|  |  |
| --- | --- |
| **算法实验报告** | |
|  | |
| 图片包含 标牌  已生成极高可信度的说明 | |
|  | |
|  | |
|  | |
| **姓名** | 代珉玥 |
| **班级** | 软件001班 |
| **学号** | 2205223077 |
| **电话** | 18585038226 |
| **Email** | 2040257842@qq.com |
| **日期** | 2022-12 |

目录

[题目 1](#_Toc1020)

[1. 问题描述 1](#_Toc2174)

[2. 数据输入 1](#_Toc13403)

[分析 2](#_Toc15947)

[主干代码说明 2](#_Toc2772)

[运行结果展示 3](#_Toc28001)

[异常处理 4](#_Toc6065)

[代码附录 5](#_Toc25479)

# 题目

## 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，计算出某一顾客所购商品应付的最少费用。

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

## 数据输入

控制台的第1行中有1个整数B(0≤B≤5)，表示所购商品种类数。接下来的B行，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≤C≤999。K表示购买该种商品总数，1≤K≤5。P是该种商品的正常单价(每件商品的价格)，1≤P≤999。请注意，一次最多可购买5\*5=25件商品。

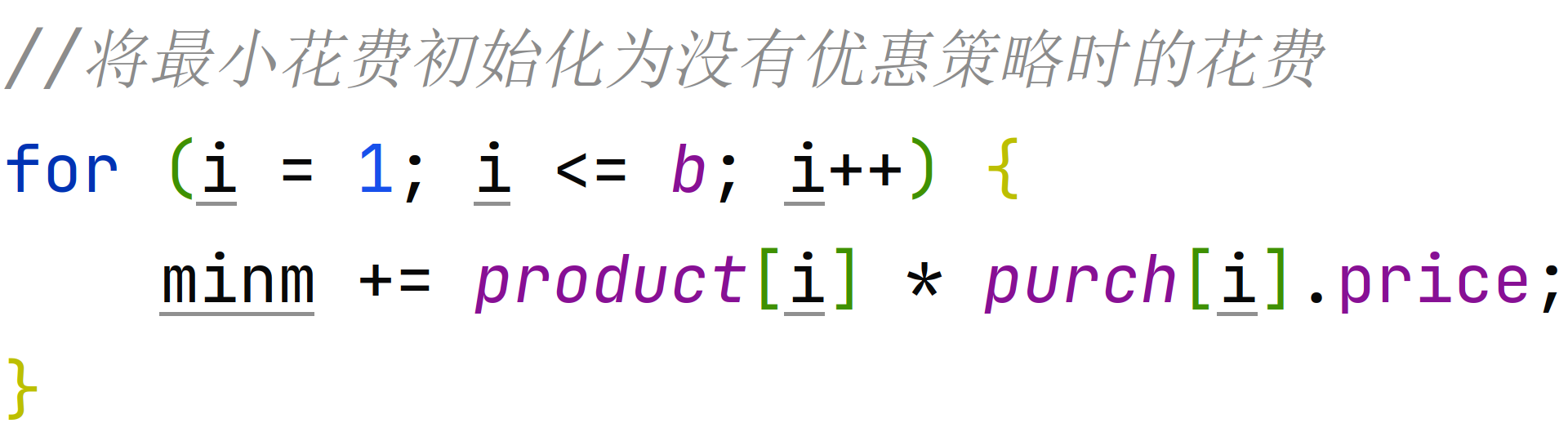
接来下输入1个整数S(0≤S≤99)，表示共有S种优惠商品组合。接下来的S行，每行的第一个数描述优惠商品组合中商品的种类数j。接着是j个数字对(C，K)，其中C是商品编码，1≤C≤999。K表示该种商品在此组合中的数量，1≤K≤5。每行最后一个数字P(1≤P≤9999)表示此商品组合的优惠价。

# 分析

找出问题的规律，设cost（a,b,c,d,e）表示购买商品组合（a,b,c,d,e）需要的最少费用，A[k],B[k],C[k],D[k],E[k]表示第k种优惠方案的商品组合。offer(m)是第m种优惠方案的价格。如果cost(a,b,c,d,e)使用了第m种优惠方案，则找出最优子问题的递归表达式： cost(a,b,c,d,e)=cost(a-A[m],b-B[m],c-C[m],d-D[m],e-E[m])+offer(m)

即该问题具有最有子结构性质，可以用动态规划算法来实现。

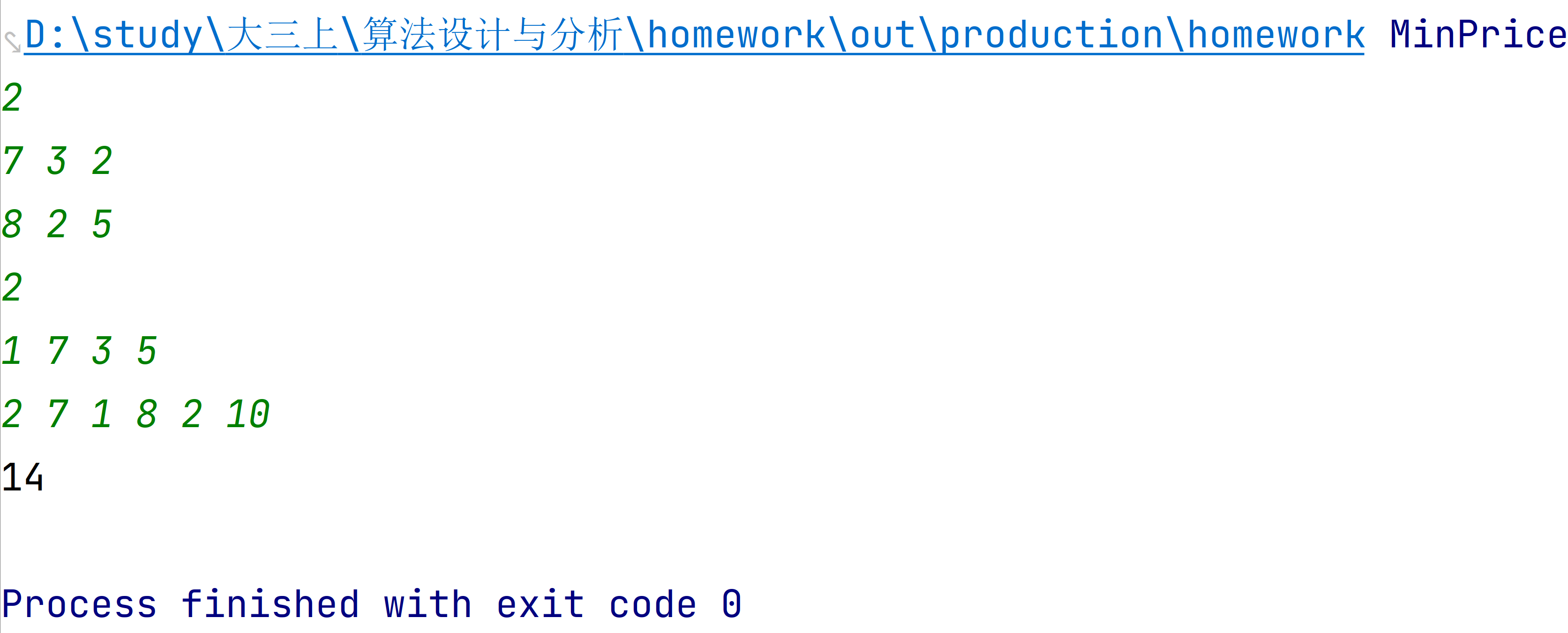
# 主干代码说明

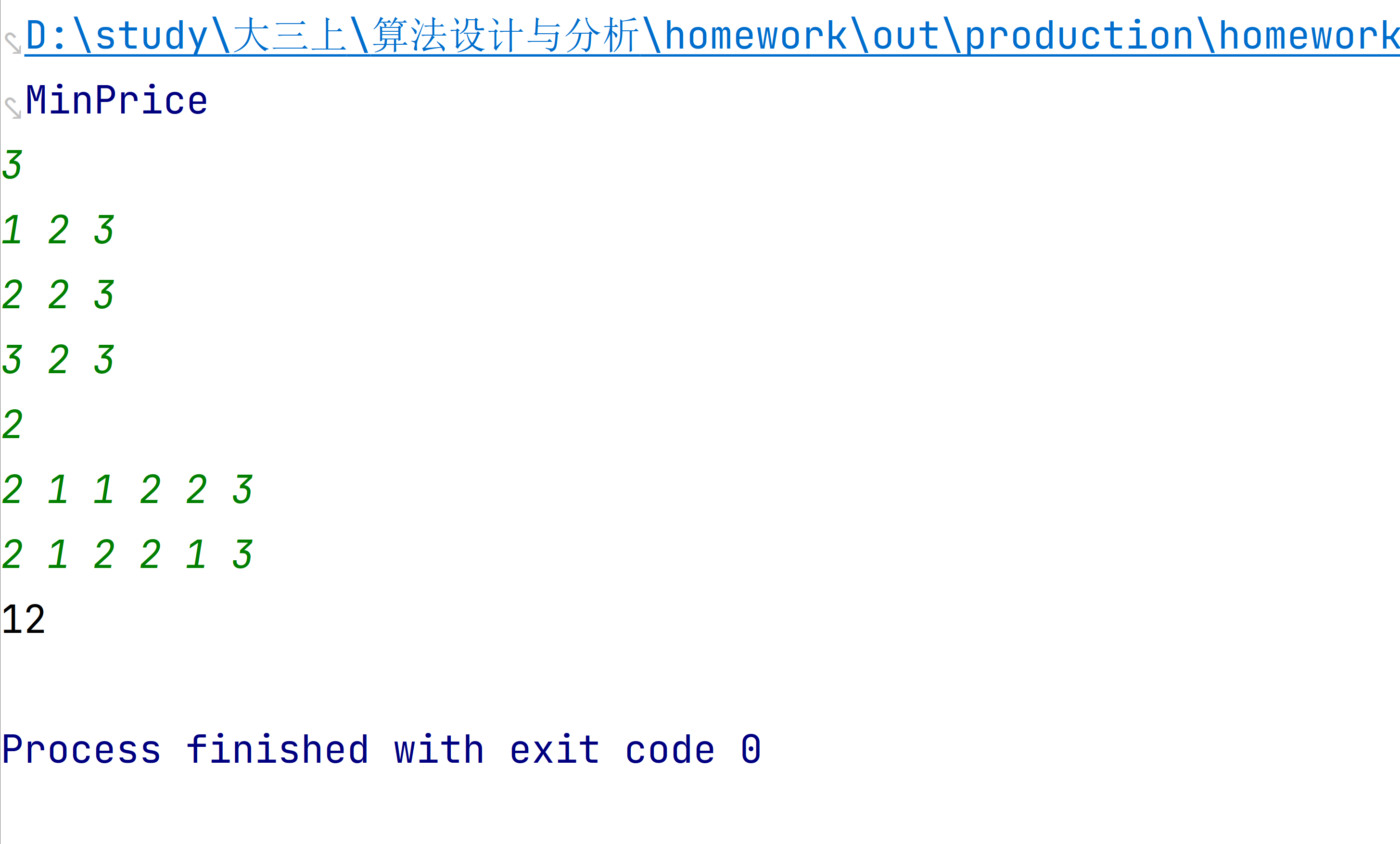






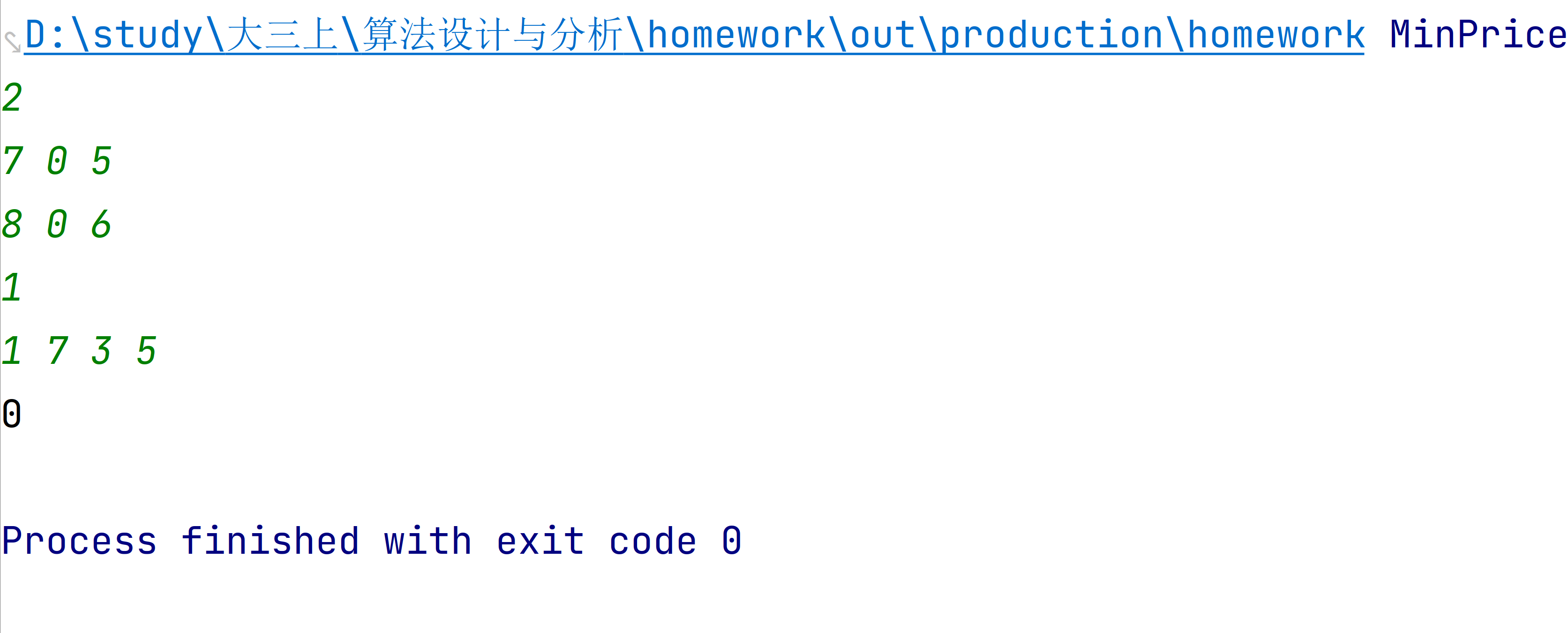
# 运行结果展示





# 异常处理

购买量为0时，应花费0元。



# 代码附录

import java.util.Scanner;

/\*\*

\* @author IDK You Yet

\*/

public class Homework3to14 {

}

class Commodity {

/\*\*

\* 购买数量

\*/

int piece;

/\*\*

\* 购买价格

\*/

int price;

}

class MinPrice {

/\*\*

\* 商品编码的最大值

\*/

private static final int MAX\_CODE = 999;

/\*\*

\* 优惠商品组合数

\*/

private static final int SALE\_COMB = 99;

/\*\*

\* 商品种类

\*/

private static int KIND = 5;

/\*\*

\* 购买某种商品数量的最大值

\*/

private static int QUANTITY = 5;

/\*\*

\* 购买商品种类数

\*/

private static int b;

/\*\*

\* 当前优惠组合数

\*/

private static int s;

/\*\*

\* 记录商品编码与商品种类的对应关系

\*/

private static int[] num = new int[MAX\_CODE + 1];

/\*\*

\* 记录不同种类商品的购买数量

\*/

private static int[] product = new int[KIND + 1];

/\*\*

\* offer[i][j]: 商品组合的优惠价(j=0)；某种优惠组合中某种商品需要购买的数量(j>0)

\*/

private static int[][] offer = new int[SALE\_COMB + 1][KIND + 1];

/\*\*

\* 记录不同商品的购买数量和购买价格

\*/

private static Commodity[] purch = new Commodity[KIND + 1];

/\*\*

\* 记录本次购买的总花费

\*/

private static int[][][][][] cost = new int[QUANTITY + 1][QUANTITY + 1][QUANTITY + 1][QUANTITY + 1][QUANTITY + 1];

private static void init() {

Scanner input = new Scanner(System.in);

int i, j, n, p, t, code;

//置0初始化

int length = 100;

int times = 6;

for (i = 0; i < length; i++) {

for (j = 0; j < times; j++) {

offer[i][j] = 0;

}

}

for (i = 0; i < times; i++) {

purch[i] = new Commodity();

purch[i].piece = 0;

purch[i].price = 0;

product[i] = 0;

}

//从控制台录入数据

b = input.nextInt();

for (i = 1; i <= b; i++) {

code = input.nextInt();

purch[i].piece = input.nextInt();

purch[i].price = input.nextInt();

//一个数组实现的哈希，目的是在O(1)的时间复杂度找到对用商品编码所在的数组下标

num[code] = i;

}

s = input.nextInt();

for (i = 1; i <= s; i++) {

t = input.nextInt();

for (j = 1; j <= t; j++) {

n = input.nextInt();

p = input.nextInt();

offer[i][num[n]] = p;

}

offer[i][0] = input.nextInt();

}

}

/\*\*

\* 用于确定对于b种商品，给定每种商品的购买量，其最小花费

\* 即确定子问题的最优解

\*/

private static void minicost() {

//已经购买1~5类商品的数量

int i, j, k, m, n;

//找到对应优惠后的花费

int p;

//最小花费

int minm = 0;

//将最小花费初始化为没有优惠策略时的花费

for (i = 1; i <= b; i++) {

minm += product[i] \* purch[i].price;

}

//对s中优惠策略依次讨论

for (p = 1; p <= s; p++) {

/\*\*

\* 第一种商品扣除当前优惠组合后的剩余购买量（offer[i][j]: 商品组合的优惠价(j=0)；某种优惠组合中某种商品需要购买的数量(j>0)）

\*/

i = product[1] - offer[p][1];

j = product[2] - offer[p][2];

k = product[3] - offer[p][3];

m = product[4] - offer[p][4];

n = product[5] - offer[p][5];

//判断在当前优惠组合下，购买的商品总花费是否比没有优惠政策的花费少

if (i >= 0 && j >= 0 && k >= 0 && m >= 0 && n >= 0 && cost[i][j][k][m][n] + offer[p][0] < minm)

//购买优惠组合前的花费+当前优惠组合花费

{

minm = cost[i][j][k][m][n] + offer[p][0];

}

}

//子问题最优组合花费

cost[product[1]][product[2]][product[3]][product[4]][product[5]] = minm;

}

/\*\*

\* 对于第i类商品,如果i大于最大商品种类b时，计算此时组合的最小花费

\*/

private static void comp(int i) {//i类商品

//确定一个子问题，计算一次当前最小花费

if (i > b) {

minicost();

return;

}

/\*\*

\* purch[i].piece表示第i类商品的最大数量

\*/

for (int j = 0; j <= purch[i].piece; j++) {

/\*\*

\* 记录第i类商品购买数量j的情况

\*/

product[i] = j;

/\*\*

\* 控制遍历所有的商品类

\*/

comp(i + 1);

}

}

/\*\*

\* 输出结果

\*/

private static void out() {

System.out.println(cost[product[1]][product[2]][product[3]][product[4]][product[5]]);

}

public static void main(String[] args) {

init();

/\*\*

\* 从第一类商品开始

\*/

comp(1);

out();

}

}