本节大纲

- 有状态的序列型动态规划
- 状态划分
- 特点：
  - 题目变种多
  - 没有固定模板
  - 见招拆招

序列型动态规划

- 给定一个序列
- 动态规划方程f[i]中的下标i表示**前i个元素a[0], a[1], …, a[i-1]**的某种性质
  - 坐标型的f[i]表示以a_i为结尾的某种性质
- 初始化中，f[0]表示空序列的性质
  - 坐标型动态规划的初始条件f[0]就是指以a_0为结尾的子序列的性质

序列型动态规划

LintCode 516 Paint House II

- 题意：
- 有一排N栋房子，每栋房子要漆成K种颜色中的一种
- 任何两栋相邻的房子不能漆成同样的颜色
- 房子i染成第j种颜色的花费是cost[i][j]
- 问最少需要花多少钱油漆这些房子

- 例子：
- 输入：
  - N=3, K=3
  - Cost = [[14,2,11],[11,14,5],[14,3,10]]
- 输出：
  - 10 （房子0蓝色，房子1绿色，房子2蓝色, 2+5+3=10）

题目分析

- 这题和Paint House非常类似，只是颜色种类变成K种
- 动态规划思路和Paint House一样，需要记录油漆前i栋房子并且房子i-1是颜色1, 颜色2, …, 颜色K的最小花费：f[i][1], f[i][2], …, f[i][K]

动态规划组成部分二：转移方程

- 设油漆前i栋房子并且房子i-1是颜色1, 颜色2, …颜色K的最小花费分别为f[i][1], f[i][2], …, f[i][K]

f[i][1] = min{f[i-1][2] + cost[i-1][1], f[i-1][3] + cost[i-1][1], …, f[i-1][K] + cost[i-1][1]}

f[i][2] = min{f[i-1][1] + cost[i-1][2], f[i-1][3] + cost[i-1][2], …, f[i-1][K] + cost[i-1][2]}

…

f[i][K] = min{f[i-1][1] + cost[i-1][K], f[i-1][2] + cost[i-1][K], …, f[i-1][K-1] + cost[i-1][K]}

动态规划组成部分二：转移方程

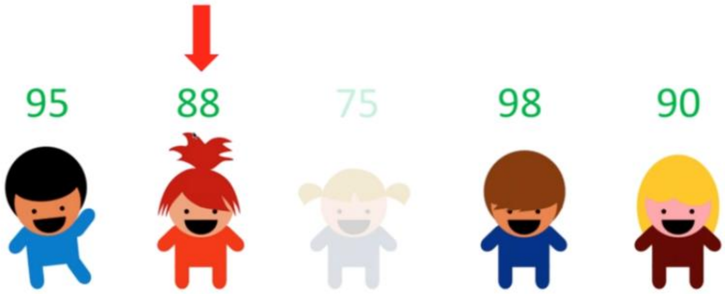
- 设油漆前i栋房子并且房子i-1是颜色1, 颜色2, …颜色K的最小花费分别为f[i][1], f[i][2], …, f[i][K]

- $f[i][j] = \min_{k\neq j} \{f[i-1][k]\} + cost[i-1][j]$
- 直接计算的时间复杂度（计算步数）：
  - i从0到N
  - j从1到K
  - k从1到K
  - $O(NK^2)$

能不能加快？

动态规划常见优化

- $f[i][j] = \min_{k \neq j} \{f[i-1][k]\} + cost[i-1][j]$
- 每次需要求f[i-1][1], …, f[i-1][K]中除了一个元素之外其他元素的最小值



动态规划常见优化

- $f[i][j] = \min_{k \neq j} \{f[i-1][k]\} + cost[i-1][j]$

如果最小值是第i个元素，次小值是第j个元素
1. 只要除掉的元素不是第i个，剩下的最小值就是第i个元素
2. 如果除掉的元素是第i个，剩下的最小值就是第j个元素

动态规划常见优化

- $f[i][j] = \min_{k \neq j} \{f[i-1][k]\} + cost[i-1][j]$
- 记录下f[i-1][1], …, f[i-1][K]中的最小值和次小值分别是哪个
- 假如最小值是f[i-1][a], 次小值是f[i-1][b]
- 则对于j=1,2,3,…,a-1, a+1,…,K, $f[i][j] = f[i-1][a] + cost[i-1][j]$
- $f[i][a] = f[i-1][b] + cost[i-1][a]$、
- 时间复杂度降为O(NK)

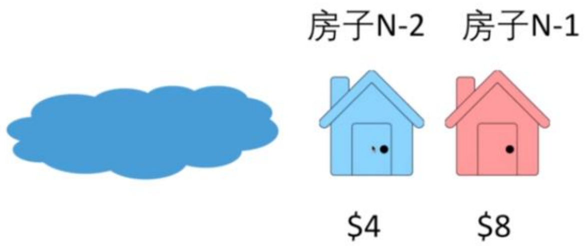
LintCode 392 House Robber

- 题意：
- 有一排N栋房子(0~N-1)，房子i里有A[i]个金币
- 一个窃贼想选择一些房子偷金币
- 但是不能偷任何挨着的两家邻居，否则会被警察逮住
- 最多偷多少金币

- 例子：
- 输入：A={3, 8, 4}
- 输出：8 （只偷第二家的金币）

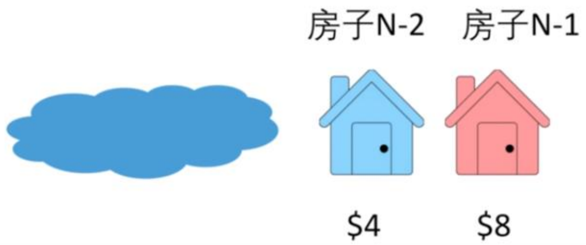
动态规划组成部分一：确定状态

- 最后一步：窃贼的最优策略中，有可能偷或者不偷最后一栋房子N-1
- 情况1：不偷房子N-1
  - 简单，最优策略就是前N-1栋房子的最优策略
- 情况2：偷房子N-1
  - 仍然需要知道在前N-1栋房子中最多能偷多少金币，但是，需要保证不偷**第N-2栋房子**



动态规划组成部分一：确定状态

- 如何知道在不偷**房子N-2**的前提下，在前N-1栋房子中最多能偷多少金币呢？
  - 用f[i][0]表示**不偷**房子i-1的前提下，前i栋房子中最多能偷多少金币
  - 用f[i][1]表示**偷**房子i-1的前提下，前i栋房子中最多能偷多少金币



动态规划组成部分二：转移方程

- 设f[i][0]为**不偷**房子i-1的前提下，前i栋房子中最多能偷多少金币
- 设f[i][1]为**偷**房子i-1的前提下，前i栋房子中最多能偷多少金币

$$f[i][0] = \max\{f[i-1][0], f[i-1][1]\}$$

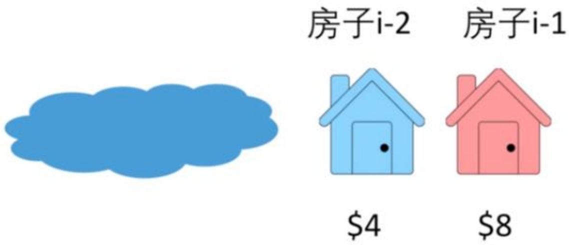
因为**不偷**房子i-1，所以房子i-2可以选择偷或不偷

$$f[i][1] = f[i-1][0] + A[i-1]$$

偷房子i-1，房子i-2必须不偷

简化

- 在**不偷**房子i-1的前提下，前i栋房子中最多能偷多少金币
  - 其实这就是前i-1栋房子最多能偷多少金币
- 所以我们可以简化前面的表示



动态规划组成部分三：初始条件和边界情况

- 设f[i]为窃贼在前i栋房子最多偷多少金币
- $f[i] = \max\{f[i-1], f[i-2] + A[i-1]\}$
- 初始条件：
  - $f[0] = 0$ （没有房子，偷0枚金币）
  - $f[1] = A[0]$
  - $f[2] = \max\{A[0], A[1]\}$

动态规划组成部分四：计算顺序

- 初始化f[0]
- 计算f[1], f[2], …, f[n]
- 答案是f[n]
- 时间复杂度O(N), 空间复杂度O(1)

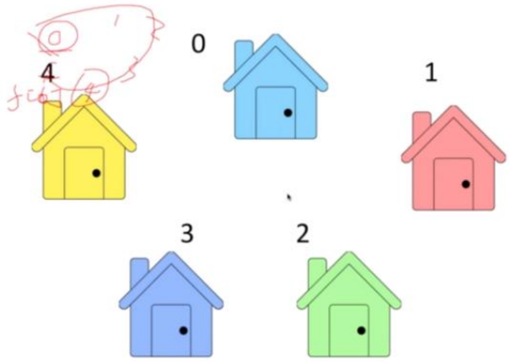
LintCode 534 House Robber II

- 题意：
- 有一圈N栋房子，房子i-1里有A[i]个金币
- 一个窃贼想选择一些房子偷金币
- 但是不能偷任何挨着的两家邻居，否则会被警察逮住
- 最多偷多少金币

- 例子：
- 输入：A={3, 8, 4}
- 输出：8 （只偷房子1的金币）

题意分析

- 这题和House Robber非常类似，只是房子现在排成一个圈
- 于是房子0和房子N-1成了邻居，不能同时偷盗
- 要么没偷房子0
- 要么没偷房子N-1

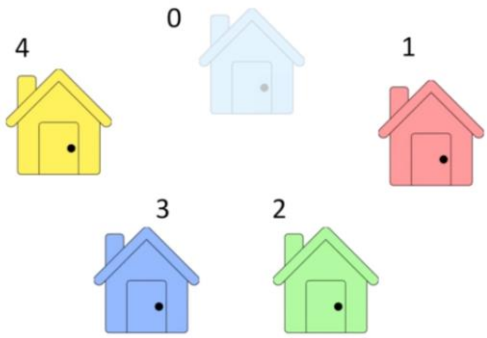


题意分析

- 我们可以枚举窃贼是没有偷房子0还是没有偷房子N-1

情况1：没偷房子0

最优策略就是窃贼对于房子1~N-1的最优策略→化为House Robber



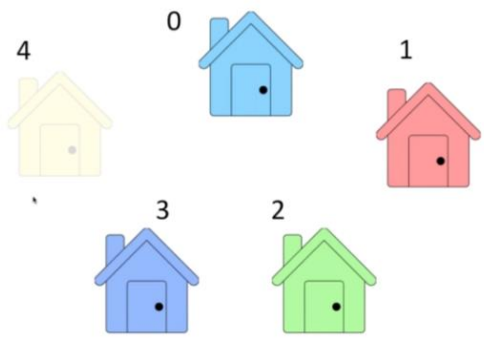
题意分析

Closing "QuickTime Player.app" may help your Mac cool down.

- 我们可以枚举窃贼是没有偷房子0还是没有偷房子N-1

情况2：没偷房子N-1

最优策略就是窃贼对于房子0~N-2的最优策略→化为House Robber



小结

Closing "QuickTime Player.app" may help your Mac cool down.

- 圈的情况比序列复杂
- 但是，通过对于房子0和房子N-1不能同时偷的原理，进行分情况处理
- 经过处理，变成序列情况
- 问题迎刃而解

LintCode 149 Best Time to Buy and Sell Stock

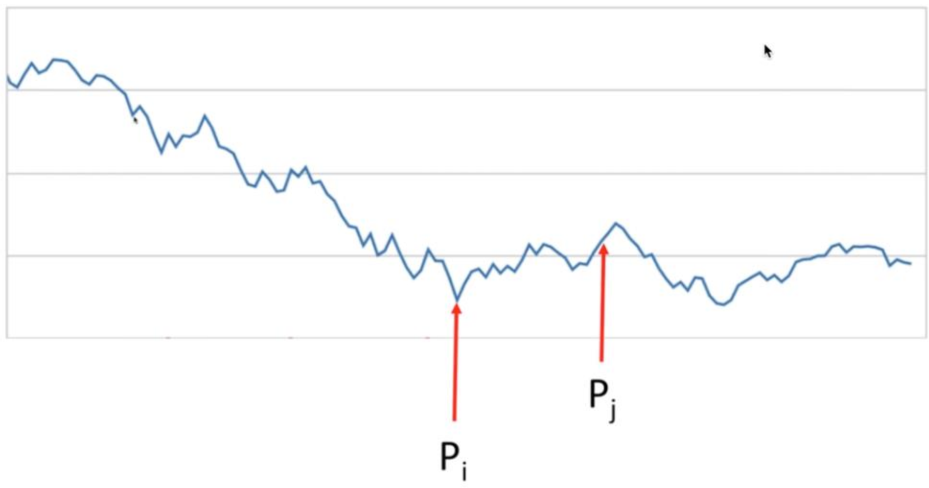
- 题意：
- 已知后面N天一支股票的每天的价格$P_0, P_1, \ldots, P_{N-1}$
- 可以最多买一股卖一股
- 求最大利润

- 例子：
- 输入：[3, 2, 3, 1, 2]
- 输出：1 (2买入，3卖出)

题目分析

- 保底策略：什么都不做，利润0
- **低买高卖，先买后卖**
- 如果买卖一股，一定是第i天买，第j天卖 (j > i)，获利是$P_j - P_i$
- 枚举j，即第几天卖
- 显然，希望找到最小的买入价格P_i (i < j)

图示



动态规划解法

- 从0到N-1枚举j，即第几天卖
- 时刻保存当前为止（即0~j-1天）的最低价格P_i
- 最大的P_j - P_i即为答案

LintCode 150 Best Time to Buy and Sell Stock II

- 题意：
- 已知后面N天一支股票的每天的价格P_0, P_1, …, P_{N-1}
- 可以买卖一股任意多次，但任意时刻手中最多持有一股
- 求最大利润

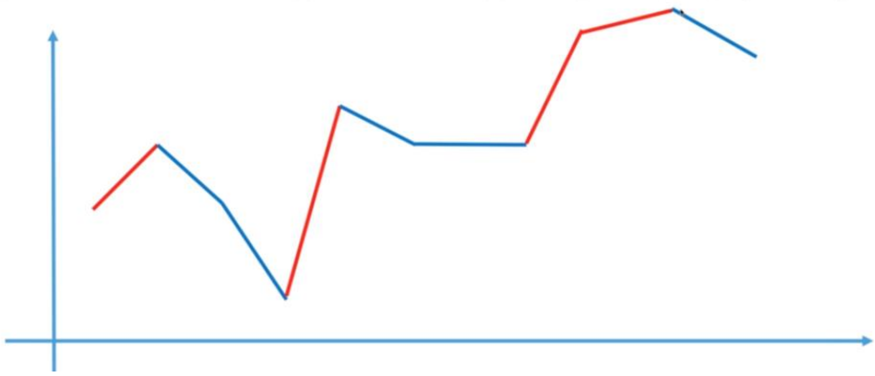
- 例子：
- 输入：[2, 1, 2, 0, 1]
- 输出：2 (1买入，2卖出，0买入，1卖出)

题目分析

- 买卖任意多次

题目分析

- 买卖任意多次
- 最优策略是如果今天的价格比明天的价格低，就今天买，明天卖（**贪心**）



题目分析

- 正确性证明可以从这里下手：
  – 如果最优策略第10天买，第15天卖，我们可以把它分解成5天，结果不会变差

LintCode 151: Best Time to Buy and Sell Stock III

- 题意：
- 给定一支股票N天的价格
- 可以进行最多两次买+两次卖，每次买卖都是一股
- 不能在卖光手中股票前买入，但可以同一天卖完后买入
- 问最大收益

- 例子：
- 输入：[4,4,6,1,1,4,2,5]
- 输出：6 (4买入，6卖出，1买入，5卖出，收益为(6-4) + (5-1) = 6)

题目分析

- 题目大意和I, II基本相似
- 只能最多两次买卖
- 所以需要记录已经买卖了多少次

动态规划组成部分一：确定状态

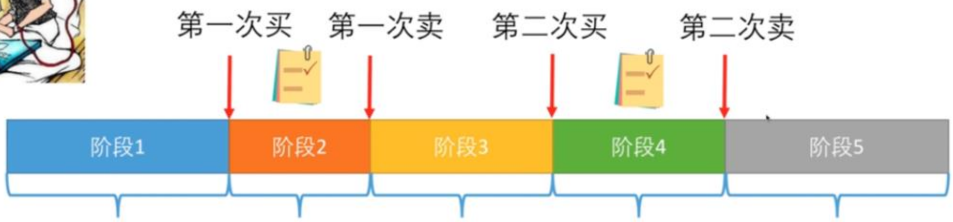
- 最后一步：最优策略中，最后一次卖发生在第j天
- 枚举最后一次买发生在第几天
- 但是不知道之前有没有买卖过

f[i]　f[i]

买过？没买过？

记录阶段

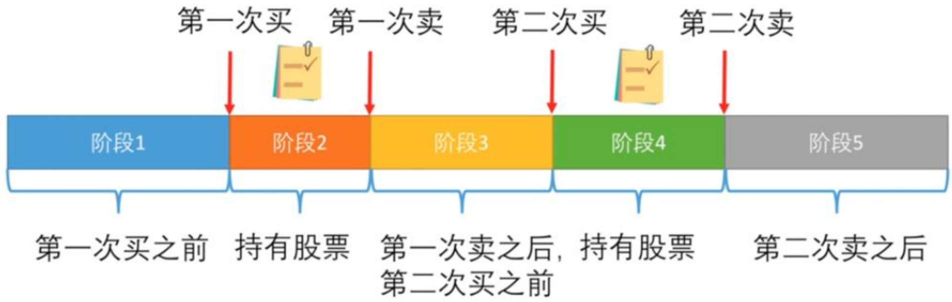
- **不知道有没有买过，就记录下来**



第一次买之前　持有股票　第一次卖之后，第二次买之前　持有股票　第二次卖之后

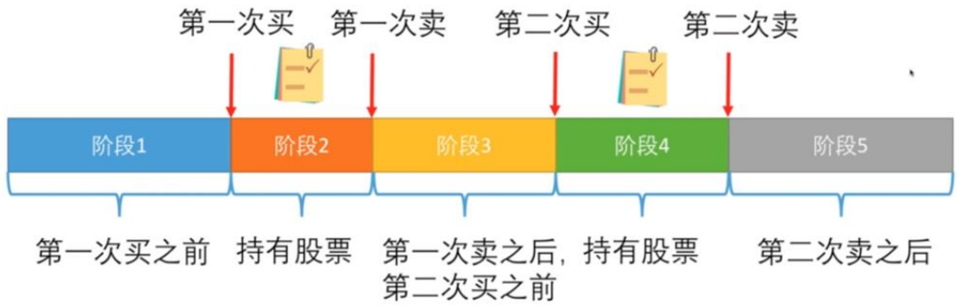
记录阶段

- 阶段可以保持：即不进行买卖操作
- 阶段可以变化：买或卖
  - 在阶段2，卖了一股后，进入阶段3 第i天
  - 在阶段2，卖了一股后**当天**买一股，进入阶段4



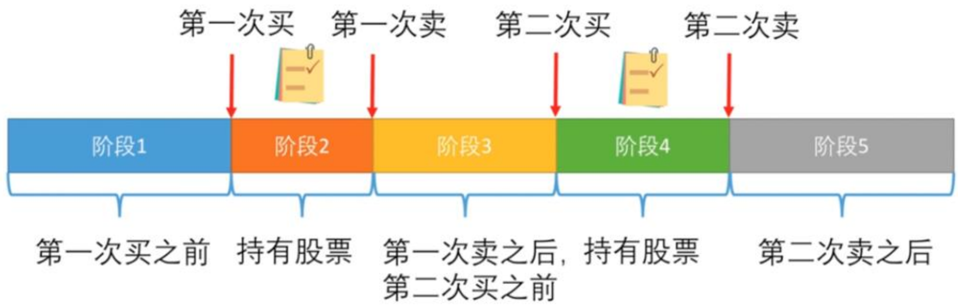
动态规划组成部分一：确定状态

- 最优策略一定是前N天(第N-1天)结束后，处于
  - 阶段1：没买卖过
  - 阶段3：买卖过一次
  - 阶段5：买卖过两次



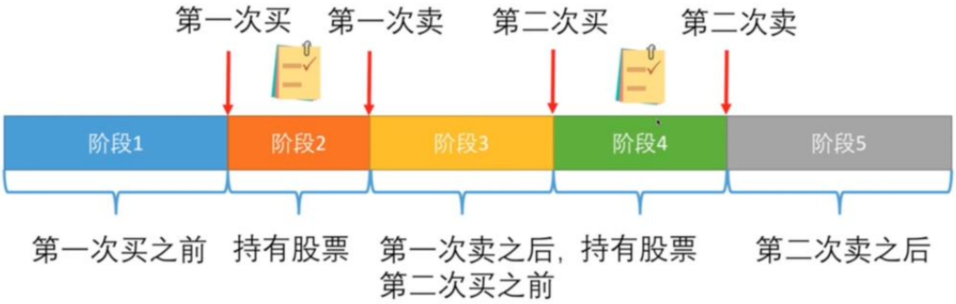
动态规划组成部分一：确定状态

- 例如，如果要求前N天(第N-1天)结束后，在阶段5的最大获利，设为f[N][5]
  - 情况1：第N-2天就在阶段5 --- f[N-1][5]
  - 情况2：第N-2天还在阶段4（第二次持有股票），第N-1天卖掉
    - f[N-1][4] + $(P_{N-1}-P_{N-2})$



动态规划组成部分一：确定状态

- 例如，如果要求前N天(第N-1天)结束后，在阶段4的最大获利，设为f[N][4]
  - 情况1：第N-2天就在阶段4 --- f[N-1][4] + (P_{N-1}-P_{N-2})
  - 情况2：第N-2天还在阶段3 --- f[N-1][3]
  - 情况3：第N-2天还在阶段2，第N-1天卖完了立即买 --- f[N-1][2] + (P_{N-1}-P_{N-2})



子问题

- 要求f[N][1], …, f[N][5]
- 需要知道f[N-1][1], …, f[N-1][5]
- 子问题
- 状态：f[i][j]表示前i天(第i-1天)结束后，在阶段j的最大获利

动态规划组成部分二：转移方程

- f[i][j]:前i天(第i-1天)结束后，处在阶段j, 最大获利

阶段1, 3, 5 --- 手中无股票状态:

$$f[i][j] = \max\{f[i-1][j],\ f[i-1][j-1] + P_{i-1} - P_{i-2}\}$$

昨天没有持有股票

昨天持有股票，今天卖出清仓

阶段2, 4 --- 手中有股票状态:

$$f[i][j] = \max\{f[i-1][j] + P_{i-1} - P_{i-2},\ f[i-1][j-1],\ f[i-1][j-2] + P_{i-1} - P_{i-2}\}$$

昨天就持有股票，继续持有并获利

昨天没有持有股票，今天买入

昨天持有上一次买的股票，今天卖出并立即买入

动态规划组成部分三：初始条件和边界情况

- 刚开始（前0天）处于阶段1
  - $f[0][1] = 0$
  - $f[0][2] = f[0][3] = f[0][4] = f[0][5] = -\infty$
- 阶段1, 3, 5: $f[i][j] = \max\{f[i-1][j], f[i-1][j-1] + P_{i-1} - P_{i-2}\}$
- 阶段2, 4: $f[i][j] = \max\{f[i-1][j] + P_{i-1} - P_{i-2}, f[i-1][j-1], f[i-1][j-2] + P_{i-1} - P_{i-2}\}$
- 如果$j - 1 < 1$或$j - 2 < 1$或$i - 2 < 0$, 对应项不计入max
- 因为**最多**买卖两次，所以答案是$\max\{f[N][1], f[N][3], f[N][5]\}$, 即清仓状态下最后一天最大获利

动态规划组成部分四：计算顺序

- 初始化$f[0][1], \ldots, f[0][5]$
- $f[1][1], \ldots, f[1][5]$
- …
- $f[N][1], \ldots, f[N][5]$
- 时间复杂度：O(N), 空间复杂度：O(N)，优化后可以O(1)，因为$f[i][1..5]$只依赖于$f[i-1][1..5]$

LintCode 393 Best Time To Buy and Sell Stock IV

- 题意：
- 给定一支股票N天的价格
- 可以进行最多K次买+K次卖，每次买卖都是一股
- 不能在卖光手中股票前买入，但可以同一天卖完后买入
- 问最大收益

- 例子：
- 输入：[4,4,6,1,1,4,2,5], K = 2
- 输出：6 (4买入，6卖出，1买入，5卖出，收益为(6-4) + (5-1) = 6)

题目分析

- 首先，如果K很大，K>N/2，则题目可以化简成为Best Time to Buy and Sell Stock II，每天买入当且仅当价格比下一天低

思考：为什么是N/2

- Best Time to Buy and Sell Stock III 相当于这题中K = 2
- 所以我们可以借鉴之前的解法

记录阶段

记录阶段

- 阶段1：没买卖过
- 阶段3：买卖过一次，现在空仓
- 阶段5：买卖过两次，现在空仓
- …
- 阶段2K+1：买卖过K次，现在空仓

- 阶段2：第一次持有，还没有卖
- 阶段4：第二次持有，还没有卖
- 阶段6：第三次持有，还没有卖
- …
- 阶段2K：第K次持有，还没有卖

动态规划组成部分二：转移方程

- f[i][j]: 前i天(第i-1天)结束后，处在阶段j, 最大获利

阶段1, 3, 5, .., 2K+1 --- 手中无股票状态:

$f[i][j] = \max\{f[i-1][j], f[i-1][j-1] + P_{i-1} - P_{i-2}\}$

- 昨天没有持有股票
- 昨天持有股票，今天卖出清仓

阶段2, 4, .., 2K --- 手中有股票状态:

$f[i][j] = \max\{f[i-1][j] + P_{i-1} - P_{i-2}, f[i-1][j-1], f[i-1][j-2] + P_{i-1} - P_{i-2}\}$

- 昨天就持有股票，继续持有并获利
- 昨天没有持有股票，今天买入
- 昨天持有上一次买的股票，今天卖出并立即买入

动态规划组成部分三：初始条件和边界情况

- 刚开始（前0天）处于阶段1
  - $f[0][1] = 0$
  - $f[0][2] = f[0][3] = \ldots = f[0][2K+1] = +\infty$
- 阶段1, 3, 5, .., 2K+1: $f[i][j] = \max\{f[i-1][j], f[i-1][j-1] + P_{i-1} - P_{i-2}\}$
- 阶段2, 4, .., 2K: $f[i][j] = \max\{f[i-1][j] + P_{i-1} - P_{i-2}, f[i-1][j-1], f[i-1][j-2] + P_{i-1} - P_{i-2}\}$
- 如果$j - 1 < 1$或$j - 2 < 1$或$i - 2 < 0$, 对应项不计入max
- 因为**最多**买卖K次，所以答案是$\max\{f[N][1], f[N][3], .., f[N][2K+1]\}$, 即清仓状态下最后一天最大获利

动态规划组成部分四：计算顺序

- 初始化f[0][1], …, f[0][2K+1]
- f[1][1], …, f[1][2K+1]
- …
- f[N][1], …, f[N][2K+1]
- 时间复杂度：O(NK), 空间复杂度：O(NK)，优化后可以O(K)，因为f[i][1.. 2K+1]只依赖于f[i-1][1.. 2K+1]

序列+状态型动态规划小结

- 当思考动态规划最后一步时，这一步的选择依赖于前一步的某种状态

题目	最后一步需要知道的信息	一维+状态
Paint House	房子N-1油漆成红色，则房子N-2不能油漆成红色	记录油漆前N-1栋房子**并且**房子N-2是红/蓝/绿色的最小花费
House Robber	偷房子N-1时，不能也偷房子N-2	记录偷前N-1栋房子**并且**偷/不偷房子N-2的最大收获
Best Time to Buy and Sell Stock III	第j天卖股票，第i天买股票(i<j)时，需要知道第i天之前是不是已经买了股票	记录前N天买卖股票最大获利，**并且**第N-1天：1. 未买卖股票; 2. 买了第一次股票还没卖; ...; 5. 已经第二次卖了股票

序列+状态型动态规划小结

- 当思考动态规划最后一步时，这一步的选择依赖于前一步的某种状态

- 初始化时，f[0]代表前0个元素/前0天的情况
  - 与坐标型动态规划区别

- 计算时，f[i]代表前i个元素（即元素0~i-1）的某种性质

最长序列型动态规划

- 题目给定一个序列

- 要求找出符合条件的最长子序列

- 方法
  - 记录以每个元素i结尾的最长子序列的长度
  - 计算时，在i之前枚举子序列上一个元素是哪个

最长序列型动态规划

LintCode 76 Longest Increasing Subsequence

- 题意：
- 给定a[0], …, a[n-1]
- 找到最长的子序列$0<=i_1<i_2<…<i_K<n$, 使得$a[i_1]<a[i_2]<…<a[i_K]$，输出K
- 例子：
- 输入：[4, 2, 4, 5, 3, 7]
- 输出：4 (子序列2, 4, 5, 7)

动态规划组成部分一：确定状态

- 最后一步：对于最优的策略，一定有最后一个元素a[j]
- 第一种情况：最优策略中最长上升子序列就是{a[j]}，答案是1

- 第二种情况，子序列长度大于1，那么最优策略中a[j]前一个元素是a[i]，并且a[i] < a[j]

- 因为是最优策略，那么它选中的以a[i]结尾的上升子序列一定是最长的

子问题

- 因为不确定最优策略中a[j]前一个元素a[i]是哪个，需要枚举每个i
- 求以a[i]结尾的最长上升子序列
- 本来是求以a[j]结尾的最长上升子序列
- **化为子问题: i < j**
- 状态：设f[j] =**以a[j]结尾**的最长上升子序列的长度

动态规划组成部分二：转移方程

- f[j] =以a[j]结尾的最长上升子序列的长度

f[j] = max{ 1,   f[i] + 1| i < j and a[i] < a[j]}

以a[j]结尾的最长上升子序列的长度

情况1：子序列就是a[j]本身

情况2：以a[i]结尾的最长上升子序列的长度，加上a[j]

动态规划组成部分三：初始条件和边界情况

f[j] = max{ 1,   f[i]+1| i < j and a[i] < a[j]}

以a[j]结尾的最长上升子序列的长度

情况1：子序列就是a[j]本身

情况2：以a[i]结尾的最长上升子序列的长度，加上a[j]

- 情况2必须满足：
  - i>=0
  - a[j] > a[i], 满足单调性

- 初始条件：空

动态规划组成部分四：计算顺序

- f[j] =以a[j]结尾的最长上升子序列的长度

- 计算f[0], f[1], f[2], …, f[n-1]

- 答案是max{f[0], f[1], f[2], …, f[n-1]}

- 算法时间复杂度$O(n^2)$，空间复杂度O(n)

思考：如何做到时间复杂度O(nlogn)

LintCode 602 Russian Doll Envelopes

- 题意：
- 给定N个信封的长度和宽度
- 如果一个信封的长和宽都分别小于另一个信封的长和宽，则这个信封可以放入另一个信封
- 问最多嵌套多少个信封

- 例子：
- 输入：[[5,4],[6,4],[6,7],[2,3]]
- 输出：3 ([2,3] => [5,4] => [6,7])

最长序列型动态规划

- 可能出现一个信封A能放入信封B和信封C，但是信封B和信封C互相不能放入
- 将所有信封按照长度一维进行排序：$E_0, E_1, \ldots, E_{n-1}$
- 这样，如果信封E_i能够放入信封E_j里，一定有i<j
- 排序后，如果一个信封E_j是最外层的信封，那么它里面的第一层信封E_i一定满足i<j

动态规划组成部分一：确定状态

- 最后一步：设最优策略中最后一个信封，即最外层的信封，是E_j
- 考虑次外层信封是哪个
  – 一定是某个E_i，i < j

- 最优策略里，以E_i为最外层信封的嵌套层数也一定是最多的

子问题

- 要求以E_i为最外层信封时最多的嵌套层数
- 需要知道以E_j为最外层信封时最多的嵌套层数 (j < i)
- 子问题
- 状态：设f[i]表示以E_i为最外层信封时最多的嵌套层数

动态规划组成部分二：转移方程

- 设f[i]表示以E_i为最外层信封时最多的嵌套层数

$$f[i] = \max\{1, f[j]+1 \mid E_j \text{ 能放在} E_i \text{里}\}$$

以E_i为最外层信封时最多的嵌套层数

只用E_i这一个信封

以E_j为次外层信封时最多的嵌套层数，加上E_i

动态规划组成部分三：初始条件和边界情况

- 设f[i]表示以E_i为最外层信封时最多的嵌套层数
- $f[i] = \max\{1, f[j]+1 \mid E_j \text{ 能放在} E_i \text{里，} j<i\}$
- 无初始条件
- 需要先把所有信封按照长度排序

动态规划组成部分四：计算顺序

- f[0], f[1], …, f[N-1]
- 时间复杂度$O(N^2)$，空间复杂度$O(N)$