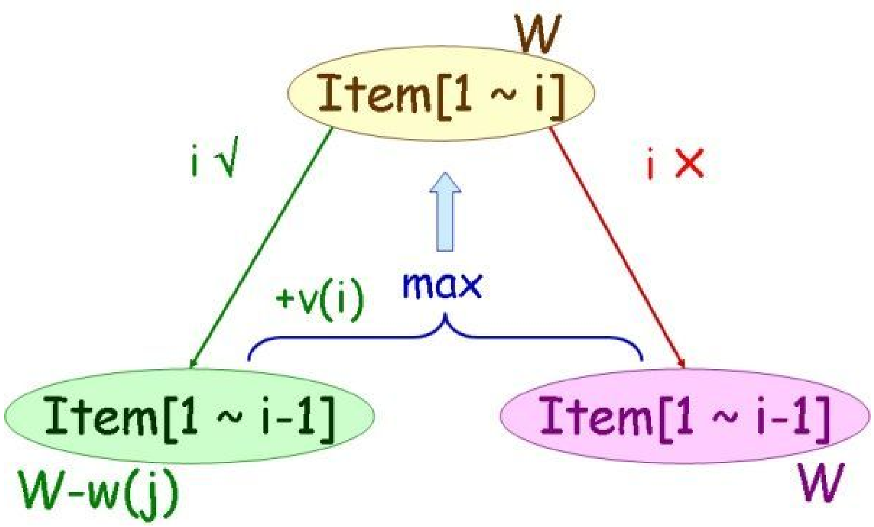
受限于 $$\sum_{j=1}^{n} w_j x_j \leqslant W, \quad x_j \in \{0,1\}$$

二、算法步骤与实现

定义子问题

p(i,W):在前 i 个物品中挑选总重量不超过W的物品，每种物品最多只能1个

最优值为 m（i，W） = max{m（i-1，W），m（i-1，W-wi）+vi}



从而得到m 的表示为：

$$m(i,W)=\begin{cases}0 & \text{if } i=0\\ 0 & \text{if } W=0\\ m(i-1,W) & \text{if } w_i>W\\ \max\{m(i-1,W),v_i+m(i-1,W-w_i)\} & \text{otherwise}\end{cases}$$

算法的伪代码：

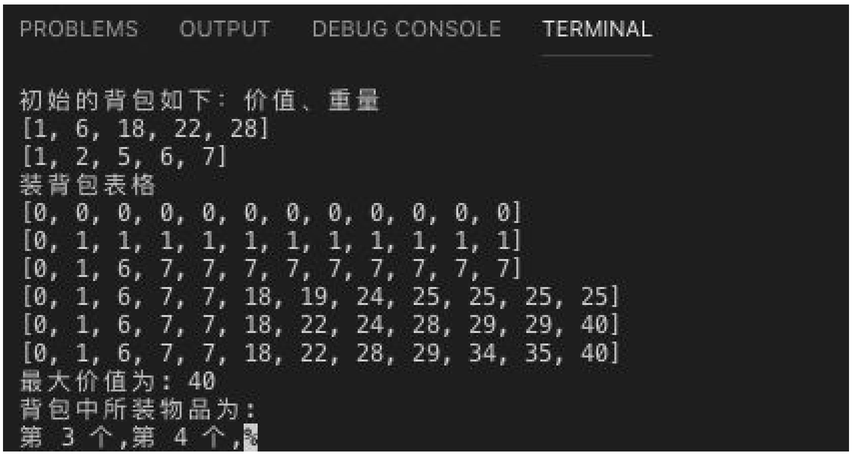
```
Input: n, w_1,…,w_n, v_1,…,v_n , C
for W = 0 to C
    m[0, W] = 0
for i = 1 to n
    m[i, 0] = 0
for i = 1 to n
    for W = 1 to C
        if (w_i > w)
            m[i, W] = m[i-1, W]
        else
            m[i, W] = max { m[i-1, W], v_i + m[i-1, W-w_i] }
return m[n, C]
```

python 实现的代码：

```
import numpy as np
#多个（n）物品，每种物品都有自己的重量（w_i）和价值（v_i），在限定的总重量/总容量（C）内，
#选择其中若干个（也即每种物品可以选0个或1个），设计选择方案使得物品的总价值最高。

#setting parameters
value_list = [1,6,18,22,28]
weight_list = [1,2,5,6,7]
n = 5
w = 11

#p(i,W):在前 i 个物品中挑选总重量不超过W的物品，每种物品最多只能1个
#最优值为 m（i，W） = max{m（i-1，W），m（i-1，W-wi）+vi}
```

```python
def bag(n, c, w, v):
    """
    n 物品的数量,
    c 书包能承受的重量,
    w 每个物品的重量,
    v 每个物品的价值
    """
    # 置零, 表示初始状态
    value = [[0 for j in range(c + 1)] for i in range(n + 1)]
    for i in range(1, n + 1):
        for j in range(1, c + 1):
            value[i][j] = value[i - 1][j]
            # 背包总容量够放当前物体, 遍历前一个状态考虑是否置换
            if j >= w[i - 1] and value[i][j] < value[i - 1][j - w[i - 1]] + v[i - 1]:
                value[i][j] = value[i - 1][j - w[i - 1]] + v[i - 1]
    for x in value:
        print(x)
    return value

def show(n, c, w, value):
    print('最大价值为:', value[n][c])
    x = [False for i in range(n)]
    j = c
    for i in range(n, 0, -1):
        if value[i][j] > value[i - 1][j]:
            x[i - 1] = True
            j -= w[i - 1]
    print('背包中所装物品为:')
    for i in range(n):
        if x[i]:
            print('第', i+1, '个,', end='')

#print(solve(value_list,weight_list,w,n))
print("初始的背包如下：价值、重量")
print(value_list)
print(weight_list)
print("装背包表格")
show(n,w,weight_list,bag(n,w,weight_list,value_list))
```

三、实验结果与分析



可以通过实验结果看出，使用动态规划的算法的确可以得到最优解。
上图在背包承重为11，物品价值为1，6，18，22，28，且对应重量为1，2，5，6，7的情况下。选择了最高价值的组合3和4。

而通过装背包的表格可以看出动态规划的过程中的结果。