# 最小购物问题算法设计

每一个买商品的用户都会有所需购买的数量，每个商品也会有它对应的价格，所以我们定义一个类来存放它们。

**class** CommodityCommodity{

**int** piece;

**int** price;

}

同时，商品自然要有编码来区分种类，优惠也要有优惠的组合方式在这里，我们将商品编码的最大值定义在三位数以内，虽然并不一定会用到这么多

**private** **static** **int** *MAXCODE* = 99;

同理定义，优惠商品组合数，商品的种类和可以购买的最大值、以及定义两个量来存放实例化的商品种类数和组合优惠数。同时记录购买数量和它们之间的关系。

**private** **static** **int** *SALECOMB* = 99;

**private** **static** **int** *KIND* = 5;

**private** **static** **int** *QUANTITY* = 5;

**private** **static** **int** *b*;

**private** **static** **int** *s*;

**private** **static** **int**[] *num* = **new** **int**[*MAXCODE*+1];

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

同时，需要定义一个数组，来记录在某一状态下，该购物的最少总花费

**private** **static** **int**[][] *offer* = **new** **int**[*SALECOMB*+1][*KIND*+1];

**private** **static** CommodityCommodity[]

*purch* = **new** CommodityCommodity[*KIND*+1];

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

当然，算法对问题的处理过程才是最关键的。我们定义一个数组，让让它从0开始，对要购买的商品从小到大的遍历，有优惠方案的就用优惠方案，然后记录下在每一个当前状态下的最优的那个方案，同时如果有更优的则替换它

**private** **static** **void** minicost(){

**int** i,j,k,m,n,p,minm; minm = 0;

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

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

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

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];

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

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

}

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

}

在这里，我们对一个购物方案的选择其实是受前一个状态的影响的，所以从小到的的规划遍历，就省去了很多重复的计算从而降低了算法的复杂度。

最后就是将计算得到的结果输出

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