# 多维背包问题算法设计与分析

**问题描述：**

设共n种物品，每种物品都有体积，重量，数量，价值等m种限制条件。问如何选择物品，使得带上的物品总价值最大，这个最大总价值为多少？

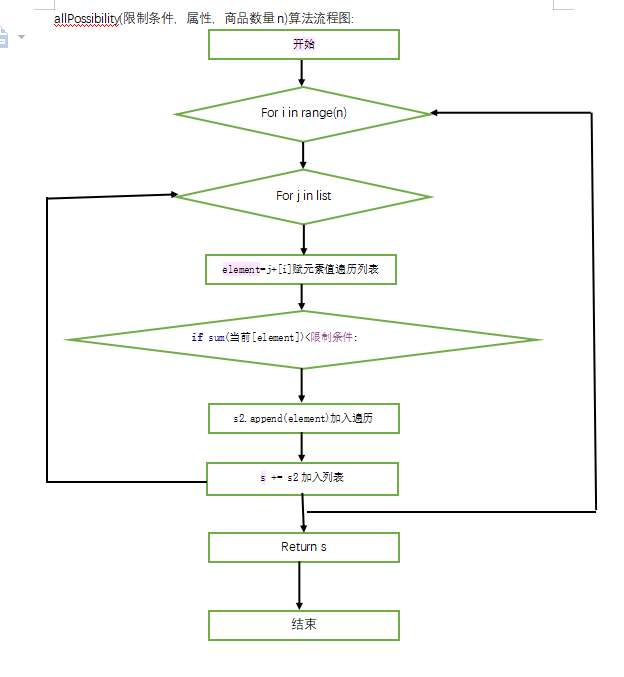
**算法设计：**

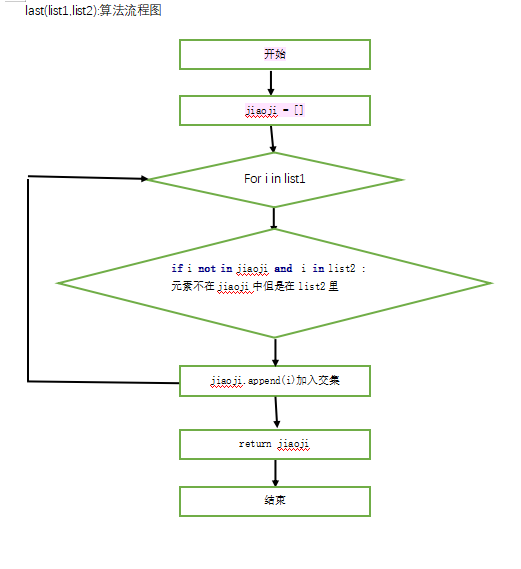
多维背包问题是常规0-1背包问题的扩展问题，它是带有一组约束条件（重量 尺寸 可靠性等）的背包问题，解决多维背包问题仍要基于0-1背包问题的基础上进行求解，物品仍有放与不放两种选择，最终目标仍是求解出使得物品总价值最大的选择。多维背包问题与0-1背包问题不同的是，物品放入不同背包的重量是不同的。

我选择用Python来解决这个问题，因为Python提供了许多对数据进行操作的库及许多对数据进行相关处理的函数，在多维背包问题中有许多限制条件、属性、商品个数等数据，利用Python能简单高效的对这些数据进行管理。

在解决这个问题的算法中我引入了交集的概念，整个算法的思路是通过一个普通的找遍历和交集算法，取上一个满足限制条件的组合，和下一个满足限制条件的组合的交集，一直交集，交到最后，剩下都满足所有限制条件的组合，再取里面的最大值，即是我们所求的最佳价值。

流程分析图：





**def** allPossibility(restrict,attribute, nums):parameter1 = [[]]  
 element = []  
 **for** i **in** range(nums):  
 parameter2 = []  
 **for** j **in** parameter1:  
 element=j+[i]  
 **if** sum(attribute[element])<restrict:  
 parameter2.append(element)  
 parameter1 += parameter2  
 **return** parameter1

第一个算法用来计算所有满足限制条件的组合，两个for循坏和allPossibility算法的调用使得我们能从第0个开始不断寻找满足限制条件的可能性且不断的取各个可能性之间的交集，如找到了第一个可能性之后把它存起来，再找第二个可能性，对第一个可能性和第二个可能性找交集，将找出的交集存到列表里，再调用allpossibility算法找第三个可能性，第三个可能性与之前存入的第一、第二个可能性的交集再找交集，循坏往复，交到最后剩下都满足所有限制条件的组合。

**def** last( aggregate1, aggregate2):  
 *#寻找满足所有限制条件的价值组合* jiaoji = []  
 **for** i **in** aggregate1:  
 **if** i **not in** jiaoji **and** i **in** aggregate2 :  
 jiaoji.append(i)  
 **return** jiaoji

这里使用两个方法将取得的交集存到数组里，用in来进行循坏，判断i在aggregate1这个交集里，同时判断i不在一开始的jiaoji 里且i一定要在aggregate2 交集里，将i加入jiaoji数组里，返回交集，一直返回到最后面的交集就成为满足所有属性的交集。

**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 print(**"正在读取文本数据...."**)  
 f1 = open(**"MKP\_Object input.txt"**) *# 读取文本数据* first\_input = f1.read()  
 all\_data = first\_input.split(**'\n'**) *# 切割文本数据* want\_data = []  
 bestValue=[]  
 data\_1th = list(map(float, all\_data[0].split())) *# 切割商品数据和约束条件数据* commodity\_number = int(data\_1th[0]) *# 读入商品数* constraints\_numbers = int(data\_1th[1]) *# 读入约束条件数* constraints = list(map(float, all\_data[int(len(all\_data) - 1)].split())) *# 读入约束条件* values = list(map(float, all\_data[1].split())) *# 读入商品的价值* cost=pandas.Series(values)  
 **for** i **in** range(2, constraints\_numbers + 2):  
 want\_data.append(list(map(int, all\_data[i].split()))) *# 添加数据到列表中* data\_array=numpy.array(want\_data)

这部分是整个代码的主方法，这些主方法是对数据的读取，多维背包问题需要传入的数据很多，所以读取数据部分显得尤为重要。open函数、read函数等都是读取数据的函数，切割文本数据以回车来切割，这里需要注意的是要保证数据里没有多余的空格空行等，否则数据就会读错，我在调试代码过程中因为没有注意到这一点，就一直得不到最后的结果。want\_data.append方法用于将属性补进列表里，属性+2,一直读到属性最后。这部分的代码就是将数据读取到列表里，再从列表中找出我们需要的最大价值bestvalue。

**数据输入：**

如我们输入第一组数据

6 10 3800（6为背包个数，10为属性个数，3800为最大价值）

100 600 1200 2400 500 2000（cost[i]，价值）

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(蓝色部分为属性）

80 96 20 36 44 48 10 18 22 24（约束条件）

**结果输出：**

输出计算各个交集所需要的时间及程序运行的时间及最大价值等，如我们以第一组数据为例，输出结果为：

商品的个数为： 6 属性的个数为： 10

价值分别为： [100.0, 600.0, 1200.0, 2400.0, 500.0, 2000.0]

选择第 2 个商品 | 选择第 3 个商品 | 选择第 6 个商品 |

最大价值为 3800.0 元

程序运行时间为： 0.18969416618347168 s

可以看到程序运行所花费的时间较短，整个代码的可行性较高。因为无法确定可能性，所以无法计算出整个代码的复杂度。