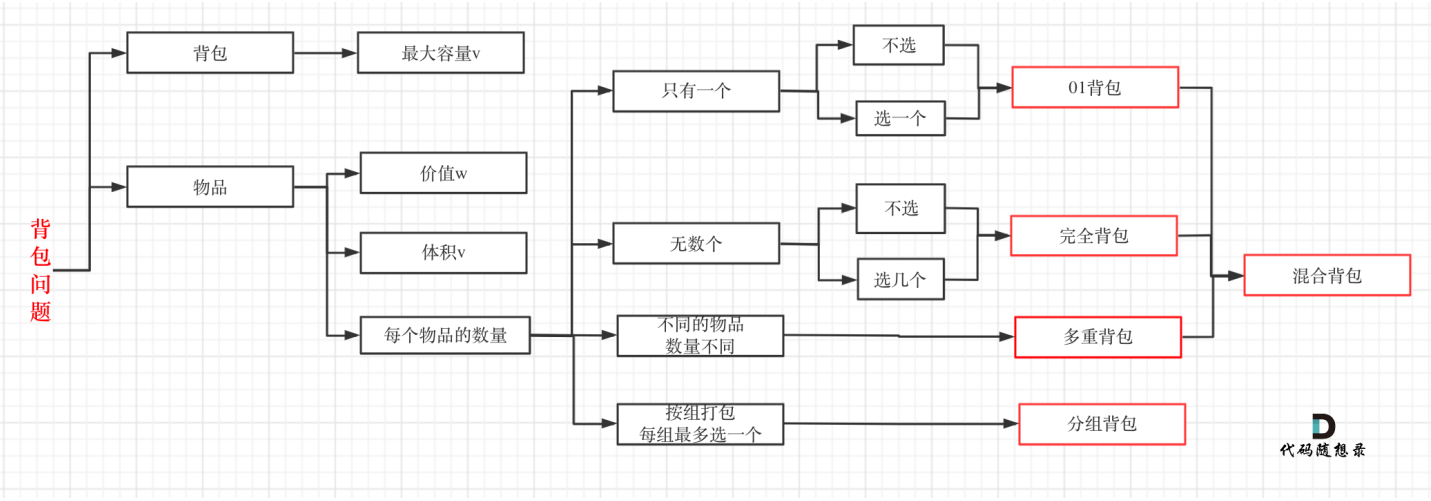
背包问题

分类及解法：



背包问题大体的解题模板是两层循环，分别遍历物品nums和背包容量target，然后写转移方程，
根据背包的分类我们确定物品和容量遍历的先后顺序，根据问题的分类我们确定状态转移方程的写法

首先是背包分类的模板：
1、0/1背包：外循环nums,内循环target,target倒序且target>=nums[i];
2、完全背包：外循环nums,内循环target,target正序且target>=nums[i];
3、组合背包：外循环target,内循环nums,target正序且target>=nums[i];

- **如果求组合数就是外层for循环遍历物品，内层for遍历背包。**
- **如果求排列数就是外层for遍历背包，内层for循环遍历物品。**

4、分组背包：这个比较特殊，需要三重循环：外循环背包bags,内部两层循环根据题目的要求转化为1,2,3三种背包类型的模板

然后是问题分类的模板：
1、最值问题: dp[j] = max/min(dp[j], dp[j-nums[i]]+1)或dp[j] = max/min(dp[j], dp[j-num[i]]+nums[i]);
2、存在问题(bool)：dp[j]=dp[j]||dp[j-num[i]];
3、不考虑顺序组合问题（有几种方法）：dp[j]+=dp[j-num[i]];

Q：有n件物品和一个最多能背重量为w 的背包。第i件物品的重量是weight[i]，得到的价值是value[i] 。**每件物品只能用一次**，求解将哪些物品装入背包里物品价值总和最大。

A：

二维数组解法

dp的定义：dp[i][j] 表示从下标为[0-i]的物品里任意取，放进容量为j的背包，价值总和最大是多少

递推公式：`dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - weight[i]] + value[i])`

打表：具体做法是：画一个 len 行，target + 1 列的表格。这里 len 是物品的个数，target 是背包的容量。len 行表示一个一个物品考虑，target + 1多出来的那 1 列，表示背包容量从 0 开始考虑。很多时候，我们需要考虑这个容量为 0 的数值。

初始化：

- dp[0][j]，即：i为0，存放编号0的物品的时候，各个容量的背包所能存放的最大价值。
- dp[i][0],即背包重量为0，物品价值当然初始化为0.

```
rows, cols = len(weight), bag_size + 1
dp = [[0 for _ in range(cols)] for _ in range(rows)]

# 初始化dp数组.
for i in range(rows):
  dp[i][0] = 0
first_item_weight, first_item_value = weight[0], value[0]
for j in range(1, cols):
  if first_item_weight <= j:
    dp[0][j] = first_item_value
```

遍历顺序：先遍历物品还是重量?

主要代码：

```
# 更新dp数组: 先遍历物品, 再遍历背包.
for i in range(1, len(weight)):
  cur_weight, cur_val = weight[i], value[i]
  for j in range(1, cols):
    if cur_weight > j: # 说明背包装不下当前物品.
      dp[i][j] = dp[i - 1][j] # 所以不装当前物品.
    else:
    # 定义dp数组: dp[i][j] 前i个物品里，放进容量为j的背包，价值总和最大是多少。
      dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - cur_weight]+ cur_val)
```

一维数组解法（滚动数组）

条件：上一层可以重复利用，直接拷贝到当前层。但是还是要用i和j遍历。

dp[j]表示：容量为j的背包，所背的物品价值可以最大为dp[j]。

递推公式：`dp[j] = max(dp[j], dp[j - weight[i]] + value[i])`

初始化：全为0即可,bag_weight是背包的容量 `dp = [0] * (bag_weight + 1)`

遍历顺序！：二维dp遍历的时候，背包容量是从小到大，而一维dp遍历的时候，背包是从大到小。倒序遍历的目的是为了只让物品i被放入1次。倒序遍历的原因是，本质上还是一个对二维数组的遍历，并且右下角的值依赖上一层左上角的值，因此需要保证左边的值仍然是上一层的，从右向左覆盖。

主要代码：

```
# 先遍历物品，再遍历背包容量
 for i in range(len(weight)):
     for j in range(bag_weight, weight[i] - 1, -1):
        #bag_weight >= weight[i] -> bag_weight > weight[i]-1
        # 递归公式
         dp[j] = max(dp[j], dp[j - weight[i]] + value[i])
```