**算法设计与分析**

1. **分析题目**：题目为多维背包问题，其实我们也可以叫它为多约束背包问题，它带有一组约束，比如说重量、尺寸、可靠性、体积等等。该问题也可以简单描述为n个物品要放入m个称重不同的背包，它与0-1背包问题不同的是，物品放入不同背包的装量是不同的，首先我们要确定是否放入背包，还要确定我们要放入哪个背包，最终目的就是求解我们物品最高的价值比重，也就是最优解。
2. **分析数据：**

我们以第一份数据为例子：

变量数n：6 约束数m：10 最终价值：3800

放入背包的价值:100 600 1200 2400 500 2000

十行约束：8 12 13 64 22 41

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

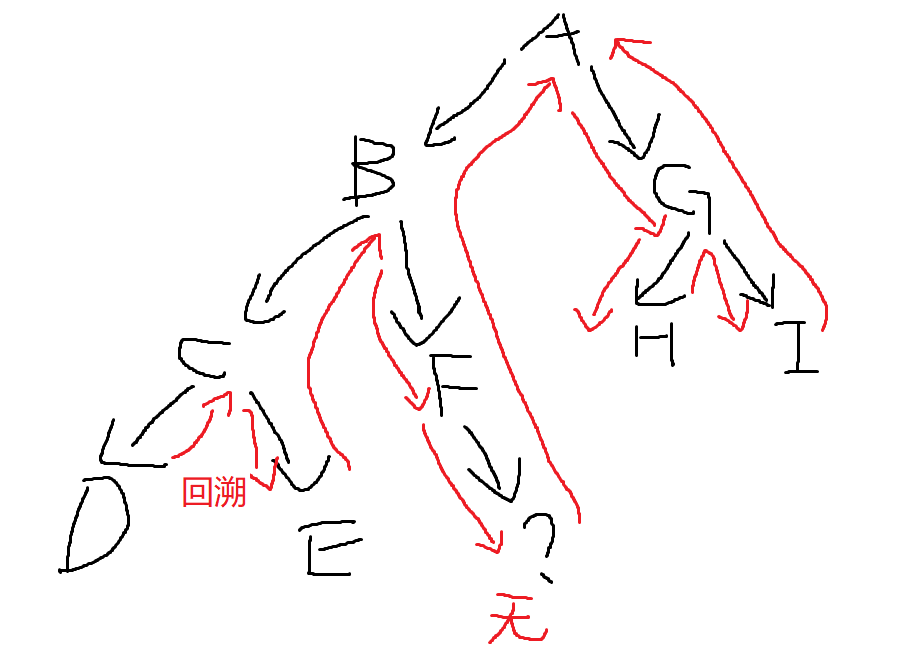
3 2 4 0 8 4

3 2 4 8 8 4

约束条件：80 96 20 36 44 48 10 18 22 24

1. **设计算法：**我们综合运用所学过得算法设计与分析方法，选择了**回溯法**来解决我们的多维背包问题，回溯法是一种以深度优先方式系统搜索问题解的算法，它的基本思想是确定解空间的组织结构后，回溯法从开始结点（根结点）出发，以深度优先方式搜索整个解空间，每次扩大当前部分解时，都面临一个可选的状态集合，新的部分解就通过在改集合中选择构造而成。

我们设计算法的时候充分观察了数据的结构和求解多维背包问题的步骤，设计出代码的结构层次，就是往下搜索的时候，到达最底层我们就要返回，那么这就是回溯法。

**回溯法流程：**

A→B→C→D→C（回溯）→E→B(回溯)→F→?→A(回溯)→G→H→G（回溯）→I→A(回溯)

**伪代码：**

import pandas numpy #导入两个第三方库

n,m=input() #输入变量数(n)和约束数(m)

value=list(input()) #接收物品的价值，定义为列表

for i in range(m):

a=list( input()) #接收输入的属性

data\_1.append(a)

data = pd.DataFrame(data\_1) #将属性变成数据帧

b=list(input()) #接收属性的约束条件

data['约束条件']=b #将属性的约束条件添加到数据帧

v=0

max\_1 #定义临时价值最终价值

def recall(i)

if i >n-1: #判断是否到达叶子结点

if v>max\_1: #判断临时价值是否大于最终价值

max\_1=v #交换

else:

c=(data['约束条件']-data[i])>=0 #约束条件 - 属性的值

for d in range(len(c)):

if x==False:

y=0 #for循环里面出现一次fales，即不放入背包

if y==1:

data['约束条件']-=data[i] #约束条件 - 属性的值

v+=value[i] #临时价值升高

recall(i+1) #调用方法

v-=value[i] #临时价值降低

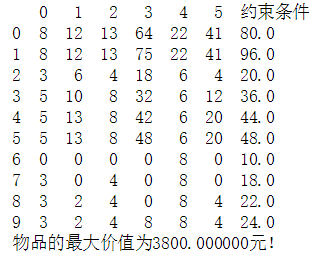
data['约束条件']+=data[i] #约束条件 + 属性的值

recall(i+1) #递归调用

recall(0) #开始调用函数

print(%max\_1) #输出最终价值

1. **代码设计：**我们使用的是python程序设计语言，它是一门易读、易维护，并且被大量用户所欢迎的、用途广泛的语言。我们代码开头部分是输入了两个第三方库，调用了他们里面的一些函数，方便快捷的实现一些特定功能，接下来是数据的输入，再把数据变成数据帧，数据帧如下图，用于将约束对应上相应的价值，以便后面的计算，然后定义一个函数，参数为i，用于判断是否到达叶子结点，和 判断临时价值是否大于最终价值，大于就交换，如果达到了叶子结点就会回溯到上一层，判断n-1层是否有结点，有就减掉相应的属性值，临时价值就加上原来减掉的属性值，然后调用一下函数判断是否到了另一个叶子结点，没有的话继续回溯到上一层，以此类推，最后调用一下函数令参数i=0，开始计算价值，比较出最大值，也就是最优解，最后输出物品的最大价值。

**输出结果**如下图：

1. **算法复杂度：** O()。由于代码中只出现过几次if语句的判断，还有两次for循环，第一个是简单的输出将要输入的属性，第二个是循环穷举一下c的值，使算法复杂度增大的是递归调用了两次函数，使得运算过程呈现了一个指数的增长，变为O()，就是先后递归函数来判断是否到达叶子结点和临时价值是否大于最终价值，然后执行下面的代码，下面再调用函数就是下一组约束，进行一系列的计算和比较，也就是约束条件的相加减，里面也是运用到了一些穷举法把所有的物品价值都列举出来，然后经过比较，选出的最大物品价值。

计算机172班

林盛斌

201706401205

**附件：**

Python源码：

import pandas as pd

import numpy as np#导入两个第三方库

n,m=map(int,input("请设定变量数(n)和约束数(m):").split())

#首先设定一下变量数和约束数的数量，都为整数，split（）为可以输入多个数

print("请输入%d个物品价值:"%n)

value=list(map(float,input().split()))#用来接收物品的价值，定义为列表

data\_1=[] #定义一个空列表，用来存储下面将要输入的属性

for i in range(m): #利用for循环来显示将要输入第几个属性

print("请输入第%d个属性:"%(i+1))

a=list(map(int, input().split()))#用来接收输入的属性，定义为列表

data\_1.append(a)

#将输入的属性添加到刚刚的空列表中，为建立二维表做好准备，相当于一个中间值

data = pd.DataFrame(data\_1)#将属性变成数据帧（二维数组表格）

print("请输入%d个属性的约束条件:"%m)

b=list(map(float,input().split())) #用来接收属性的约束条件，定义为列表

data['约束条件']=b #将属性的约束条件添加到data数据帧中成为一列，并与每行的属性相对应

print(data) #查看数据帧

v=0 #用于临时存储物品价值的变量（临时价值）

max\_1=value[0] #我们最终要求的最高价值（最终价值）

def recall(i):#定义一个函数f，用于回溯，参数为i

global v,max\_1 #调用全局变量

y=1 #默认放入背包

if i >n-1: #判断是否到达叶子结点，到了就回溯到上一层

if v>max\_1: #判断临时价值是否大于最终价值

max\_1=v #把临时价值交换给最终价值

else:

c=0

c=(data['约束条件']-data[i])>=0 #约束条件 - 属性的值，赋值给c

c=np.array(c) #把c转为数组

for d in range(len(c)):#循环次数为c的长度

x=c[d] #穷举c每次的

if x==False: #循环判断c中是否有false

y=0 #如果for循环里面有一次fales，物品就不放入背包

if y==1: #判断，如果c全部为True才为真，进行下面，反之执行if下面的代码

data['约束条件']-=data[i] #当一个属性满足时，约束条件就要减掉它原本的值

v+=value[i] #零时价值要加上属性对应那个价值

recall(i+1)#调用方法,判断是否是叶子结点

v-=value[i] #回溯到上一层，减掉原来的价值

data['约束条件']+=data[i] #回溯以后，约束条件又得以增加

recall(i+1) #递归调用，回溯到上一层，继续原来的方法运算

recall(0) #开始调用函数，从0开始，0 1 2 3 4 5

print("物品的最大价值为%f元！"%max\_1)