**算法设计与分析**

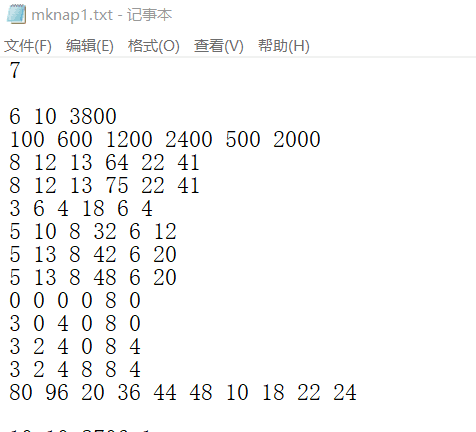
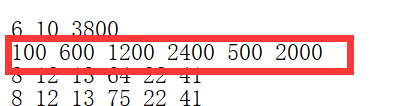
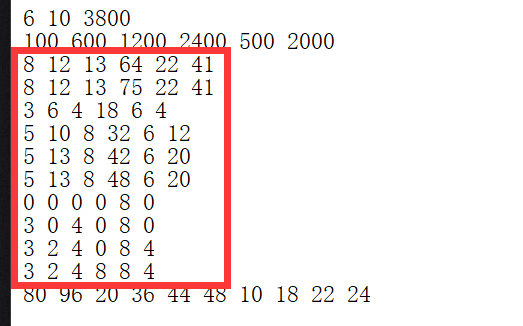
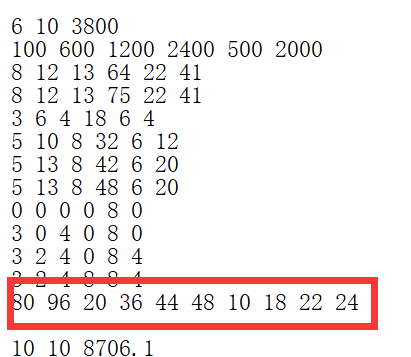
计算机171班 黄潆 201706401142

**一、题目分析**

该多维背包问题给出了7个实例，每个实例中定义了，我们需要根据所给的实例数据，设计相应算法，求出满足所有约束条件的背包的最大价值。

**二、算法设计与分析**

我使用JAVA语言来设计算法，运用回溯法来找出最优解，根据背包个数、背包的属性个数、背包的价值、背包的所有属性值、每个属性总的约束条件求出物品总和最大值。

C:\Users\wwww\Documents\Tencent Files\2112918682\Image\C2C\99009DCDD8F2F963350BC9EC4D49F326.JPG  
这是有6组数据（6个物品）   10个维度  最优价值是3800     
  
这是那6个物品的价值      
  
这里是10行对应上面的10维     
  
最后一行是那十个维度的上限，也就是说选取的物品中上面第一行的和不能超过80，第二行不能超过96，……以此类推。

和0-1背包问题不同的是，这是多维背包问题，需要定义多维数组来存储数据。如下；

static double bestp,mis; //最优总价值，当前背包最大价值

static int [] vs=new int[100]; //当前的选取状态

static int [] best=new int[100]; //存储最佳路径

static int [] bbs=new int[100]; //当前所选物品属性总和

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int[] n =new int[10];

int[] l =new int[100]; //n和l代表物品数量和有多少行的物品

double[][] value = new double[10][200]; //value是物品价值

int[][][] du = new int[10][200][200]; //du是存储物品质量

int[][] K = new int[10][200]; //K是背包容量上限

定义物品的数量，物品属性的数量，物品的价值，物品的属性值，各属性的约束值，当前的选取状态，最优的选取状态，当前背包的最大价值，最大价值和当前所选物品的属性值总和。

回溯法解决多维背包问题是一个子集选取问题，适合于用子集树表示多维背包问题的解空间。在搜索解空间树时，只要其左儿子节点是一个可行节点，搜索就进入左子树，在右子树中可能含有最优解时才进入右子树搜索。否则将右子树剪去。

回溯法是判断已有背包是否满足条件，如果满足则装入物品，递归下一物品，不符合直接递归。判断已放入的物品是否超过容量最大值。如下：

int flag=0; //回溯判断一下是否所有的背包都符合条件，如果符合则装入物品，递归下一物品，不符合则直接递归,通过判断来知道是否要将物品放入背包。

for(int i=0;i<l;++i){

if(bbs[i]+du[i][t]>K[i]){

flag=1;

break;

}

}

如果没有，那么开始进行遍历，1表示物品放入背包中，0表示物品不放入，首先需要判断是否有物品可以放入背包中，要是没有物品可以放入背包中，直接返回结果，当前的价值即为背包的最优价值。然后判断左节点是否为可行节点，当左节点可行，就优先搜索左子树，不满足则进入右子树。当达到物品的数量上限，直接返回当前物品数量的结果；若左右节点同时不可行，则直接将结果返回上一步。以此类推，将结果从下至上一层一层进行返回，得到问题的最优结果。

/\*搜索左子树\*/

if(flag==0){

vs[t]=1; //装入物品搜索左子树

for(int i=0;i<l;++i)

bbs[i]+=du[i][t]; //表示当前属性值总和加上下一个属性值

mis+=value[t]; //更新当前背包的最大价值

Backtrack(t+1,value,n,l,K,du); //调用回溯法

for(int i=0;i<l;++i)

bbs[i]-=du[i][t]; //当前属性值总和减去上一个属性值

mis-=value[t]; //当前背包的最大价值恢复为原来的价值

}

/\*搜索右子树\*/

if (Bound(t+1,n,value)>bestp){

vs[t]=0; //不装入这个物品，直接搜索右边的节点

Backtrack(t+1,value,n,l,K,du); //调用回溯法

}//判断左节点是否为可行节点，当左节点可行，就优先搜索左子树，不满足则进入右子树。

}

**三、程序输出**

本程序将输出背包在装入满足各项约束条件的物品时最大价值，以及各个物品此时的装入情况（0表示不装入背包，1表示装入背包），装入背包的物品及物品个数。

**四、运行实例**

输入：

6 10

8 12 13 75 22 41

3 6 4 18 6 4

5 10 8 32 6 12

5 13 8 42 6 20

5 13 8 48 6 20

0 0 0 0 8 0

3 0 4 0 8 0

3 2 4 0 8 4

3 2 4 8 8 4

80 96 20 36 44 48 10 18 22 24

输出：

装入的物品[0 1 1 0 0 1]

最优总价值为3800.0

即：将第2、3、6、共3个物品装入背包，如下：



**五、算法分析复杂度**

空间复杂度：

一个二维数组存储n个物品的l种属性，所以算法的空间复杂度为O(n\*l)。

时间复杂度：

每个物品只有装入或不装入两种状态总共有n个物品，遍历这些物品的子集树算法时间复杂度为O(2n)，但由于每个物品有l个约束，所以算法中执行次数最多的语句为l\*2n次，即算法的时间复杂性为O(l\*2n)。（注意l是字母）