0-1背包问题：时间复杂度O(n*v) 空间复杂度O(V)

有一个容量为V的背包，一个数组N，有两个属性，体积为volumn，价值为Value，每种物品只有一个，要求使用这个背包装下价值尽可能多的物品，背包可以不装满；由于每个物品只有两种状态，在背包中，不在背包中。

由于每次把物品放入包内需要判断是否是最优的价值，所以需要进行比较；子问题中的物品数量和剩下的背包容量都应该作为变量。子问题：当背包大小为i时，其中前j个物品能达到的最大价值。

确定状态转移矩阵：

当前遍历到了第j个物品，求出在背包大小为i的时候，背包的最大价值

如果背包容量小于value[j]

dp[i][j]=dp[i][j-1]

否则

dp[i][j]= dp[i-volumn[j]] [j-1] +value[j]意思是为j号元素预留位置的情况下的最大收益+value[j]

或者dp[i][j]=dp[i][j-1]

为保证每个物品只能使用一次，我们倒序遍历所有i的值，

这样就dp[i-volumn[j]] [j-1]就会落在i的左边，这样就不会用到重复的值；

观察状态方程其实dp[i][j] 只与前一行dp[i][j-1]有关，所以可以修改为一维的DP方程，注意先遍历i后遍历j

完全背包问题：

在这个问题中，每个物品的数量都是不限制的

我们从左到右序遍历所有 i 的值，

这样就dp[i-volumn[j]] [j-1]就会落在i的左边，可能用到重复的值；

多重背包问题：

每一个物品的数量由一个数组nums[] 提供

直接的方法就是把多个重复的物品看做完全不同的物品，套用0-1背包的解法复杂度为O(V*∑nums[j])

有一种巧妙的方法是：对于每个物品的数量K，把K拆分为1，2，4，8...这样的小块，每个小块求和组成一个新的物品。这样物品的数量能够缩减为原来的Log(K)个物品；

算法复杂度为O(V*∑log(nums[j]))