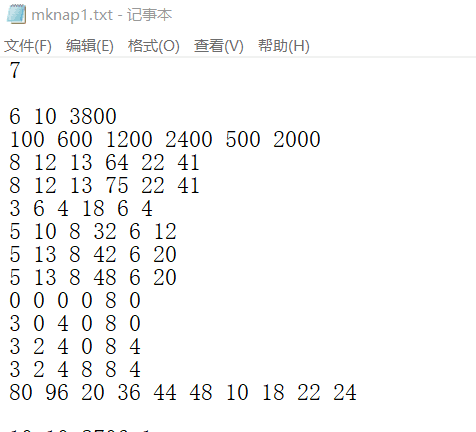
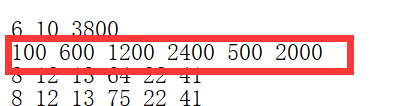
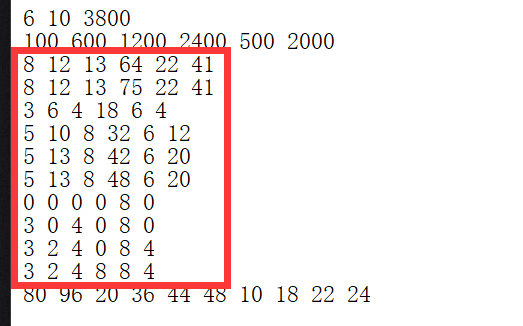
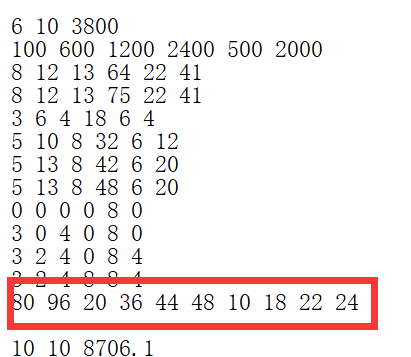
一、题目分析

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 int [] vs=new int[51];

static double bestn,cp;

static int [] best=new int[51];

static int [] sw=new int[51];

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int[] n =new int[7];

int[] l =new int[51];

double[][] value = new double[7][151];

int[][][] wei = new int[7][151][151];

int[][] K = new int[7][151];

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。

回溯法的根本思想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 j;//已搜索到根节点

if(t>=n){

if(cp<=bestn) return;//如果搜索到叶子节点

for(j=0;j<n;++j)

best[j]=vs[j];//最大价值变成当前背包得总价值，保存最优选取方案

bestn=cp;

return;

}

int flag=0;

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

if(sw[i]+wei[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。

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。回溯法解决多维背包问题的时间复杂度和空间复杂度均为O(n)。