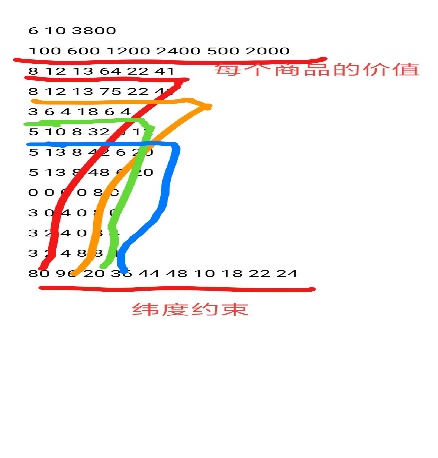
算法设计分析与文档

首先，我们要了解什么是多维背包问题（MKP问题）。多维背包问题属于组合最优化问题，一般的最优化问题由目标函数和约束条件组成。将满足所有条件的解的空间称为可行域，可行域中的解称为可行解，将可行域中使目标函数最大的解称为最优解。MKP本质是在满足某些资源约束的条件下，从候选对象集中找出能够使使总的利益函数最大的一个对象子集。多选择背包问题定义为有附加约束条件的背包问题，该问题带有互不相关的多选择约束。该问题的特点是只有一个承重有限的背包，将要放入背包的物品被分为相互排斥的若类，而每类中有若干不同的项目。

多维多选择背包问题是一类特殊的0-1背包问题。问题的描述如下：存在m个背包，其称重分别是Wk（k=1，2，3…m）和n类物品，每类物品又分别有Ii（i=1,2,…n）个物品，其价值分别为Vij（j=1,2,…）而对每一个物品，由于其装入的背包不同而其重量也有所不同，分别为Wijk,该问题要求每一类只选择装入一个物品，在满足背包称重的限制下最大化装入背包的物品总价值。

接着我们先把题目的的第一组数据弄懂。问题描述：给出了6个商品，有10个维度，每个商品有不同的对应价值，3800是其最大价值，最后一行是10个维度的约束。然后选出第三行的值要小于最后一行的第一个数，第四行的值要小于最后一行的第二个，以此类推，并且选的时候要同时考虑十个上限有没有超出约束限制，难点就在这里。其他组数据理解都按这个思路来。伪代码如下：

private boolean loading(double[] curslimitg,int m,int[] limit,int j,double[][] alllimit){

//用函数来判断每一个当前限制是否大于最大限制

int i;

if(curslimit[i]+alllimit[i][j]<limit[i]) 判断属可行性约束条件，判断限界是否每一个都遍历一遍

else break; 如果没有就跳出

return true; 可以继续向下执行

else

return false; 否则就结束

然后我用的是回溯法解决问题的。回溯法的基本思想是对于用回溯法求解的问题，首先要将问题进行适当的转化，得出状态空间树。这棵树的每条完整路径都代表了一种解的可能。通过深度优先搜索这棵树，枚举每种可能的解的情况；从而得出结果。但是，回溯法中通过构造约束函数，可以大大 提升程序效率，因为在深度优先搜索的过程中，不断的将每个解（并不一定是完整的，事实上这也就是构造约束函数的意义所在）与约束函数进行对照从而删除一些 不可能的解，这样就不必继续把解的剩余部分列出从而节省部分时间。

利用回溯法解题的具体步骤

(1)描述解的形式，定义一个解空间，它包含问题的所有解。

(2)构造状态空间树。

(3)构造约束函数（用于杀死节点）。

伪代码：private void backtrack(int a) { //调用回溯法

if (a >= n) {

采用限界函数-剪枝操作，剪去不是最优解的树枝

避免无效搜索，提高回溯法效率

if ( curvalue > bestvalue) { //如果当前价值大于最优价值，赋值

bestvalue= (float) curvalue;

// 保存最优的选择值

bestchoice[i] = choice[i];

// 把最优的选择序列保存在best\_choice中

else {

// 搜索左子树

if (loading(curslimit,m,maxlimit,a,alllimit)) {

//计算限制条件

backtrack(a + 1); //第a+1的回溯

// 背包当前限制和所有限制还原，遍历右子树

choice[a] = 0;

backtrack(a + 1); // 搜索右子树

结果调试截图：

