
西安交通大學

XI'AN JIAOTONG UNIVERSITY

# 最小费用购物问题

算法设计与分析大作业

| | |
|---|---|
| 课程名称： | 算法设计与分析 |
| 姓名： | 凌晨 |
| 学院： | 软件工程学院 |
| 专业： | 软件工程 |
| 学号： | **2214414320** |

**2023 年 11 月 27 日**

# 目录

## 一、 题目描述

**问题描述:** 商店中每种商品都有标价。例如，一朵花的价格是 2 元，一个花瓶的价格是 5 元。为了吸引顾客，商店提供了一组优惠商品价。优惠商品是把一种或多种商品分成一组，并降价销售。例如，3 朵花的价格不是 6 元而是 5 元，2 个花瓶加 1 朵花的优惠价是 10 元。试设计一个算法，计算出某顾客所购商品应付的最少费用。

**算法设计:** 对于给定欲购商品的价格和数量，以及优惠商品价，计算所购商品应付的最少费用。

**数据输入:** 由文件 input.txt 提供欲购商品数据。文件的第 1 行中有 1 个整数 B($O \leq B \leq 5$), 表示所购商品种类数。在接下来的 B 行中，每行有 3 个数 C、K 和 P。C 表示商品的编码 (每种商品有唯一编码)，$1 \leq C \leq 999$;K 表示购买该种商品总数，$1 \leq K \leq 5$; 是该种商品的正常单价（每件商品的价格)，$1 \leq P \leq 999$。注意，一次最多可购买 25 件商品。

由文件 offer.txt 提供优惠商品价数据。文件的第 1 行中有 1 个整数 S($0 \leq S \leq 99$)，表示共有 S 种优惠商品组合。接下来的 S 行，每行的第 1 个数描述优惠商品组合中商品的种类数 j。接着是 j 个数字对 (C、K)，其中 C 是商品编码，$1 \leq C \leq 999$;K 表示该种商品在此组合中的数量，$1 \leq K \leq 5$。每行最后一个数字 P($1 \leq P \leq 9999$）表示此商品组合的优惠价。结果输出：将计算出的所购商品应付的最少费用输出到文件 output.txt。

| 输入文件示例 | | 输出文件示例 |
|---|---|---|
| input.txt | offer.txt | output.txt |
| 2 | 2 | 14 |
| 7 3 2 | 1 7 3 5 | |
| 8 2 5 | 2 7 1 8 2 10 | |

## 二、 算法设计

### 1. 最优子结构设计

不妨假设下面这个式子 $dp[i1][i2][i3][i4][i5]$ 为第一个种类为 i1 个，第二个种类为 i2 个，...，第五个种类为 i5 个时，顾客所购商品应付的最少费用。

下面证明这个式子具有最优子结构，即对于任意 0<=jk<ik，k 从 1 到 5，都有 $dp[j1][j2][j3][j4][j5]$ 为第一个种类为 j 个，第二个种类为 j2 个，...，第五个种类为 j5 个时，顾客所购商品应付的最少费用。证明过程如下：

证明:
假设dp[j1][j2][j3][j4][j5]不是第一个种类为j1个，第二个种类为j2个，...，第五个种类为j5个时，顾客所购商品应付的最少费用。(0<=jk<ik，k从1到5)
那么说明存在一个var变量满足:
var < dp[j][j2][j3][j4][j5] (1) 式
而dp[i1][i2][i3][i4][i5]的计算必然会使用到dp[j1][j2][j3][j4][j5]的值，优惠价格和商品单价进行计算，假设优惠价格和商品单价的总计算值为t。
则dp[i1][i2][i3][i4][i5]满足以下方程:
dp[i1][i2][i3][i4][i5] = dp[j1][j2][j3][j4][j5] + t (2) 式
注意: 这里的dp[j1][j2][j3][j4][j5]指定为用到的子结构的值。
结合 (1) (2) 式有:

```
   dp[i1][i2][i3][i4][i5] = dp[j1][j2][j3][j4][j5] + t > var + t
   与原定义矛盾，因此假设错误。
   因此，dp[j1][j2][j3][j4][j5]是第一个种类为j1个，第二个种类为j2个，...
，第五个种类为j5个时，顾客所购商品应付的最少费用。（0<=jk<ik，k从1到5）
```

### 2. 递归计算最优值

若不采用任何优惠，dp 的计算满足下面的定义式：

$$dp[i1][i2][i3][i4][i5] = i1 * price[1] + i2 * price[2] + i3 * price[3] + i4 * price[4] + i5 * price[5]$$

若采用优惠策略进行购买物品，dp 的计算同时满足下面的递归式：

$$dp[i1][i2][i3][i4][i5] = min\{f[i1][i2][i3][i4][i5], temp\}$$

其中,

$$temp = f[i1 - a[i].seq[1]][i2 - a[i].seq[2]][i3 - a[i].seq[3]][i4 - a[i].seq[4]][i5 - a[i].seq[5]] + a[i].value$$

递归式子的正确性不言自证，这是题目的特性。

## 三、 算法代码

### 1. 代码展示

算法的代码如下，使用 JAVA 进行编写：

```
import java.util.Scanner;

public class Solution {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // 模拟input.txt录入
        System.out.println("input.txt输入： ");
        int B = scanner.nextInt();
        int[] id = new int[5]; //
            用于进行商品的编号转化为id号的数组：id号表示为下标，比如7、8转化为0、1.
        Goods[] goods = new Goods[5];
        for (int i = 0; i < B; i++) {
            int C = scanner.nextInt();
            int K = scanner.nextInt();
            int P = scanner.nextInt();
            goods[i] = new Goods(i,K,P);
            id[i] = C;
        }
        for (int i = B; i < 5; i++) { // 将剩余的商品假设出来
            goods[i] = new Goods(i,0,0); // 将K=0表示不需要
        }
        System.out.println("offer.txt输入： ");
```

```
        int S = scanner.nextInt();
        int j = 0;
        Offer[] offers = new Offer[S]; // 优惠策略清单
        for (int i = 0; i < S; i++) { // 将优惠政策列表记录下来!
            Offer offer = new Offer();
            j = scanner.nextInt();
            for (int k = 0; k < j; k++) {
                int C = scanner.nextInt();
                int K = scanner.nextInt();
                offer.num[findId(id,C)] = K; // 记录id号的优惠政策
            }
            int price = scanner.nextInt();
            offer.setPrice(price);
            offers[i] = offer;
        }
        System.out.println(fuc(goods,offers));
    }
    public static int fuc(Goods[] goods,Offer[] offers){
        int[][][][][] dp = new int[5][5][5][5][5]; // 因为每个产品至多只购买5个。
        // 更新dp矩阵，更新值为不使用优惠策略。
        for (int i1 = 0; i1 <= goods[0].need; i1++) {
            for (int i2 = 0; i2 <= goods[1].need; i2++) {
                for (int i3 = 0; i3 <= goods[2].need; i3++) {
                    for (int i4 = 0; i4 <= goods[3].need; i4++) {
                        for (int i5 = 0; i5 <= goods[4].need; i5++) {
                            dp[i1][i2][i3][i4][i5] = i1*goods[0].price + i2*goods[1].price
                                +i3*goods[2].price
                                   + i4*goods[3].price + i5*goods[4].price;
                        }
                    }
                }
            }
        }
        // 使用优惠政策
        for (int i = 0; i < offers.length; i++) {
            for (int i1 = offers[i].num[0]; i1 <= goods[0].need; i1++) {
                for (int i2 = offers[i].num[1]; i2 <= goods[1].need; i2++) {
                    for (int i3 = offers[i].num[2]; i3 <= goods[2].need; i3++) {
                        for (int i4 = offers[i].num[3]; i4 <= goods[3].need; i4++) {
                            for (int i5 = offers[i].num[4]; i5 <= goods[4].need; i5++) {
                                dp[i1][i2][i3][i4][i5] =Integer.min(dp[i1][i2][i3][i4][i5],
        dp[i1-offers[i].num[0]][i2-offers[i].num[1]]
        [i3-offers[i].num[2]][i4-offers[i].num[3]][i5-offers[i].num[4]]
                                        +offers[i].price);
                            }
                        }
                    }
                }
            }
        }
```

```
        return
            dp[goods[0].need][goods[1].need][goods[2].need][goods[3].need][goods[4].need];
    }
    public static int findId(int[] id, int C){ // 找到商品编号对于的id号
        for (int i = 0; i < id.length; i++) {
            if(id[i]==C){
                return i;
            }
        }
        return -1;
    }
    public static class Goods{ // 商品类
        int id; // 编号
        int price; // 价格
        int need; // 需要数量
        public Goods(int C,int K,int P){
            this.id = C;
            this.price = P;
            this.need = K;
        }
    }
    public static class Offer{ // 用来表示优惠组合的类
        int[] num = new int[5]; // num下标表示第几个商品，num的数字表示所需商品的数量
        int price; // 优惠价格
        public int getPrice() {
            return price;
        }
        public void setPrice(int price) {
            this.price = price;
        }
    }

}
```

## 2. 复杂度分析

时间复杂度：

尽管至多存在 6 重循环，但其中 5 重循环中，每层循环至多执行 5 次，因此执行的次数至多为 5*5*5*5*5=3125 次。所以主要的复杂度看最外层的次数，即 offer.length（优惠策略的数量）；

因此，记优惠策略数量为 n，该算法的时间复杂度为 O(3125n)=O(n)

空间复杂度：

使用了两个类，计算空间开销主要集中在商品类和优惠类，商品类至多 5 个，因此空间开销为常数，而优惠类的个数不定，空间开销主要取决于优惠类个数。

因此，即优惠策略数量为 n，该算法的空间复杂度为 O(n);

## 四、 测试结果

下面为代码测试的结果
结果 1：

```
input.txt输入:
2
7 3 2
8 2 5
offer.txt输入:
2
1 7 3 5
2 7 1 8 2 10
14
```

结果 2：

```
input.txt输入:
3
7 4 3
8 2 1
3 2 5
offer.txt输入:
1
2 8 2 3 2 10
22
```

结果 1 为题目展示，结果 2 为自己测试的数据，明显可见正确。
剩下的测试不再给出，老师感兴趣可以自行调试！

## 五、 解题心得

在解题的时候，我们不能被形式所蒙蔽了双眼，或者纠结于具体的代码实现过程。我们应该先抽丝剥茧，找到问题的核心，即该问题具有最优子结构和递归算法。在弄清楚这两点后，再面对这道题目的时候，就显得更加从容了。可以确定算法的实现为动态规划。

但是，在具体的代码实现，还是需要多多斟酌数据结构，循环的使用，尽可能降低空间复杂度，时间复杂度！