### 多维背包问题

**算法分析：**

多维背包问题也称为多约束背包问题或者多背包问题，它是带有一组约束条件，比如重量、数量、价值等条件的背包问题。这个问题可以简单描述为n个物品要放入m个承重不同的背包，它与0-1背包问题不同的是，物品放入不同背包的重量是不同的。在多维背包的问题中，除了确定每个物品是否被放入背包之外，还需要确定它需要放入哪个背包，其次，在约束条件下，使其载重满足要求，且价值达到最优。如下图所示：

三个数字分别表示背包个数、属性及物品价值

6 10 3800

80

96

20

36

44

48

10

18

22

24

8 12 13 64 22 41

8 12 13 75 22 4

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

对应10种属性的最大约束值

**整体思路：**

整体考虑，用动态规划算法求解所有可能的背包装载方案，再进行整数规划，最后计算最大价值。

整体规划过程中，可能会存在长时间获取不到最优解得可能性，解决方法如下：

1. 按照实际的结果增加或减少方案集合的数量
2. 使用贪心算法来减少方案集合的数量。
3. 动态规划：

求解所有可能的背包方案

1. 整数规划：

方案：C1，C2，C3，...，Cn

每个方案采用的数量：X1，X2，X3，...，Xn

每个方案物品的装入情况：A1i，A2i，...，Ani

每种物品需求数量：Bi

**代码实现：**

**from** copy **import** deepcopy  
**import** numpy **as** np  
**import** pulp **as** lp  
  
**def** bag\_program(weights, nums, max\_weight):  
 weight\_num = [[]] \* max\_weight  
  
 **for** il, l **in** enumerate(weights):  
 **for** i **in** range(max\_weight):  
 **if** weight\_num[i]:  
 **if** i + l < max\_weight:  
 vals = deepcopy(weight\_num[i])  
 vals\_ = []  
 **for** val **in** vals:  
 val[il] += 1  
 **if** val[il] <= nums[il]:  
 vals\_.append(val)  
 tmp = deepcopy(weight\_num[i + l])  
 tmp.extend(vals\_)  
 weight\_num[i + l] = tmp  
 val = np.zeros(len(weights))  
 **for** i **in** range(l, max\_weight + 1, l):  
 val[il] += 1  
 **if** val[il] > nums[il]:  
 **break** tmp = deepcopy(weight\_num[i - 1])  
 tmp.append(deepcopy(val))  
 weight\_num[i - 1] = tmp  
 **return** weight\_num[-weights[0]:]  
  
  
**def** integer\_program(num\_arrs, nums):  
  
 num\_arrays\_list = []  
 **for** vals **in** num\_arrs:  
 num\_arrays\_list.extend(vals)  
  
 nums\_mat = np.array(num\_arrays\_list)  
 print(**"\t"**, nums\_mat.shape)  
  
 A = nums\_mat.T  
 b = nums  
  
 prob = lp.LpProblem(**"The GY Problem"**, lp.LpMinimize)  
 x = [lp.LpVariable(**"x\_%06d"** % i, lowBound=0, cat=**"Integer"**)  
 **for** i **in** range(A.shape[1])]  
  
 prob += lp.lpSum(x), **"Total Number"  
 for** i **in** range(len(b)):  
 prob += lp.lpSum([A[i][j] \* x[j] **for** j **in** range(A.shape[1])]) >= b[i], **"lb%04d"** % weights\_array[i]  
 prob.solve()  
 print(**"\tStatus:"**, lp.LpStatus[prob.status])  
  
 res = []  
 **for** v **in** prob.variables():  
 **if** v.varValue:  
 res.append((A[:, int(v.name[2:])], v.varValue))  
 **return** res  
  
*# 初始化变量*weights\_array = np.array(  
 [8,12,13,64,22,41,8,12,13,75,22,41,3,6,4,18,6,4,5,10,8,32,6,12,5,13,8,42,6,20,0,0,0,0,8,0,3,0,4,0,8,0,3,2,4,0,8,4])  
nums\_array = np.array(  
 [80,96,20,36,44,48,10,18,22,24])  
bag\_weight =3800  
  
weight\_num\_arrays = bag\_program(weights\_array, nums\_array, bag\_weight)  
  
bags\_num\_min = np.ceil(sum(weights\_array \* nums\_array) / bag\_weight)  
print(**"每个约束条件的约束值"**, bags\_num\_min)  
  
**for** i **in** range(1, len(weight\_num\_arrays) + 1):  
 print(i)  
 **if not** weight\_num\_arrays[-i]:  
 **continue** result = integer\_program(weight\_num\_arrays[-i:], nums\_array)  
 bags\_num = sum([n[-1] **for** n **in** result])  
 print(**"\t"**, bags\_num)  
 **if** bags\_num == bags\_num\_min:  
 **break**print(**"最优价值"**, sum([n **for** \_, n **in** result]))