

## LintCode 513 Perfect Squares

九章算法

- 题意：
- 给定一个正整数 $n$
- 问最少可以将 $n$ 分成几个完全平方数(1,4,9,...)之和
- 例子：
- 输入： $n=13$
- 输出：2 ( $13=4+9$ )

### 动态规划组成部分一：确定状态

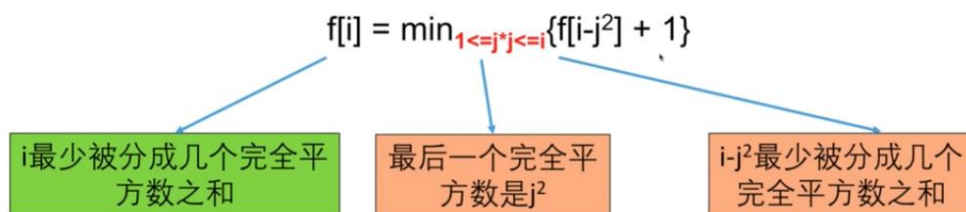
九章算法

- 最后一步：关注最优策略中最后一个完全平方数 $j^2$
- 最优策略中 $n-j^2$ 也一定被划分成最少的完全平方数之和
- 需要知道 $n-j^2$ 最少被分成几个完全平方数之和
- 原来是求 $n$ 最少被分成几个完全平方数之和
- 子问题
- 状态：设 $f[i]$ 表示 $i$ 最少被分成几个完全平方数之和

### 动态规划组成部分二：转移方程

九章算法

- 设 $f[i]$ 表示 $i$ 最少被分成几个完全平方数之和



### 动态规划组成部分三：初始条件和边界情况

九章算法

- 设 $f[i]$ 表示 $i$ 最少被分成几个完全平方数之和
- $f[i] = \min_{1 \leq j^2 \leq i} \{f[i-j^2] + 1\}$
- 初始条件：0被分成0个完全平方数之和  
-  $f[0] = 0$

## 动态规划组成部分四：计算顺序

九章算法

- 初始化 $f[0]$
- 计算 $f[1], \dots, f[N]$
- 答案是 $f[N]$

## LintCode 108 Palindrome Partitioning II

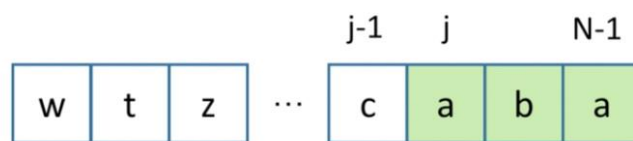
九章算法

- 题意：
- 给定一个字符串 $S[0..N-1]$
- 要求将这个字符串划分成若干段，每一段都是一个回文串（正反看起来一样）
- 求最少划分几次
- 例子：
- 输入：
  - “aab”
- 输出：
  - 1（划分1次→ “aa”, “b”）

## 动态规划组成部分一：确定状态

九章算法

- 最后一步：关注最优策略中最后一段回文串，设为 $S[j..N-1]$
- 需要知道 $S$ 前 $j$ 个字符 $[0..j-1]$ 最少可以划分成几个回文串

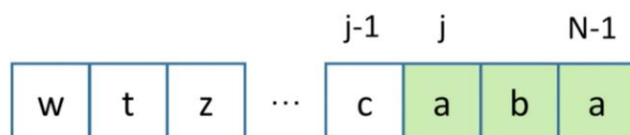


最后一段回文串

## 子问题

九章算法

- 求S前N个字符S[0..N-1]最少划分为几个回文串
- 需要知道S前j个字符[0..j-1]最少可以划分成几个回文串
- 子问题
- 状态：设S前i个字符S[0..i-1]最少可以划分成f[i]个回文串

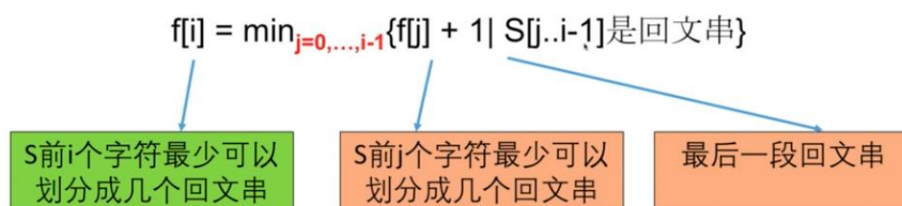


最后一段回文串

## 动态规划组成部分二：转移方程

九章算法

- 设f[i]为S前i个字符S[0..i-1]最少可以划分成几个回文串



## 动态规划组成部分三：初始条件和边界情况

九章算法

- 设f[i]为S前i个字符S[0..i-1]最少可以划分成几个回文串
- $f[i] = \min_{j=0, \dots, i-1} \{f[j] + 1 \mid S[j..i-1] \text{ 是回文串}\}$
- 初始条件：空串可以被分成0个回文串  
–  $f[0] = 0$

## 动态规划组成部分四：计算顺序

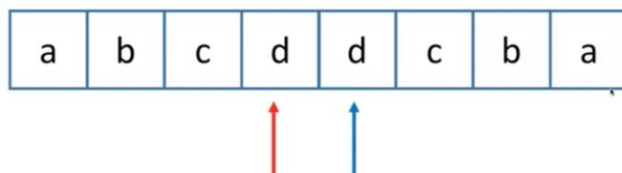
九章算法

- 计算f[0], f[1], ..., f[N]

等等，怎么判断回文串？

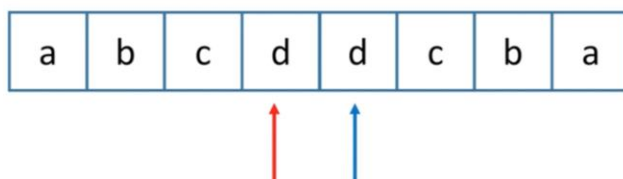
## 回文串判断

- 从左到右和从右到左各读一遍，完全一样
- 可以用两个指针从两头向中间移动，每一步两个指针指向的字符都必须相等



## 回文串判断

- 但是动态规划转移方程是  $f[i] = \min_{j=0, \dots, i-1} \{f[j] + 1 \mid S[j..i-1] \text{ 是回文串} \}$
- 每次都判断  $S[j..i-1]$  是不是回文串很慢



## 回文串种类

- 回文串分两种
  - 长度为奇数
  - 长度为偶数

### 奇数长度回文串



### 偶数长度回文串



## 生成回文串

九章算法

- 假设我们现在不是寻找回文串，而是生成回文串
- 从中间开始，向两边扩展，每次左右两端加上同样的字符

奇数长度回文串



偶数长度回文串



## 在字符串中找到所有回文串

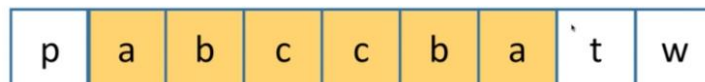
九章算法

- 以字符串的每一个字符为中点，向两边扩展，找到所有回文串

S



S

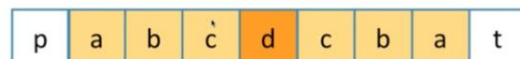


## 记录回文串

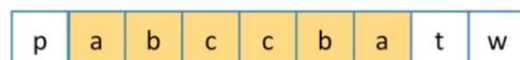
九章算法

- 从S每一个字符开始向两边扩展
  - 考虑奇数长度回文串和偶数长度回文串
- 用isPalin[i][j]表示S[i..j]是否是回文串
- 时间复杂度 $O(N^2)$

S

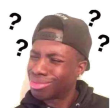


S





- S最少划分成多少个回文串
- $f[i] = \min_{j=0, \dots, i-1} \{f[j] + 1 \mid S[j..i-1] \text{ 是回文串}\}$
- $f[i] = \min_{j=0, \dots, i-1} \{f[j] + 1 \mid \text{isPalin}[j][i-1] = \text{True}\}$
- 答案是  $f[N]-1$  (因为原题是求最少划分几次)
- 时间复杂度  $O(N^2)$ , 空间复杂度  $O(N^2)$



## LintCode 437 Copy Books

- 题意：
- 有N本书需要被抄写，第i本书有A[i]页， $i=0, 1, \dots, N-1$
- 有K个抄写员，每个抄写员可以抄写连续的若干本书（例如：第3~5本书，或者第10本书）
- 每个抄写员的抄写速度都一样：一分钟一页
- 最少需要多少时间抄写完所有的书
- 例子：
- 输入：
  - $A = [3, 2, 4], K=2$
- 输出：
  - 5（第一个抄写员抄写第1本和第2本书，第二个抄写员抄写第3本书）

## 题目分析

- 如果一个抄写员抄写第i本到第j本书，则需要时间  $A[i] + A[i+1] + \dots + A[j]$
- 最后完成时间取决于耗时最长的那个抄写员
- 需要找到一种分段方式，分成不超过K段，使得所有段的数字之和的最大值最小

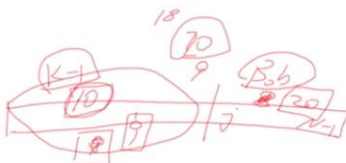




## 动态规划组成部分一：确定状态

九章算法

- 最后一步：最优策略中最后一个抄写员Bob(设他是第K个)抄写的部分  
— 一段连续的书，包含最后一本
- 如果Bob抄写第j本到第N-1本书
- 则Bob需要时间 $A[j] + \dots + A[N-1]$
- 需要知道前面K-1个人最少需要多少时间抄完前j本书(第0~j-1本书)



## 子问题

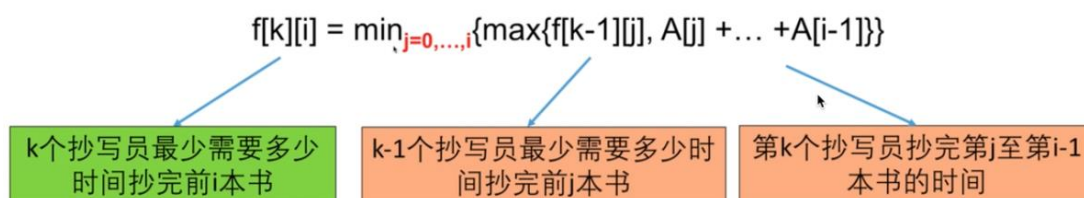
九章算法

- 求K个人最短需要多少时间抄完前N本书
- 需要知道K-1个人最少需要多少时间抄完前j本书
- 子问题
- 状态：设 $f[k][i]$ 为k个抄写员最少需要多少时间抄完前i本书

## 动态规划组成部分二：转移方程

九章算法

- 设 $f[k][i]$ 为k个抄写员最少需要多少时间抄完前i本书



## 动态规划组成部分三：初始条件和边界情况

九章算法

- 设 $f[k][i]$ 为k个抄写员最少需要多少时间抄完前i本书
- $f[k][i] = \min_{j=0, \dots, i} \{ \max\{f[k-1][j], A[j] + \dots + A[i-1]\} \}$
- 初始条件：
  - 0个抄写员只能抄0本书
    - $f[0][0] = 0, f[0][1] = f[0][2] = \dots = f[0][N] = +\infty$
  - k个抄写员( $k > 0$ )需要0时间抄0本书
    - $f[k][0] = 0 (k > 0)$

- 计算  $f[0][0], f[0][1], \dots, f[0][N]$
- 计算  $f[1][0], f[1][1], \dots, f[1][N]$
- ...
- 计算  $f[K][0], f[K][1], \dots, f[K][N]$
- 答案是  $f[K][N]$
- 时间复杂度  $O(N^2K)$ , 空间复杂度  $O(NK)$ , 优化后可以达到  $O(N)$
- 如果  $K > N$ , 可以赋值  $K \leftarrow N$

## 小结

- 划分性动态规划
- 要求将一个序列或字符串划分成若干满足要求的片段
- 解决方法：最后一步  $\rightarrow$  最后一段
- 枚举最后一段的起点
- 如果题目不指定段数, 用  $f[i]$  表示前  $i$  个元素分段后的可行性/最值, 可行性, 方式数: *Perfect Squares, Palindrome Partition II*
- 如果题目指定段数, 用  $f[i][j]$  表示前  $i$  个元素分成  $j$  段后的可行性/最值, 可行性, 方式数: *Copy Books*



## 博弈型动态规划

- 博弈为两方游戏
- 一方先下, 在一定规则下依次出招
- 如果满足一定条件, 则一方胜
- 目标: 取胜



# 博弈型动态规划



- 先手：先出招的一方
- 出招后，先手换人，新的先手面对一个新的局面



### LintCode 394 Coins in a Line

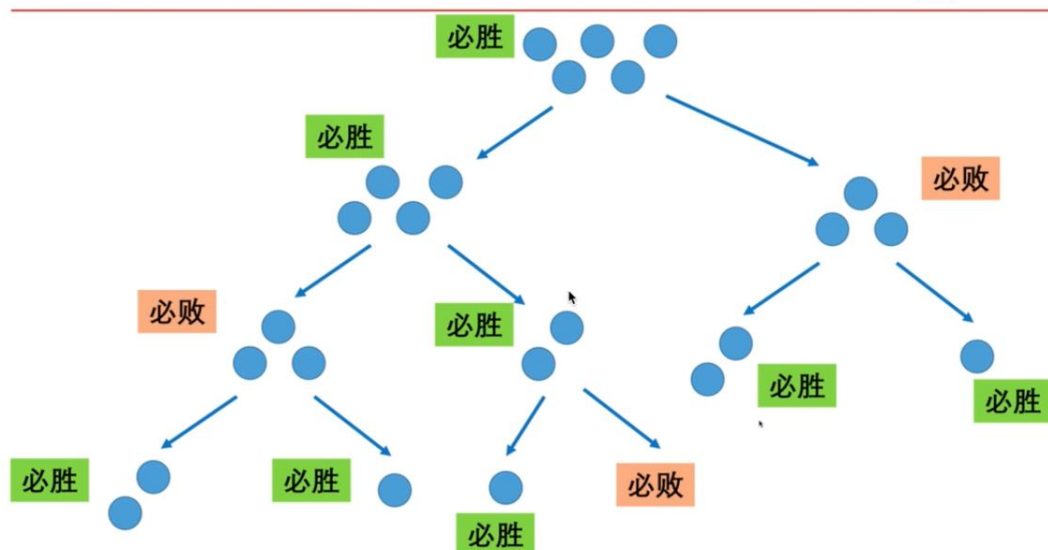
- 题意：
- 有一排N个石子，Alice, Bob两人轮流取石子
- 每次一个人可以从最右边取走1个或2个石子
- 取走最后石子的人胜
- 问先手Alice是否必胜 (先手必胜: true, 先手必败: false)
- 例子：
- 输入：N=5
- 输出：true (先手取走2个石子，剩下3个石子，无论后手怎么拿，先手都可以取走最后一个石子)

### 动态规划组成部分一：确定状态

- 假设后手Bob面对N-1个石子
- 其实这和一开始Bob是先手，有N-1个石子的情况是一样的
- 那么Bob也会选择让自己赢的一步：取走1个或2个石子
- 之后Alice面对新的局面，自己成为新的先手，选择让自己赢的一步

### 博弈动态规划：必胜 vs 必败

- 怎么选择让自己赢的一步
- 就是走了这一步之后，对手面对剩下的石子，他必输



## 知识点

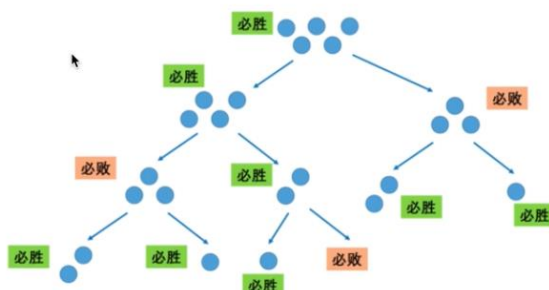
如果取1个或2个石子后，能让剩下的局面先手必败，则当前先手必胜

## 知识点

如果不管怎么走，剩下的局面都是先手必胜，则当前先手必败

## 宗旨

必胜：在当下的局面走一步，让对手无路可逃  
必败：自己无路可逃



- 要求面对N个石子，是否先手必胜
- 需要知道面对N-1个石子和N-2个石子，是否先手必胜
- 子问题
- 状态：设 $f[i]$ 表示面对i个石子，是否先手必胜( $f[i] = \text{TRUE} / \text{FALSE}$ )

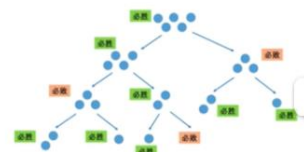
## 动态规划组成部分二：转移方程

九章算法

- 设 $f[i]$ 表示面对 $i$ 个石子，是否先手必胜( $f[i] = \text{TRUE} / \text{FALSE}$ )

$$f[i] = \begin{cases} \text{TRUE, } f[i-1] == \text{FALSE AND } f[i-2] == \text{FALSE} & \text{拿1或2个石子都必胜} \\ \text{TRUE, } f[i-1] == \text{FALSE AND } f[i-2] == \text{TRUE} & \text{拿1个石子必胜} \\ \text{TRUE, } f[i-1] == \text{TRUE AND } f[i-2] == \text{FALSE} & \text{拿2个石子必胜} \\ \text{FALSE, } f[i-1] == \text{TRUE AND } f[i-2] == \text{TRUE} & \text{必败} \end{cases}$$

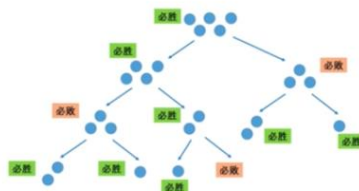
$$f[i] = f[i-1] == \text{FALSE OR } f[i-2] == \text{FALSE}$$



## 动态规划组成部分三：初始条件和边界情况

九章算法

- 设 $f[N]$ 表示面对 $N$ 个石子，是否先手必胜( $f[N] = \text{TRUE} / \text{FALSE}$ )
- $f[i] = f[i-1] == \text{FALSE OR } f[i-2] == \text{FALSE}$
- $f[0] = \text{FALSE}$  --- 面对0个石子，先手必败
- $f[1] = f[2] = \text{TRUE}$  --- 面对1个石子或2个石子，先手必胜

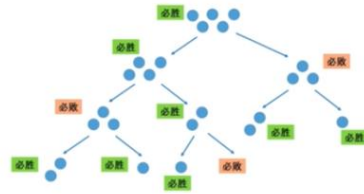




## 动态规划组成部分四：计算顺序

九章算法

- $f[0], f[1], f[2], \dots, f[N]$
- 如果  $f[N] = \text{TRUE}$  则先手必胜，否则先手必败
- 时间复杂度  $O(N)$
- 空间复杂度  $O(N)$ ，可以滚动数组优化至  $O(1)$



## 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



## 直觉

九章算法

- 逐个放物品，看是否还能放入
- 两个关键点
  - 还有几个物品
  - 还剩多少承重



最大承重  
14公斤





- 题意:
- 给定 $N$ 个物品，重量分别为正整数 $A_0, A_1, \dots, A_{N-1}$
- 一个背包最大承重是正整数 $M$
- 最多能带走多重的物品
- 例子：
- 输入：4个物品，重量为2, 3, 5, 7. 背包最大承重是11
- 输出：10 （三个物品：2, 3, 5）

## 背包动态规划：如何下手

- 给定 $N$ 个物品，重量分别为正整数 $A_0, A_1, \dots, A_{N-1}$
- 一个背包最大承重是 $M$
- 物品的重量都是整数
- 每个装物品的方案的总重量都是0到 $M$
- 如果对于每个总重量，我们能知道有没有方案能做到，就可以解决



## 动态规划组成部分一：确定状态

- 需要知道 $N$ 个物品是否能拼出重量 $W$  ( $W = 0, 1, \dots, M$ )
- 最后一步：最后一个物品 $A_{N-1}$  (重量 $A_{N-1}$ ) 是否进入背包

情况一：如果前 $N-1$ 个物品能拼出 $W$ ，当然前 $N$ 个物品也能拼出 $W$

情况二：如果前 $N-1$ 个物品能拼出 $W - A_{N-1}$ ，再加上最后的物品 $A_{N-1}$ ，拼出 $W$





## 动态规划组成部分一：确定状态

九章算法

- 例子：
- 4个物品，重量为2, 3, 5, 7
- 前3个物品可以拼出重量8 (3+5)，自然4个物品可以拼出重量8
- 前3个物品可以拼出重量2 (2)，加上最后一个物品，可以拼出重量9

情况一：如果前N-1个物品能拼出W，当然前N个物品也能拼出W

情况二：如果前N-1个物品能拼出 $W - A_{N-1}$ ，再加上最后的物品 $A_{N-1}$ ，拼出W



## 子问题

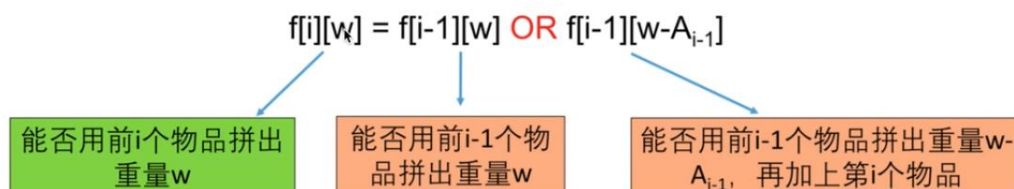
九章算法

- 要求前N个物品能不能拼出重量0, 1, ..., M
- 需要知道前N-1个物品能不能拼出重量0, 1, ..., M
- 子问题
- 状态：设 $f[i][w]$  = 能否用前i个物品拼出重量w (TRUE / FALSE)
- 常见误区：**错误** 设 $f[i]$ 表示前i个物品能拼出的最大重量（不超过Target）
  - 反例：A=[3 9 5 2], Target=10
  - 错误原因：最优策略中，前N-1个物品拼出的不一定是超过Target的最大重量

## 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)





### 动态规划组成部分三：初始条件和边界情况



- $f[i][w] = f[i-1][w] \text{ OR } f[i-1][w-A_{i-1}]$
- 初始条件：
  - $f[0][0] = \text{TRUE}$ : 0个物品可以拼出重量0
  - $f[0][1..M] = \text{FALSE}$ : 0个物品不能拼出大于0的重量
- 边界情况：
  - $f[i-1][w-A_{i-1}]$ 只能在 $w \geq A_{i-1}$ 时使用

#### 例子



|                    |   |   |   |   |   |   |   |   |   |   |    |    |
|--------------------|---|---|---|---|---|---|---|---|---|---|----|----|
|                    | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 0个物品               | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗  | ✗  |
| 前1个物品<br>(重量2)     | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗  | ✗  |
| 前2个物品<br>(重量2,3)   | ✓ | ✗ | ✓ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗  | ✗  |
| 前3个物品<br>(重量2,3,5) | ✓ | ✗ | ✓ | ✓ | ✗ | ✓ | ✗ | ✓ | ✓ | ✗ | ✓  | ✗  |

### 动态规划组成部分四：计算顺序



- 初始化 $f[0][0], f[0][1], \dots, f[0][M]$
- 计算前1个物品能拼出哪些重量： $f[1][0], f[1][1], \dots, f[1][M]$
- 计算前2个物品能拼出哪些重量： $f[2][0], f[2][1], \dots, f[2][M]$
- ...
- 计算前N个物品能拼出哪些重量： $f[N][0], f[N][2], \dots, f[N][M]$
- 时间复杂度（计算步数）： $O(MN)$ ，空间复杂度（数组大小）：优化后可以达到 $O(M)$

## 小结

九章算法

- 要求不超过Target时能拼出的最大重量
- 记录前i个物品能拼出哪些重量
- 前i个物品能拼出的重量：
  - 前i-1个物品能拼出的重量
  - 前i-1个物品能拼出的重量+第i个物品重量 $A_{i-1}$



## LintCode 563: Backpack V

九章算法

- 题意:
- 给定N个正整数： $A_0, A_1, \dots, A_{N-1}$
- 一个正整数Target
- 求有多少种组合加起来是Target
- 每个 $A_i$ 只能用一次
- 例子：
- 输入： $A=[1, 2, 3, 3, 7]$ , Target=7
- 输出：2 ( $7=7, 1+3+3=7$ )

## 题目分析

九章算法

- N个正整数： $A_0, A_1, \dots, A_{N-1} \rightarrow$  N个物品的重量为 $A_0, A_1, \dots, A_{N-1}$
- 要求和是Target  $\rightarrow$  正好装满一个载重为Target的背包
- 背包问题
- 和前一题Backpack不一样的是，我们要求出有多少组合的和是Target，而不是能不能拼出Target
- 当然，如果能知道这N个物品有多少种方式拼出0，有多少种方式拼出1，...，有多少种方式拼出Target，也就得到了答案

## 动态规划组成部分一：确定状态



- 需要知道N个物品有多少种方式拼出重量W ( $W = 0, 1, \dots, \text{Target}$ )
- 最后一步：第N个物品（重量 $A_{N-1}$ ）是否进入背包

情况一：用前N-1个物品拼出W

情况二：用前N-1个物品能拼出 $W - A_{N-1}$ ，再加上最后的物品 $A_{N-1}$ ，拼出W

情况一的个数+情况二的个数 = 用前N个物品拼出W的方式

## 子问题



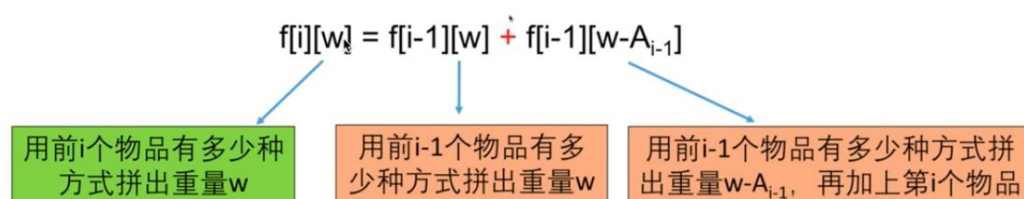
- 要求前N个物品有多少种方式拼出重量0, 1, ..., Target
- 需要知道前N-1个物品有多少种方式拼出重量0, 1, ..., Target
- 子问题
- 状态：设 $f[i][w]$  = 用前i个物品有多少种方式拼出重量w

背包问题总承重一定要进状态。

## 动态规划组成部分二：转移方程



- 设 $f[i][w]$  = 用前i个物品有多少种方式拼出重量w



## 动态规划组成部分三：初始条件和边界情况



- $f[i][w] = f[i-1][w] + f[i-1][w-A_{i-1}]$
- 初始条件：
  - $f[0][0] = 1$ : 0个物品可以有一种方式拼出重量0
  - $f[0][1..M] = 0$ : 0个物品不能拼出大于0的重量
- 边界情况：
  - $f[i-1][w-A_{i-1}]$ 只能在 $w \geq A_{i-1}$ 时使用

## 动态规划组成部分四：计算顺序



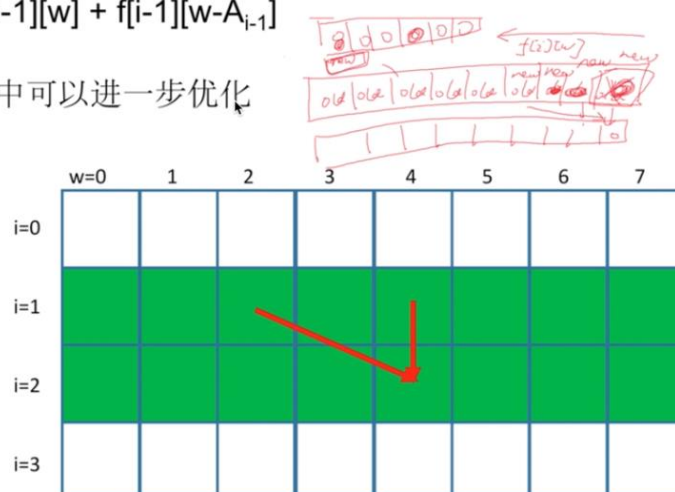
- 初始化  $f[0][0], f[0][1], \dots, f[0][\text{Target}]$
- 计算前1个物品有多少种方式拼出重量： $f[1][0], f[1][1], \dots, f[1][\text{Target}]$
- ...
- 计算前N个物品有多少种方式拼出重量： $f[N][0], f[N][2], \dots, f[N][\text{Target}]$
- 答案是  $f[N][\text{Target}]$
- 时间复杂度（计算步数）： $O(N \cdot \text{Target})$ ，空间复杂度（数组大小）：优化后可以达到  $O(\text{Target})$

## 进一步空间优化



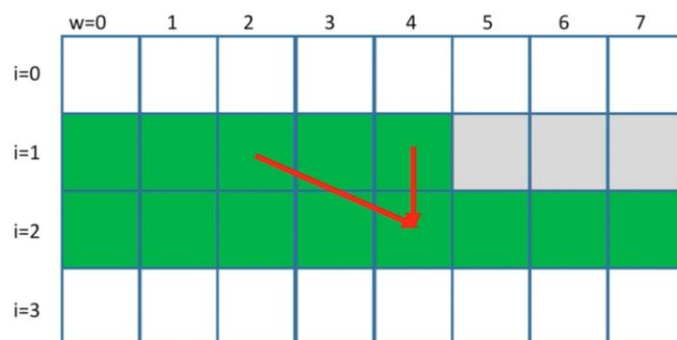
- $f[i][w] = f[i-1][w] + f[i-1][w-A_{i-1}]$

- 实际编程中可以进一步优化



实际中，可以只开一个数组，从右往左计算。

- $f[i][w] = f[i-1][w] + f[i-1][w-A_{i-1}]$
- 可以只开一个数组
- 按照  $f[i][\text{Target}], \dots, f[i][0]$  的顺序更新



## LintCode 564: Backpack VI

- 题意:
- 给定N个正整数:  $A_0, A_1, \dots, A_{N-1}$
- 一个正整数Target
- 求有多少种组合加起来是Target
- 每个  $A_i$  可以用多次
- 例子:
- 输入:  $A = [1, 2, 4]$ , Target = 4
- 输出: 6 (1+1+1+1=4, 2+2=4, 1+1+2=4, 1+2+1=4, 2+1+1=4, 4=4)

## 题目分析

- 和Backpack V唯一区别: 组合中数字可以按不同的顺序, 比如1+1+2与1+2+1算两种组合
- 不能先处理第一个物品, 再处理第二个物品, ...
- 似乎是更难的背包问题
- 其实更简单☺



## 动态规划组成部分一：确定状态



- 回到一维的思路
  - 第一讲硬币组合题
  - 用最少的硬币数凑成27元
- 关注最后一步：最后一个物品的重量是多少

### 关键点1

任何一个正确的组合中，所有物品总重量是Target

### 关键点2

如果最后一个物品重量是K，则前面的物品重量是Target-K

## 动态规划组成部分一：确定状态



- 如果最后一个物品重量是 $A_0$ ，则要求有多少种组合能拼成Target -  $A_0$
- 如果最后一个物品重量是 $A_1$ ，则要求有多少种组合能拼成Target -  $A_1$
- ...
- 如果最后一个物品重量是 $A_{N-1}$ ，则要求有多少种组合能拼成Target -  $A_{N-1}$

### 关键点1

任何一个正确的组合中，所有物品总重量是Target

### 关键点2

如果最后一个物品重量是K，则前面的物品重量是Target-K

## 动态规划组成部分一：确定状态



- 如果最后一个物品重量是 $A_0$ ，则要求有多少种组合能拼成Target -  $A_0$
- 如果最后一个物品重量是 $A_1$ ，则要求有多少种组合能拼成Target -  $A_1$
- ...
- 如果最后一个物品重量是 $A_{N-1}$ ，则要求有多少种组合能拼成Target -  $A_{N-1}$

### 关键点1

任何一个正确的组合中，所有物品总重量是Target

### 关键点2

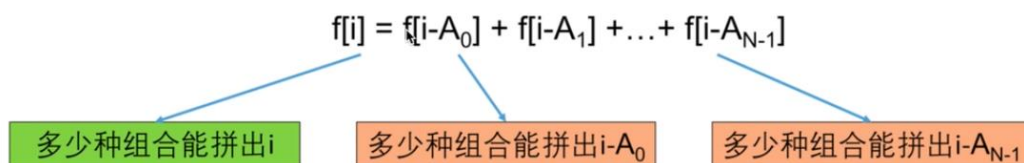
如果最后一个物品重量是K，则前面的物品重量是Target-K



- 如果最后一个物品重量是 $A_0$ , 则要求有多少种组合能拼成 $\text{Target} - A_0$
- 如果最后一个物品重量是 $A_1$ , 则要求有多少种组合能拼成 $\text{Target} - A_1$
- ...
- 如果最后一个物品重量是 $A_{N-1}$ , 则要求有多少种组合能拼成 $\text{Target} - A_{N-1}$
- 原问题要求有多少种组合能拼成 $\text{Target}$
- 子问题
- 设 $f[i]$  = 有多少种组合能拼出重量 $i$

## 动态规划组成部分二：转移方程

- 设 $f[i]$  = 有多少种组合能拼出重量 $i$



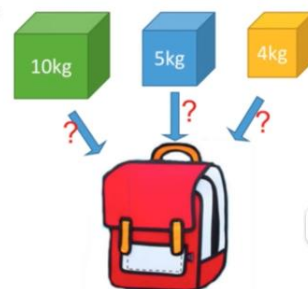
## 动态规划组成部分三：初始条件和边界情况

- $f[i] = f[i-A_0] + f[i-A_1] + \dots + f[i-A_{N-1}]$
- $f[0] = 1$ 
  - 有1种组合能拼出重量0 (什么都不选)
- 如果 $i < A_j$ , 则对应的 $f[i-A_j]$ 不加入 $f[i]$ 
  - $A_0=1, A_1=2, A_2=4$
  - $f[3] = f[3-A_0] + f[3-A_1]$

- 设  $f[i]$  = 有多少种组合能拼出重量  $i$
- $f[i] = f[i-A_0] + f[i-A_1] + \dots + f[i-A_{N-1}]$
- $f[0] = 1$
- 计算  $f[1], f[2], \dots, f[\text{Target}]$
- 结果为  $f[\text{Target}]$

## 小结

- Backpack 可行性背包
  - 题目要求最多装多少重量
  - 记录前  $i$  个物品能不能拼出重量  $w$
- Backpack V, Backpack VI, 计数型背包
  - 题目要求有多少种方式装出重量
  - Backpack V：记录前  $i$  个物品有多少种方式拼出重量  $W$
  - Backpack VI：记录有多少种方式拼出重量  $W$
- 关键点
  - 最后一步
    - 最后一个背包内的物品是哪个



## 小结

- Backpack 可行性背包
  - 要求不超过  $\text{Target}$  时能拼出的最大重量
  - 记录前  $i$  个物品能不能拼出重量  $w$
- Backpack V, Backpack VI, 计数型背包
  - 要求有多少种方式拼出重量  $\text{Target}$
  - Backpack V：记录前  $i$  个物品有多少种方式拼出重量  $w$
  - Backpack VI：记录有多少种方式拼出重量  $w$
- 关键点
  - 最后一步
    - 最后一个背包内的物品是哪个
    - 最后一个物品有没有进背包



- 划分型动态规划
- 博弈型动态规划
- 背包型动态规划