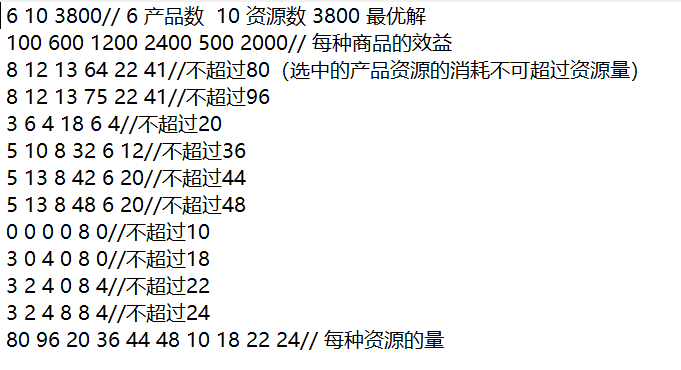
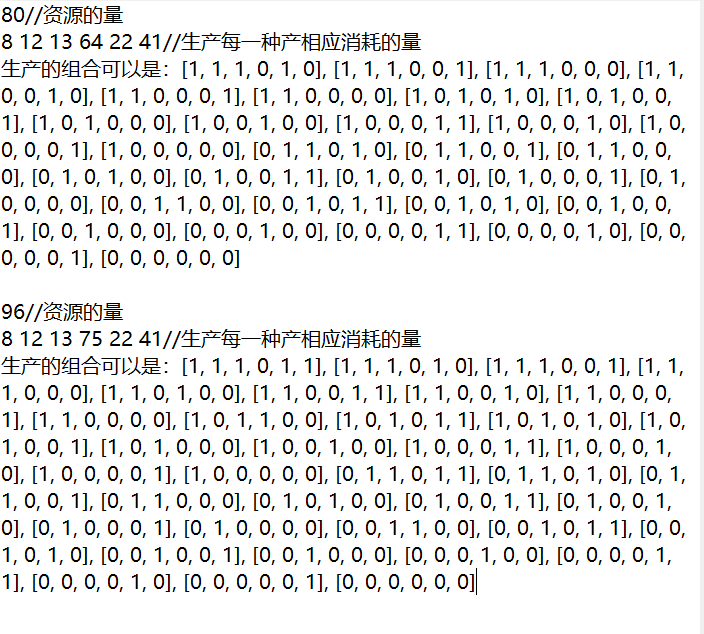
算法设计与分析文档

MKP问题即多维背包问题，在背包问题基础增加多维的约束，将背包的约束条件从一个扩展到大小各异的m个约束，是标准背包问题的一般化。多维背包问题的目标是将n个物体装入到有m个约束条件的背包中，装入尽可能多的物体，使得这些物体总的价值最大化。从产品和资源的关系解释多维背包问题：给定有限数量的m种资源，可以生产n种产品，每一种产品都有一定的收益，每生产一种产品就会消耗m种资源，在有限的资源下生产产品使得产品效益达到最大。MKP测试算例的格式说明：



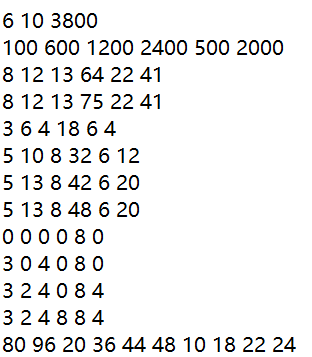
多维背包问题m个约束条件的每一个就是一个背包约束，符合每一个约束都有不同的装入物体的组合方式，所以多维背包问题也是一个组合优化问题。对于这个问题我的想法是首先利用回溯法找出所有符合条件约束的组合，例如下图：



（1表示生产，对应背包问题即放入背包；0表示不生产，对应背包问题即不放入背包）

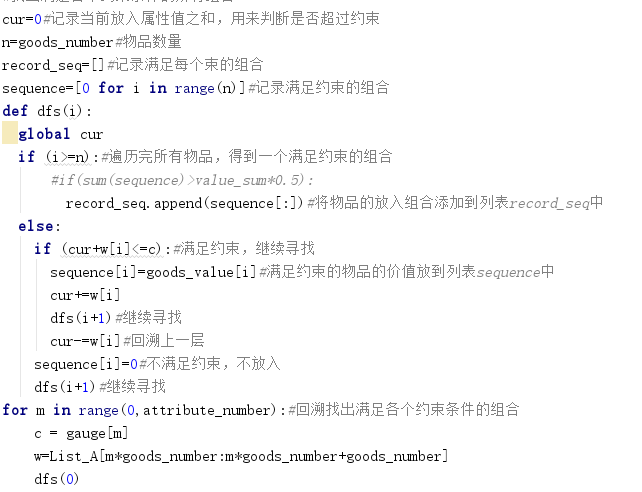
通过上图我们可以发现上面两组满足条件的组合拥有交集，比如说[1,1,0,0,0,0]、[1,0,0,0,0,0]、[1,1,0,0,0,1]等等，这说明了有些组合既满足第一个约束（资源量为80）又满足第二个组合（资源量为96）。相应地，对于m个约束的背包问题，我们需要找出同时满足m个约束的组合，分别算出满足m个约束的放入背包的物品总价值，求最大值，这个最大值就是最优解。

例如下图算出同时满足10个约束条件的组合是：[1, 1, 1, 0, 1, 0], [1, 1, 1, 0, 0, 0], [1, 1, 0, 0, 1, 0], [1, 1, 0, 0, 0, 1], [1, 1, 0, 0, 0, 0], [1, 0, 1, 0, 1, 0], [1, 0, 1, 0, 0, 1], [1, 0, 1, 0, 0, 0], [1, 0, 0, 0, 1, 1], [1, 0, 0, 0, 1, 0], [1, 0, 0, 0, 0, 1], [1, 0, 0, 0, 0, 0], [0, 1, 1, 0, 1, 0], [0, 1, 1, 0, 0, 1], [0, 1, 1, 0, 0, 0], [0, 1, 0, 0, 1, 1], [0, 1, 0, 0, 1, 0], [0, 1, 0, 0, 0, 1], [0, 1, 0, 0, 0, 0], [0, 0, 1, 0, 1, 1], [0, 0, 1, 0, 1, 0], [0, 0, 1, 0, 0, 1], [0, 0, 1, 0, 0, 0], [0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 1, 1], [0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 1], [0, 0, 0, 0, 0, 0]；其中放入组合为[0,1,1,0,0,1]，总价值最大，价值为3800，所以最优解是3800。



回溯法可以求出满足约束的产品生产组合，回溯的思想是：构造解的子集树。对于每一个产品i，对于该产品只有生产与不生产2个决策，总共有n个产品，可以顺序依次考虑每个产品，这样就形成了一棵解空间树： 基本思想就是遍历这棵树，以枚举所有情况，进行判断，如果资源消耗量没有超过资源量的话，该组合就是满足约束的组合，将该生产组合保存下来。

下面是我的程序的回溯法的复杂度分析：



回溯法因为物品只有选与不选2个决策，而总共有n个物品，时间复杂度为O(2n)。因为递归栈最多达到n层，而且存储所有物品的组合信息需要两个列表，一个列表用于存放物品放入组合的信息（如上图的列表sequence），另一个列表存储存放了物品组合信息（如上图的列表record\_seq，列表record\_seq存放列表sequence），sequence中存放物品个数n，空间复杂度为O(n)，record\_seq存放了len(record\_seq)个列表sequence，空间复杂度为len(record\_seq)\*O(n),所以最终空间复杂度为O(n)。

利用回溯法解约束条件和物品数都比较少多维背包问题挺有效的，但是一旦产品数和约束条件变得很大，找出满足所有约束的组合将会占用大量内存资源和cpu运算资源，求出最优解的时间将会变得很长。如果能找到一种快速定位满足所有约束的组合或者知道最优解所在区间，这将会提高算法的运行效率。

代码源码：

*#--------------------------  
#读取input.txt的数据*data=open(**"input.txt"**)*#打开input.txt文本*load\_data=data.read().split()*#读取文件*List\_A=[]*#存储从文本读取进来的数据***for** i **in** load\_data:*#将文本数据添加到列表List\_A* List\_A.append(i)  
List\_A = [ float(x) **for** x **in** List\_A]*#将列表中的数据类型转换为float*goods\_number=int(List\_A[0])*#读取List\_A的第一个元素，即物品的个数*attribute\_number=int(List\_A[1])*#读取List\_A的第二个元素，即约束个数*goods\_value=List\_A[3:goods\_number+3]*#读取各物品的价值*print(**"物品数:"**,goods\_number,**"约束个数:"**,attribute\_number)  
print(**"物品价值："**,goods\_value)  
*#------------------------------***del** List\_A[0:goods\_number+3]*#把读取到的商品数、属性个数以及商品价值删除，方便下面的数据读取操作*List\_A.reverse()*#反转列表*bounds=List\_A[0:attribute\_number]*#读取约束*bounds.reverse()  
print(**"约束值:"**,bounds)  
*#------------------------------***del** List\_A[0:attribute\_number]*#读取完限界条件后将其从列表中减去*List\_A.reverse()*#恢复列表，列表中剩下属性的具体数值*data.close()  
**for** j **in** range(0,attribute\_number):  
 print(**"第"**,j+1,**"行属性值:"**,List\_A[j\*goods\_number:j\*goods\_number+goods\_number])  
*#------------------------------  
#找出满足各个约束条件的所有组合*cur=0*#记录当前放入属性值之和，用来判断是否超过约束*n=goods\_number*#物品数量*record\_seq=[]*#记录满足每个束的组合*sequence=[0 **for** i **in** range(n)]*#记录满足约束的组合***def** dfs(i):  
 **global** cur  
 **if** (i>=n):*#遍历完所有物品，得到一个满足约束的组合* record\_seq.append(sequence[:])*#将物品的放入组合添加到列表record\_seq中* **else**:  
 **if** (cur+w[i]<=c):*#满足约束，继续寻找* sequence[i]=goods\_value[i]*#满足约束的物品的价值放到列表sequence中* cur+=w[i]  
 dfs(i+1)*#继续寻找* cur-=w[i]*#回溯上一层* sequence[i]=0*#不满足约束，不放入* dfs(i+1)*#继续寻找***for** m **in** range(0,attribute\_number):*#回溯找出满足各个约束条件的组合* c = bounds[m]  
 w=List\_A[m\*goods\_number:m\*goods\_number+goods\_number]  
 dfs(0)  
*#-------------------------------  
#从record\_seq中找出满足所有约束的组合*size=len(record\_seq)  
combination=[]*#存放所有符合的组合*value=[]*#存放总价值  
#print(record\_seq)***for** k **in** range(0,size):*#遍历record\_seq的组合* **if**(record\_seq[k] **in** combination):*#如果该组合已经存在列表combination，跳出该层循环* **continue  
 elif**(record\_seq.count(record\_seq[k])==attribute\_number **and** record\_seq[k] **not in** combination ):*#如果组合出现的次数等于约束个数，说明该组合* val = sum(record\_seq[k])*# 满足所有约束条件，将它放入列表combination中，计算总价值放到列表value中* combination.append(record\_seq[k])  
 value.append(val)  
 *#print(val)* **else**:*#如果该组合不存在列表combination中，出现的次数也不等与约束个数，跳出该层循环* **continue  
del** record\_seq[:]  
*#-------------------------------  
#计算最大值，并找出最大值的组合  
#print("\n")  
#print("符合所有约束的组合:",combination)*max\_value=max(value)*#求出value中的最大值，该值是最优解*index=value.index(max\_value)*#求出最大值元素的下标，用来读取组成最大值的组合  
#print("\n")*print(**"最优解:"**,max\_value,**",组合:"**,combination[index])