关于多重背包，你该了解这些

对于多重背包，我做一个简单介绍

> 有**N种物品**和一个**容量为V**的背包。**第i种物品最多有Mi件可以用，每件耗费的空间是Ci,价值是Wi**，求解将哪些物品装入背包可使这些物品的耗费的空间总和不超过背包容量，且价值总和最大。

多重背包和01背包是非常像的，为什么和01背包像呢?

每件物品最多有**Mi件可用**，把**Mi件摊开**，其实就是一个**01背包问题**了。

做法：把多重背包进行展开，转换为01背包问题。

如何展开呢?

把**多余的物品**展开加入到**weight**和**value**数组里面,转换成weight和value相同的另一种物品

```
Coding:
void test_multi_pack(){
 vector<int> weight={1,3,4};
 vector<int> value={15,20,30};
 vector<int> nums={2,3,2};
 for(int i=0;i<nums.size();i++){//表示多余的物品
 while(nums[i]>1){
 weight.push_back(weight[i]);
 value.push_back(value[i]);
  nums[i]--;
  }
  }
  for(int i=0;i<weight.size();i++){
  for(int j=bagSize;j>=weight[i];j--){
  dp[j]=max(dp[j],dp[j-weight[i]]+value[i]);
  }
  }
  return dp[bagSize];
}

另一种每种商品遍历的个数放在01背包里面在遍历一遍;
for(int i=0;i<weight.size();i++){//先遍历物品
  for(int j=bagSize;j>=weight[i];j--){
  for(int k=1;k<=nums[i];k++){
  if((j-k*weight[i])>=0){
  dp[j]=max(dp[j],dp[j-k*weight[i]]+k*value[i]);
  }
  }
  }
```

```
30  }
31  return dp[bagSize];
```

01背包

在动态规划：关于01背包问题，你该了解这些！中我们讲解二维dp数组01背包先遍历物品还是先遍历背包都是可以的，且第二层for循环是从小到大遍历。

和动态规划：关于01背包问题，你该了解这些！（滚动数组）中，我们讲解一维dp数组01背包只能先遍历物品再遍历背包容量，且第二层for循环是从大到小遍历。

一维dp数组的背包在遍历顺序上和二维dp数组实现的01背包其实是有很大差异的，大家需要注意！

背包递推公式：

背包递推公式

问能否能装满背包（或者最多装多少）：dp[j] = max(dp[j], dp[j - nums[i]] + nums[i]);，对应题目如下：

- 动态规划：416.分割等和子集
- 动态规划：1049.最后一块石头的重量 II

问装满背包有几种方法：dp[j] += dp[j - nums[i]]，对应题目如下：

- 动态规划：494.目标和
- 动态规划：518. 零钱兑换 II
- 动态规划：377.组合总和IV
- 动态规划：70. 爬楼梯进阶版（完全背包）

问背包装满最大价值：dp[j] = max(dp[j], dp[j - weight[i]] + value[i]);，对应题目如下：

- 动态规划：474.一和零

问装满背包所有物品的最小个数：dp[j] = min(dp[j - coins[i]] + 1, dp[j]);，对应题目如下：

总结篇：https://mp.weixin.qq.com/s/ZOehl3U1mDiyOQjFG1wNJA