背包问题是一个NP-complete的组合优化问题，Search的方法需要O(2^N)时间才能获得最优解。而使用动态规划，我们可以在**伪多项式（pseudo-polynomial time）**时间内获得最优解。

**0-1 Knapsack Problem 0-1背包问题**

Problem

Given N items, w[i] is the weight of the i-th item and v[i] is value of the i-th item. Given a knapsack with capacity W. Maximize the total value. Each item can be use **0 or 1 time**.

0-1背包问题的通常定义是：一共有N件物品，第i件物品的重量为w[i]，价值为v[i]。在总重量不超过背包承载上限W的情况下，能够获得的最大价值是多少？每件物品可以使用**0次或者1次**。

例子：

重量 w = [1, 1, 2, 2]

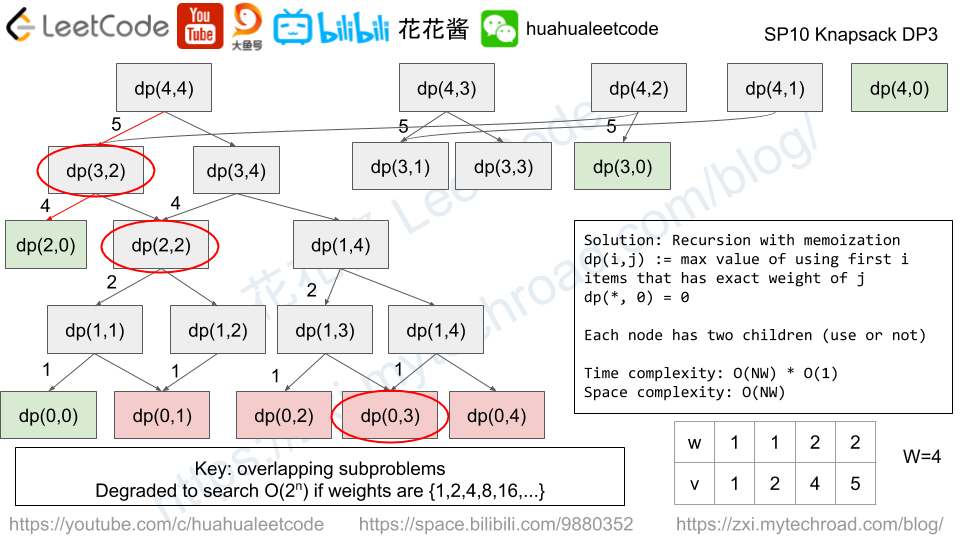
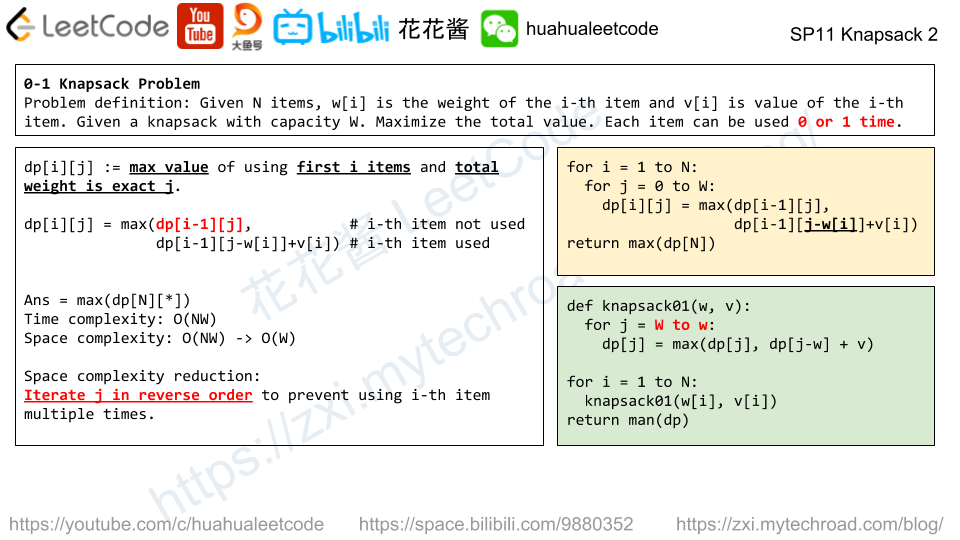
价值 v = [1, 3, 4, 5]

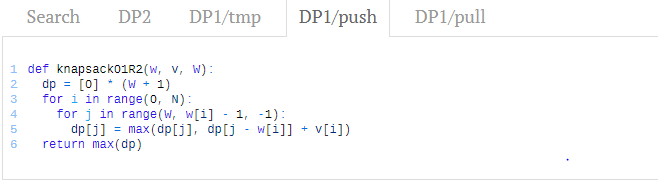
背包承重 W = 4

最大价值为9，可以选第1,2,4件物品，也可以选第3，4件物品；总重量为4，总价值为9。

动态规划的状态转移方程为：

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



**完全背包**、**多重背包**是常见的变形。和01背包的区别在于，完全背包每件物品可以使用**无限多次**，而多重背包每件物品最多可以使用**n[i]次**。两个问题都可以转换成01背包问题进行求解。

但是Naive的转换会大大增加时间复杂度：

完全背包：“复制”第i件物品到一共有 W/w[i] 件

多重背包：“复制”第i件物品到一共有 n[i] 件

这就涉及到二进制思想：任何一个正整数都可以用 (1, 2, 4, …, 2^K)的组合来表示。例如14 = 2 + 4 + 8。  
原本需要放入14件相同的物品，现在只需要放入3件（重量和价值是原物品的2倍，4倍，8倍）。大幅降低了总的物品数量从而降低运行时间。

完全背包：对于第i件物品，我们只需要创建k = log(W/w[i])件虚拟物品即可。

每件虚拟物品的重量和价值为：1\*(w[i], v[i]), 2\*(w[i], v[i]), …, 2^k\*(w[i], v[i])。

多重背包：对于第i件物品，我们只需要创建k + 1件虚拟物品即可，其中k = log(n[i])。

每件虚拟物品的重量和价值为：1\*(w[i], v[i]), 2\*(w[i], v[i]), …, 2^(k-1)\*(w[i], v[i]), 以及 (**n[i] – 2^k – 1**) \* (w[i], v[i])。

例如：n[i] = 14, k = 3, 虚拟物品的倍数为 1, 2, 4 和 7，这4个数组合可以组成1 ~ 14中的任何一个数，并且不会>14，即不超过n[i]。

二进制转换后直接调用01背包即可

