11 背包型动规

笔记本： DP Note
创建时间： 10/13/2019 3:15 PM **更新时间：** 11/10/2019 9:42 PM
作者： tanziqi1756@outlook.com

背包问题

九章算法

- 你有一个背包，背包有最大承重
- 商店里有若干物品，都是免费拿
- 每个物品有重量和价值
- 目标：不撑爆背包的前提下
  - 装下最多重量物品
  - 装下最大总价值的物品
  - 有多少种方式正好带走满满一书包物品

最大承重
14公斤

动态规划组成部分二：转移方程

九章算法

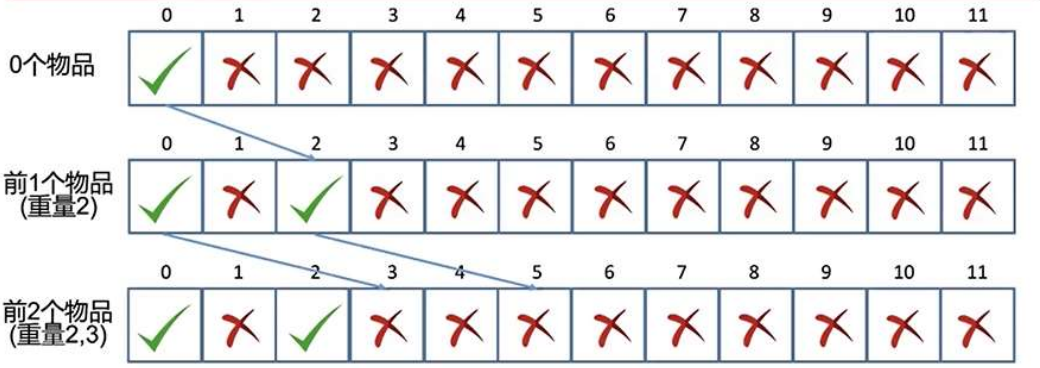
- 设f[i][w] = 能否用前i个物品拼出重量w (TRUE / FALSE)

$$f[i][w] = f[i-1][w] \text{ OR } f[i-1][w-A_{i-1}]$$

- 能否用前i个物品拼出重量w
- 能否用前i-1个物品拼出重量w
- 能否用前i-1个物品拼出重量w-A_{i-1}，再加上第i个物品

例子

九章算法



92. Backpack

中文 English

Given n items with size A_i, an integer m denotes the size of a backpack. How full you can fill this backpack?

Example

```
Example 1:
        Input:  [3,4,8,5], backpack size=10
        Output:  9

Example 2:
        Input:  [2,3,5,7], backpack size=12
        Output:  12
```

Challenge

O(n x m) time and O(m) memory.

O(n x m) memory is also acceptable if you do not know how to optimize memory.

Notice

You can not divide any item into small pieces.

Input test data (one parameter per line)

How to understand a testcase?

```
10
[3,4,8,5]
```

这个背包你能装多满?

本来想抖机灵试试用一维数组，但是如果出现重量相同的items就不好处理了。

```java
public class Solution {
    /**
     * @param m: An integer m denotes the size of a backpack
     * @param A: Given n items with size A[i]
     * @return: The maximum size
     */
    public int backPack(int m, int[] A) {
        // write your code here
        int N = A.length;
        boolean[][] dp = new boolean[m+1][N+1];
        // initialize
        dp[0][0] = true;

        // state: f[w]
        // means whether we can use items to
        // exactly fill a bag with weight w
        int ans = 0;
        for( int i = 0; i < N; i++ ) {
            int weight = A[i];
            int j = i + 1;  // 前j个物品
            for( int w = 0; w < m + 1; w++ ) {
                if( dp[w][j-1] ) {
                    dp[w][j] = true;
                    if(  w + weight < m + 1 ) {
                        dp[w+weight][j] = true;
                        ans = (w+weight > ans) ? (w+weight) : (ans);
                    }
                }
            }
        }

        return ans;
    }
}
```

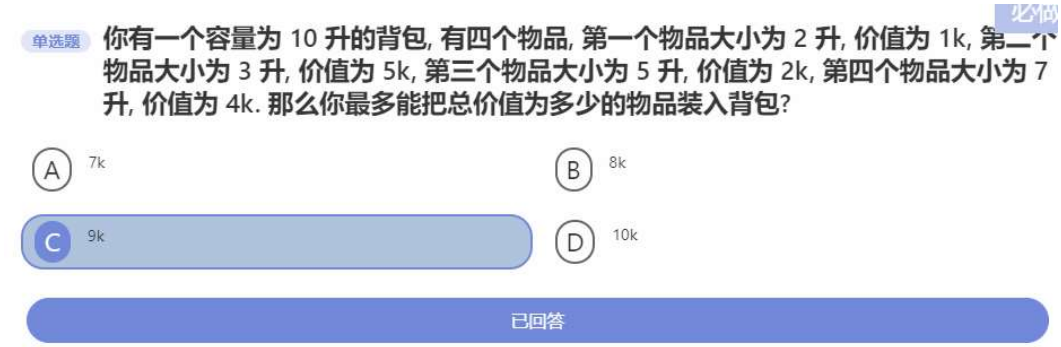
九章算法：

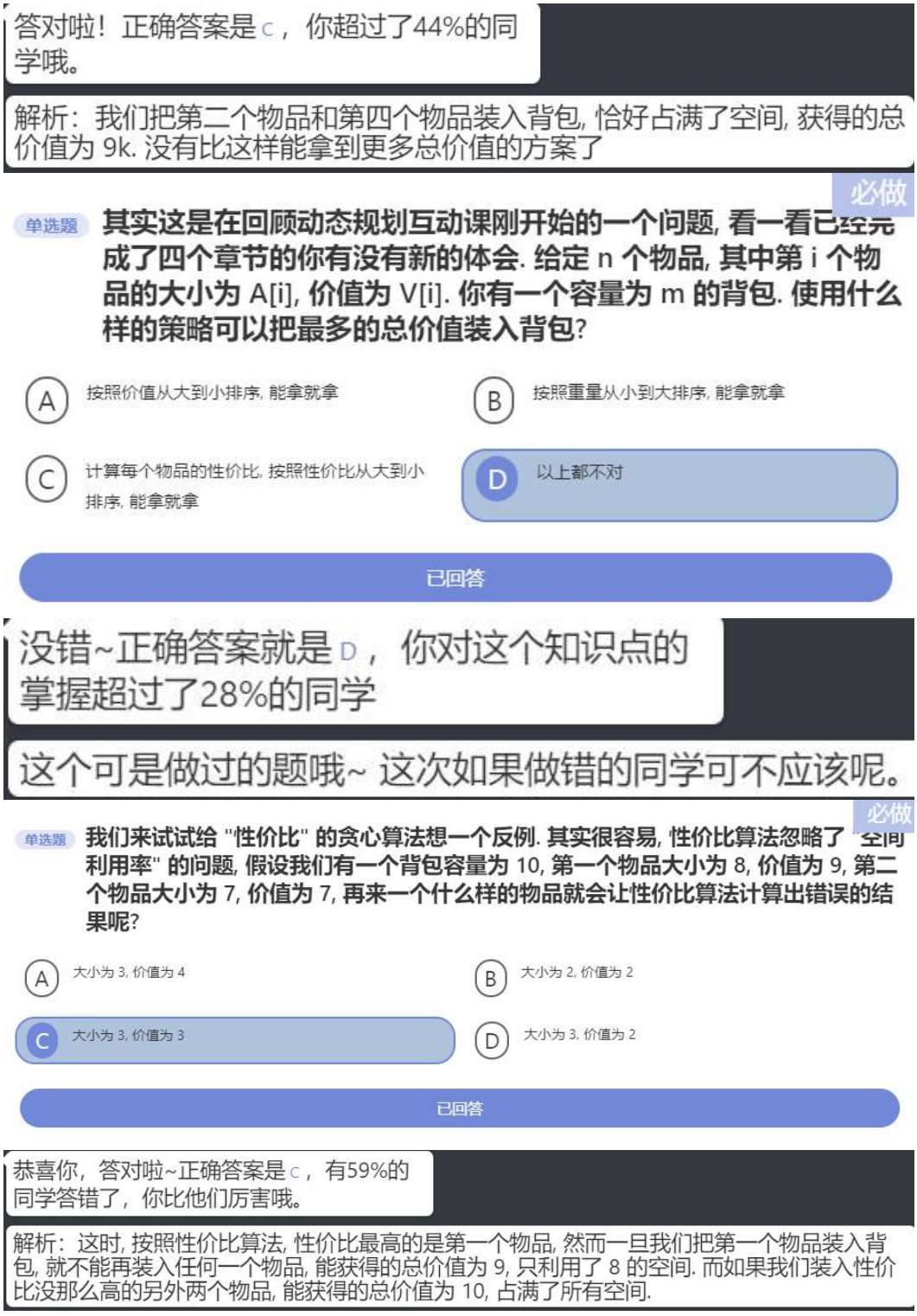
```java
boolean[][] f = new boolean[n + 1][m + 1];
int i, w;

// initialization
for (i = 1; i <= m; ++i) {
    f[0][i] = false;
}

f[0][0] = true;
// first i items
for (i = 1; i <= n; ++i) {
    for (w = 0; w <= m; ++w) {
        // case 1: not using item_{i-1}
        f[i][w] = f[i - 1][w];

        // case 2: using item_{i-1}
        if (w >= A[i - 1]) {
            f[i][w] = f[i][w] || f[i - 1][w - A[i - 1]];
        }
    }
}

for (i = m; i >= 0; --i) {
    if (f[n][i]) {
        return i;
    }
}
```

可以用滚动数组

Lintcode背包问题六连问
https://segmentfault.com/a/1190000006325321

92
125
440
563
564 Combination sum IV
800
Leetcode 518 coins change 2 背包变形





背包问题第二问

Lintcode 125

如果每个物品有价值，如何装到总价值最大的物品。

125. Backpack II

中文 English

There are `n` items and a backpack with size `m`. Given array `A` representing the size of each item and array `V` representing the value of each item.

What's the maximum value can you put into the backpack?

Example

Example 1:

```
Input: m = 10, A = [2, 3, 5, 7], V = [1, 5, 2, 4]
Output: 9
Explanation: Put A[1] and A[3] into backpack, getting the maximum value V[1] + V[3]
= 9
```

Example 2:

```
Input: m = 10, A = [2, 3, 8], V = [2, 5, 8]
Output: 10
Explanation: Put A[0] and A[2] into backpack, getting the maximum value V[0] + V[2]
= 10
```

Challenge

O(nm) memory is acceptable, can you do it in O(m) memory?

Notice

1. `A[i], V[i], n, m` are all integers.
2. You can not split an item.
3. The sum size of the items you want to put into backpack can not exceed `m`.
4. Each item can only be picked up once

状态表达：
dp[w][j]表示前j个物品能恰好拼凑出重量w，此时的最大价值为dp[w][j].
转移方程：
dp[w+weight][j] = max(dp[w][j-1]+V[j-1], dp[w+weight][j-1])
最好先把状态转移方程写出来再写代码，
不然bug找不出来，写的时候逻辑混乱。

```java
/*
Author: Ziqi Tan
*/
public class BackpackII {

    public static int backPack2(int m, int[] A, int[] V) {
        // write your code here

        int N = A.length;
        int[][] dp = new int[m+1][N+1];

        // stage: f[w][j]
        // means the total value when we can use the first j items to
        // exactly fill a bag with weight w

        // initialize
        // -1 means you cannot exactly fill a bag with weight 2 with the first j items
        for( int i = 0; i < m + 1; i++ ) {
            for( int j = 0; j < N + 1; j++ ) {
                dp[i][j] = -1;
            }
        }
        dp[0][0] = 0;
       // Stage transfer:
       for( int i = 0; i < N; i++ ) {
           int weight = A[i];
           int j = i + 1;  // 前j个物品
           for( int w = 0; w < m + 1; w++ ) {
               if( dp[w][j-1] != -1 ) {
                   // 如果当前值dp[w][j]已经之前dp[w+weight][j]被更新过，
                   // 那就不需要再次覆盖了
                   if( dp[w][j-1] > dp[w][j] ) {
                       dp[w][j] = dp[w][j-1];
                   }
                   //看看叠加后是否装得下
                   if( w + weight < m + 1 && dp[w][j-1] + V[j-1] > dp[w+weight][j-1]) {
                       // 如果叠加后的价值是已经存在的
                       // 看看当前叠加的价值是否大于之前叠加的价值
                       dp[w+weight][j] = dp[w][j-1] + V[j-1];
                       // dp[w][j-1]不能用dp[w][j]因为每个物品只能用一次
                   }
               }
           }
       }

       int ans = 0;
       for( int w = 0; w < m + 1; w++ ) {
           ans = Math.max(ans, dp[w][N]);
       }

       return ans;
    }

    public static void main(String[] args) {

        int[] A = new int[] {95,75,23,73,50};
        int[] V = new int[] {89,59,19,43,100};
        int size = 300;
        backPack2(size, A, V);
    }
}
```

九章算法

```
int[][] f = new int[n + 1][m + 1];
int i, w;
for (i = 1; i <= m; ++i) {
    f[0][i] = -1; // cannot make >0 weight with 0 items
}

f[0][0] = 0;
for (i = 1; i <= n; ++i) {
    for (w = 0; w <= m; ++w) {
        f[i][w] = f[i - 1][w]; // not using item i-1
        if (w >= A[i - 1] && f[i - 1][w - A[i - 1]] != -1) {
            // using item i - 1
            f[i][w] = Math.max(f[i][w], f[i - 1][w - A[i - 1]] + V[i - 1]);
        }
    }
}

int res = 0;
for (w = 0; w <= m; ++w) {
    if (f[n][w] != -1) {
        res = Math.max(res, f[n][w]);
    }
}
```

必做

单选题 **刚刚讲过的 Backpack II 中, 最后的答案可不可以直接返回 f[n][m], 不在 f[n][i] 中求最大值?**

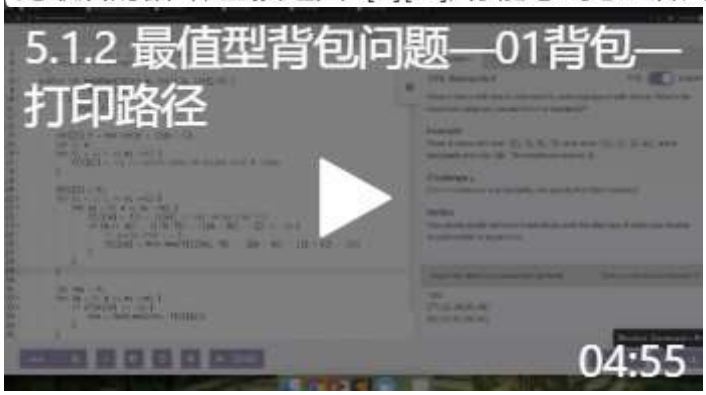
A 可以

B 不可以

已回答

答对了！看来掌握得不错哦。正确答案是 B ，你击败了25%的学员，继续努力~

解析：我们设定的状态是 f[i][j] 表示前 i 个物品拼出容量 j 时, 得到的最大总价值. 容量 j 未必能拼得出来. 不过, 我们可以做一点点小小的变动, 就可以避免最后的循环, 直接返回 f[n][m], 你能想到怎么做吗?



必做

单选题 **一个扩展问题: 多重背包. 这时每个物品不是无限个啦, 而是每个物品都有一定的数量, 或多或少. 那么这个问题我们可以用上面的哪个方案解决呢?**

A Backpack II 的算法

B Backpack III 的优化算法

已回答

是有点难吗，答错了呢。正确答案是 A ，有38%的同学超过了你，但是千万不要气馁。

解析：既然每个物品有一定的数量，那么我们完全可以把它们看成不同的物品，然后按照 Backpack II 的算法来解决. 只不过速度可能更慢一点而已. 但是 Backpack III 的优化算法是固然不可行的，因为物品数量有限，不能无限地取用.

背包第三问：
完全背包，如果某个物品可以取无限多次

440. Backpack III

中文 English

Given `n` kinds of items, and each kind of item has an infinite number available. The `i-th` item has size `A[i]` and value `V[i]`.

Also given a backpack with size `m`. What is the maximum value you can put into the backpack?

Example

Example 1:

```
Input: A = [2, 3, 5, 7], V = [1, 5, 2, 4], m = 10
Output: 15
Explanation: Put three item 1 (A[1] = 3, V[1] = 5) into backpack.
```

Example 2:

```
Input: A = [1, 2, 3], V = [1, 2, 3], m = 5
Output: 5
Explanation: Strategy is not unique. For example, put five item 0 (A[0] = 1, V[0] = 1) into backpack.
```

Notice

1. You cannot divide item into small pieces.
2. Total size of items you put into backpack can not exceed `m`.

这就更简单了，因为你只要一维数组，来一个物品更新m/k次就可以了。

```java
public int backPackIII(int[] A, int[] V, int m) {
    // write your code here
    int N = A.length;
    int[] dp = new int[m+1];

    // stage: f[w][j]
    // means the total value when we can use the first j kinds of items to
    // exactly fill a bag with weight w

    // initialize
    // -1 means you cannot exactly fill a bag with weight 2 with the first j kinds of items
    for( int i = 0; i < m + 1; i++ ) {
        dp[i] = -1;
    }
    dp[0] = 0;

    // stage transfer
    int ans = 0;
    for( int j = 0; j < N; j++ ) {
        int weight = A[j];
        for( int w = 0; w < m + 1; w++ ) {
            if( dp[w] != -1 && w + weight < m + 1 && dp[w+weight] < dp[w] + V[j]) {
                dp[w+weight] = dp[w] + V[j];
                ans = Math.max(ans, dp[w+weight]);
            }
        }
    }

    return ans;
}
```

背包第四问 每个item只能用一次 有多少种方法拼凑出特定重量target

563. Backpack V

中文 English

Given n items with size `nums[i]` which an integer array and all positive numbers. An integer `target` denotes the size of a backpack. Find the number of possible fill the backpack.

`Each item may only be used once`

Example

Given candidate items `[1,2,3,3,7]` and target `7`,

```
A solution set is:
[7]
[1, 3, 3]
return 2
```

注意这个测试用例：

nums = [1,1,1,1] target = 3

Expected output: 4

```java
public class Solution {
    /**
     * @param nums: an integer array and all positive numbers
     * @param target: An integer
     * @return: An integer
     */
    public int backPackV(int[] nums, int target) {
        // write your code here
        int[] dp = new int[target+1];

        dp[0] = 1;
        for (int i = 0; i < nums.length; i++) {
            int item = nums[i];
            for (int j = target; j >= 0; j--) {
                if (j >= nums[i])  {
                    dp[j] += dp[j-item];
                }
            }
            //System.out.println(Arrays.toString(dp));
        }
        return dp[target];
    }
}
```

只能从后往前，从前往后会重复使用item, 每个item只能使用一次。

状态：dp[j] 表示有dp[j]种方法拼成重量j

转移方程：dp[j] = dp[j] + dp[j-num[i]]

Input

```
[1,2,3,3,7]
7
```

Your stdout

```
[1, 1, 0, 0, 0, 0, 0, 0]
[1, 1, 1, 1, 0, 0, 0, 0]
[1, 1, 1, 2, 1, 1, 1, 0]
[1, 1, 1, 3, 2, 2, 3, 1]
[1, 1, 1, 3, 2, 2, 3, 2]
```

Output

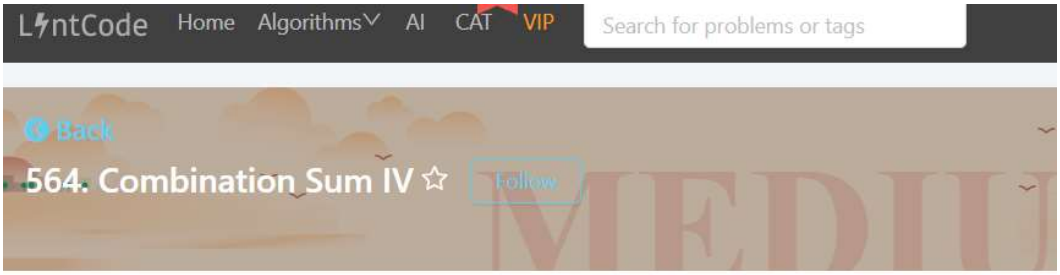
```
2
```

Expected

```
2
```

背包问题第五问 K-sum 问题 Lintcode 564 Leetcode377

这题也类似爬楼梯问题和斐波那契数列



Description

中文 English

Given an integer array `nums` with all positive numbers and no duplicates, find the number of possible combinations that add up to a positive integer `target`.

A number in the array can be used multiple times in the combination.
Different orders are counted as different combinations.

Example

Example1

```
Input: nums = [1, 2, 4], and target = 4
Output: 6
Explanation:
The possible combination ways are:
[1, 1, 1, 1]
[1, 1, 2]
[1, 2, 1]
[2, 1, 1]
[2, 2]
[4]
```

Example2

```
Input: nums = [1, 2], and target = 4
Output: 5
Explanation:
The possible combination ways are:
[1, 1, 1, 1]
[1, 1, 2]
[1, 2, 1]
[2, 1, 1]
[2, 2]
```

377. Combination Sum IV

难度 中等 84 收藏 分享 切换为中文 关注

题目描述 评论 (84) 题解(11) New 提交记录

执行结果: 通过 显示详情 >

执行用时: 2 ms , 在所有 java 提交中击败了 93.44% 的用户

内存消耗: 34 MB , 在所有 java 提交中击败了 41.51% 的用户

炫耀一下:

进行下一个挑战:

合并石头的最低成本	困难
删除一次得到子数组最大和	中等
Build a Tower of Cards	困难

提交时间	提交结果	执行用时	内存消耗	语言
几秒前	通过	2 ms	34 MB	Java
2 分钟前	通过	2 ms	34 MB	Java

Java

```java
class Solution {
    public int combinationSum4(int[] nums, int target) {
        int[] dp = new int[target+1];

        dp[0] = 1;

        for( int i = 1; i < dp.length; i++ ) {

            for( int j = 0; j < nums.length; j++ ) {
                int index = i - nums[j];
                if( index >= 0 ) {
                    dp[i] += dp[index];
                }
            }

        }

        return dp[target];
    }
}
```