动态规划之三种背包

演讲：陶柯蓉

目录

- 一. 动归铺垫：
  贪心？动归问题特点？
- 二. 三种背包
  0-1背包（一维滚动优化）
  完全背包（一维滚动优化）
  多重背包 （朴素解法，一维滚动优化，二进制解法，单调栈优化）
- 三. 总结

01背包基础

有 N 件物品和一个容量是 V的背包。每件物品只能使用一次。第 i件物品的体积是 vi，价值是 wi。
求解将哪些物品装入背包，可使这些物品的总体积不超过背包容量，且总价值最大。输出最大价值。

输入格式:
第一行两个整数，N，V，用空格隔开，分别表示物品数量和背包容积。
接下来有 N行，每行两个整数 vi,wi，用空格隔开，分别表示第 i件物品的体积和价值。

输出格式:
输出一个整数，表示最大价值。
数据范围0<N,V≤1000 0<vi,wi≤1000
输入样例
4 4
1 2
2 4
3 4
4 5
输出样例:
6

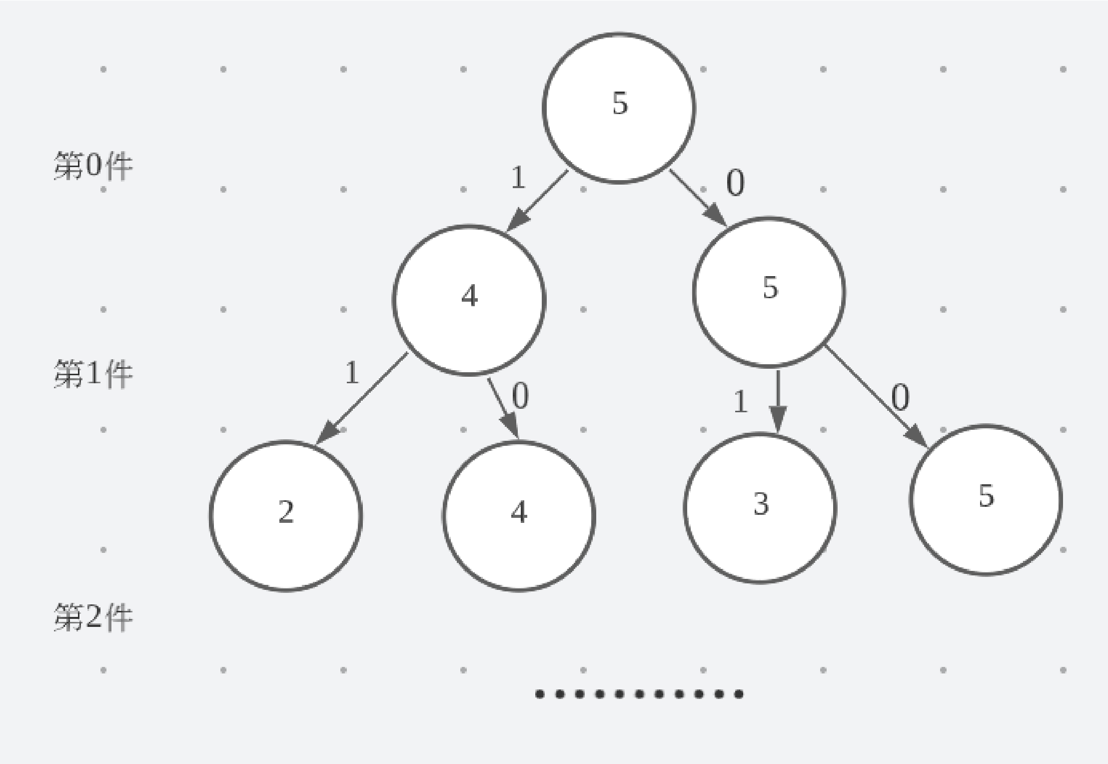
明确几个概念，i,weight[i],value[i],C

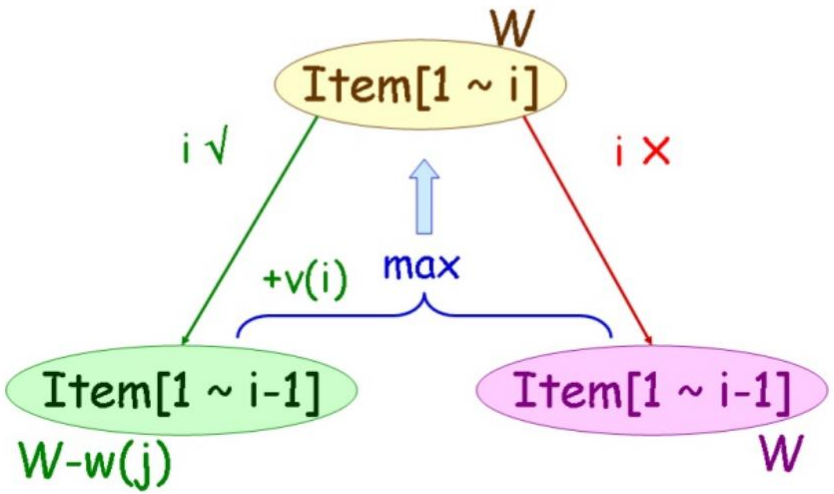
	v	w
第0件	1	2
第1件	2	4
第2件	3	4
第3件	4	5

01背包基础

暴力解法： 穷举每种可能，不符合条件的话就回溯，画出选择树

状态转移方程：F（i，C）= max(F（i-1，C），v（i）+ F（i-1，C-w（i）））





01背包基础

	v	w
第0件	1	2
第1件	2	4
第2件	3	4
第3件	4	5

动归：自顶向下调用，自底向上求解。
二维表就是省略调用步骤，直接模拟求解时的过程。

物品 \ 背包容量	0	1	2	3	4
第0件	0	2	2	2	2
第1件	0	2	4	6	6
第2件	0	2	4	6	6
第3件	0	2	4	6	6

状态转移方程：F（i，C）= max(F（i-1，C），v（i）+F（i-1，C-w（i）））

01背包基础

	v	w
第0件	1	2
第1件	2	4
第2件	3	4
第3件	4	5

空间优化——一维数组降维打击
注意遍历顺序

背包容量 / 物品	0	1	2	3	4
第0件	0	2	2	2	2
第1件	0	2	4	6	6
第2件	0	2	4	6	6
第3件	0	2	4	6	6

状态转移方程：F（i，C）= max(F（i-1，C），v（i）+ F（i-1，C-w（i）））

完全背包基础

有 N 种物品和一个容量是 V的背包，每种物品都有**无限件可用**。第 i种物品的体积是 vi，价值是 wi。
求解将哪些物品装入背包，可使这些物品的总体积不超过背包容量，且总价值最大。输出最大价值。
输入格式
第一行两个整数，N，V，用空格隔开，分别表示物品种数和背包容积。
接下来有 N行，每行两个整数 wi,vi，用空格隔开，分别表示第 i种物品的体积和价值。
输出格式
输出一个整数，表示最大价值。
数据范围0<N,V≤1000 0<vi,wi≤1000
输入样例
4 4
1 2
2 4
3 4
4 5
输出样例：
8

	v	w
第0件	1	2
第1件	2	4
第2件	3	4
第3件	4	5

完全背包基础

与01的区别：
dp[i][C]表示只使用**前i个物品（每个物品可用多次）**，装容量为C的背包，获得的最大价值。

$$dp[i][j] = max(dp[i-1][j], dp[i][j-v[i]] + w[i])$$

	v	w
第0件	1	2
第1件	2	4
第2件	3	4
第3件	4	5

背包容量 / 物品	0	1	2	3	4
前1件	0	2	4	6	8
前2件	0	2	4	6	8
前2件	0	2	4	6	8
前3件	0	2	4	6	8

多重背包

假设背包容量700，每个物品（依次编号为物品i）数量有限，有num(i)个，物品表如右图所示：

状态转移：$0 < k$ && $k * w[i] <= j$ && **$k <= num[i]$**

$dp[i][j] = \max(dp[\mathbf{i-1}][j-k*w[i]] + k*v[i], dp[i-1][j])$

	v	w	num
物品1	1	2	3
物品2	2	4	13
物品3	3	4	200
物品4	4	5	1

V为物品数总和

C

N

	0	1	2	3	4	5	6	7
前1件								
前2件								
前3件								
前4件								

…… 到700

假设背包容量700，每个物品（依次编号为物品i）数量有限，有num(i)个，物品表如右图所示：

	v	w	num
物品1	1	2	3
物品2	2	4	13
物品3	3	4	200
物品4	4	5	1

朴素解法：

0 < k && k * w[i] <= j && k <= num[i]

dp[i][j] = max(dp[i-1][j-k*w[i]] + k*v[i], dp[i-1][j])

```
for i in range(1, row):
    for j in range(1, col):
        for k in range(1, num + 1):
            if j >= k * w:
                input_val = dp[i - 1][j - k * w] + k * v
                noput_val = dp[i - 1][j]
                dp[i][j] = max(input_val, noput_val)
            else:
                dp[i][j] = dp[i - 1][j]
                break
```

一维数组优化： dp[j] = max(dp[j-k*w[i]] + k*v[i], dp[j])

```
for i in range(1, row):
    for j in range(col, w, -1):
        for k in range(1, num + 1):
            if j >= k * w:
                dp[j] = max(dp[j - k * w] + k * v, dp[j])
```

时间复杂度：O(C*N*V)

多重背包

	v	w	num
物品1	1	2	3
物品2	2	4	13
物品3	3	4	200
物品4	4	5	1

每个物品不止1个，有num(i)个，物品表如右图：

转化成01背包解：

	0	1	2	3	4	……
第0件（物品1）						
第1件（物品1）						
第2件（物品1）						
第3件（物品2）						
第4件（物品2）						
第5件（物品2）						
第6件（物品2）						

……

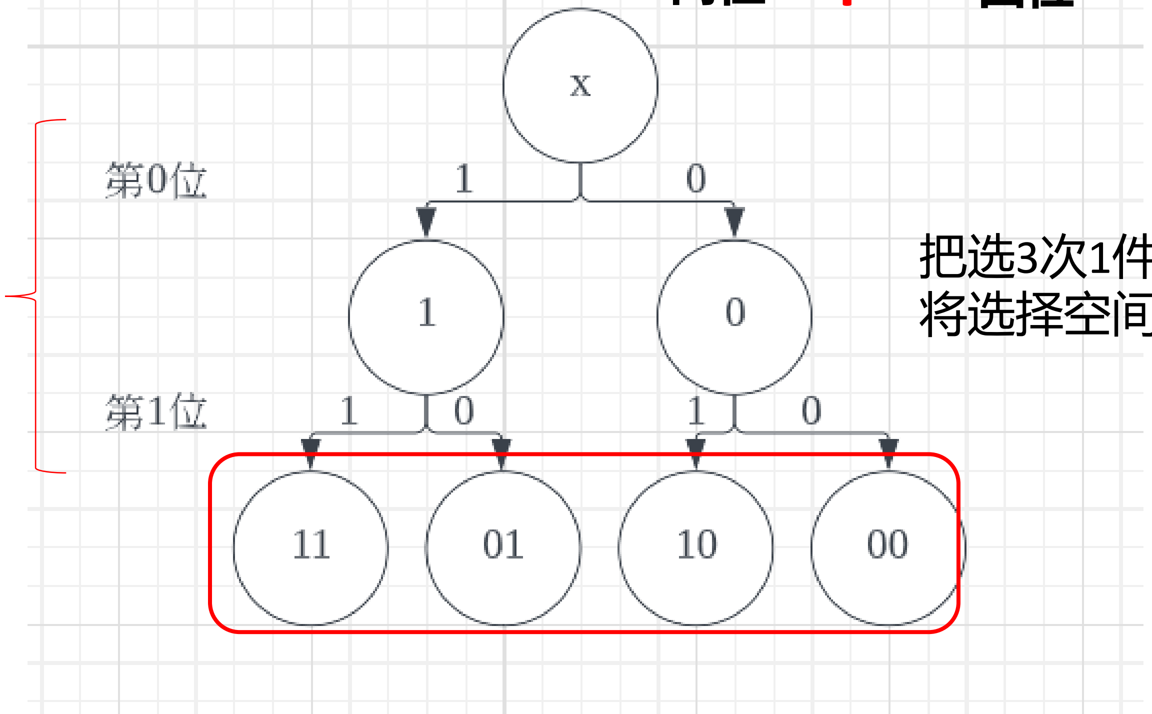
多重背包

二进制解法：　二叉分类树 <=> 二进制表示

*3　　*13　　*200

两位 **+** **四位** **+** **八位** **=十四位**

只需**两位**二进制数字



把选3次1件物品，变成选2次（第一次选1件，第二件选2次）
将选择空间从扁平变折叠

	v	w	num
物品1	1*1	2	1
物品2	2*1	8	1
物品3	1*2	4	1
物品4	2*2	8	1

二进制解法：　二叉分类树 <=> 二进制表示

*3　　*13　　*200

针对物品的压缩：　**两位** + **四位** + **八位** =**十四位**

	0	1	2	3	4	5	6	7
前1件								
前2件								
前3件								
...								
前6件								
前7件								
...								
前14件								

…… 到700

时间复杂度：O(C*N*log(V))

十四件物品的01背包

单调队列：

a:

1	3	-1	-3	5	3	6	7
0	1	2	3	4	5	6	7

操作：尾端加入，头尾两端删除。
特性：保持队列**单调性**，头部的值一直为区间内最大/小值；滑动窗口。

q:

1	3						
0	1	2	3	4	5	6	7

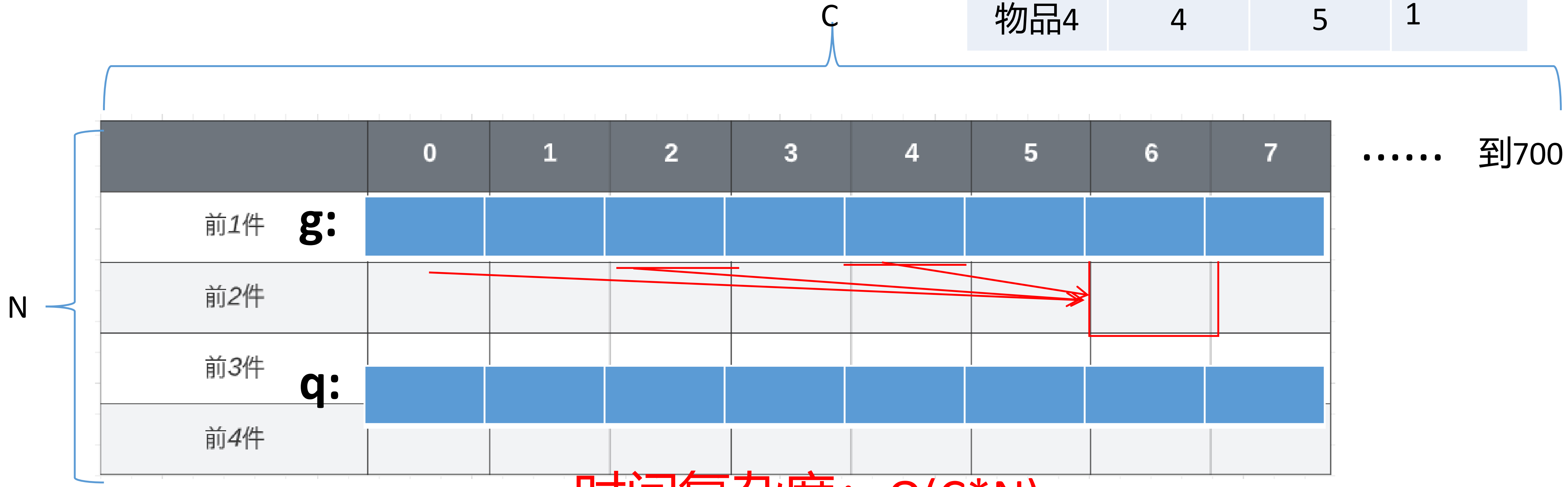
代码：　　　　用双指针维护单调队列，hh负责头，tt负责尾

```
n=8  k=3 hh=0  tt=-1
for(int i=0;i<n;i++){
    while(hh<=tt & i-k>=q[hh])hh++;//用hh删除头出界的（q数组之所以记录下标）
    while(hh<=tt & a[q[tt]]<=a[i])tt--;//用tt删除尾违反单调的
    q[++tt]=i;
    if(i>=k-1)cout<<a[q[hh]]<<’ ’;
}
```

单调队列优化dp:

针对状态的压缩:

	v	w	num
物品1	1	2	3
物品2	2	4	13
物品3	3	4	200
物品4	4	5	1



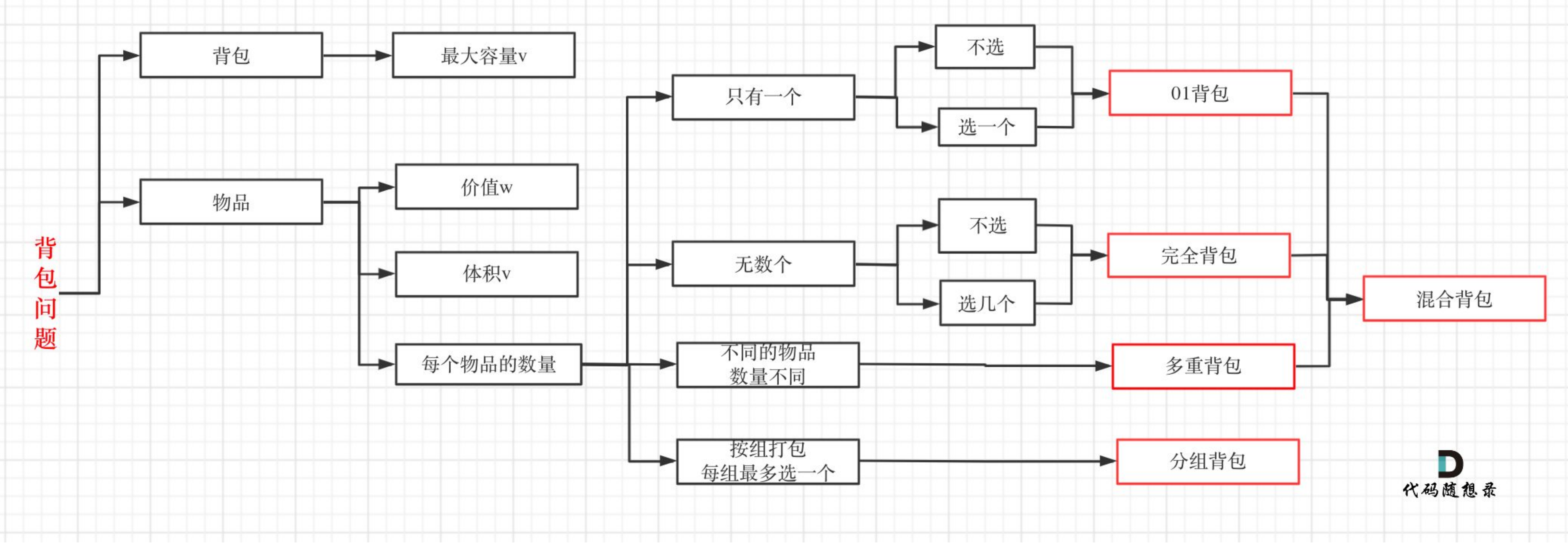
时间复杂度：O(C*N)

用单调队列动态维护，将内层状态选择O(C)优化为O(1)

```
class Solution {
    public int maxValue(int N, int C, int[] s, int[] v, int[] w) {
        int[] dp = new int[C + 1];
        int[] g = new int[C + 1]; // 辅助队列，记录的是上一次的结果
        int[] q = new int[C + 1]; // 主队列，记录的是本次的结果

        // 枚举物品
        for (int i = 0; i < N; i++) {
            int vi = v[i];
            int wi = w[i];
            int si = s[i];
            // 将上次算的结果存入辅助数组中
 g = dp.clone();
            // 枚举余数
            for (int j = 0; j < vi; j++) {
                // 初始化队列，head 和 tail 分别指向队列头部和尾部
                int head = 0, tail = -1;
                // 枚举同一余数情况下，有多少种方案。
                // 例如余数为 1 的情况下有：1、vi + 1、2 * vi + 1、3 *
vi + 1 ...
                for (int k = j; k <= C; k+=vi) {
 dp[k] = g[k];
                    // 将不在窗口范围内的值弹出
                    if (head <= tail && q[head] < k - si * vi)
head++;
                    // 如果队列中存在元素，直接使用队头
来更新
                    if (head <= tail) dp[k] = Math.max(dp[k],
g[q[head]] + (k - q[head]) / vi * wi);
                    // 当前值比对尾值更优，队尾元素没有
存在必要，队尾出队
                    while (head <= tail && g[q[tail]] - (q[tail] - j)
/ vi * wi <= g[k] - (k - j) / vi * wi) tail--;
                    // 将新下标入队
                    q[++tail] = k;
                }
            }
        }
        return dp[C];
    }
}
```

最后，总结几种背包的分类和需要注意的点



谢谢大家，终于结束了hiahia！