

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 $a[0], a[1], \dots, a[i-1]$ 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, $2+5+3=10$)

- 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： $f[i][1], f[i][2], \dots, f[i][K]$

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 $f[i][1], f[i][2], \dots, f[i][K]$



$$f[i][1] = \min\{f[i-1][2] + \text{cost}[i-1][1], f[i-1][3] + \text{cost}[i-1][1], \dots, f[i-1][K] + \text{cost}[i-1][1]\}$$

$$f[i][2] = \min\{f[i-1][1] + \text{cost}[i-1][2], f[i-1][3] + \text{cost}[i-1][2], \dots, f[i-1][K] + \text{cost}[i-1][2]\}$$

...

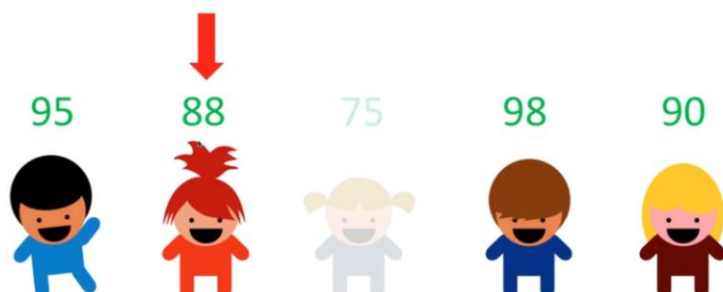
$$f[i][K] = \min\{f[i-1][1] + \text{cost}[i-1][K], f[i-1][2] + \text{cost}[i-1][K], \dots, f[i-1][K-1] + \text{cost}[i-1][K]\}$$

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 $f[i][1], f[i][2], \dots, f[i][K]$
- $f[i][j] = \min_{k \neq j} \{f[i-1][k]\} + \text{cost}[i-1][j]$
- 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？

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



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

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

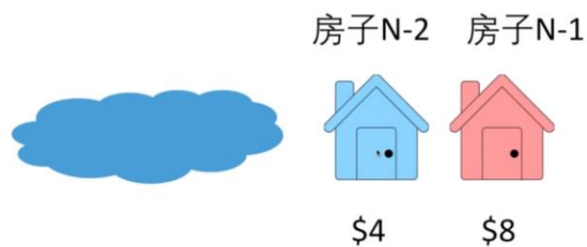


- $f[i][j] = \min_{k \neq j} \{f[i-1][k]\} + \text{cost}[i-1][j]$
- 记录下 $f[i-1][1], \dots, f[i-1][K]$ 中的最小值和次小值分别是哪个
- 假如最小值是 $f[i-1][a]$ ，次小值是 $f[i-1][b]$
- 则对于 $j=1, 2, 3, \dots, a-1, a+1, \dots, K$, $f[i][j] = f[i-1][a] + \text{cost}[i-1][j]$
- $f[i][a] = f[i-1][b] + \text{cost}[i-1][a]$
- 时间复杂度降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

- 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）

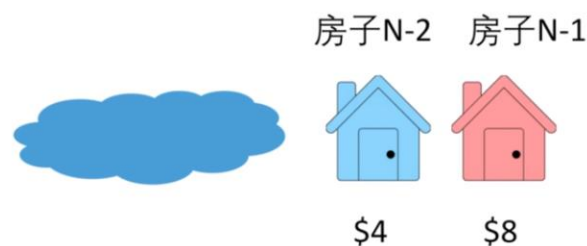
动态规划组成部分一：确定状态

- 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

九章算法

- 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



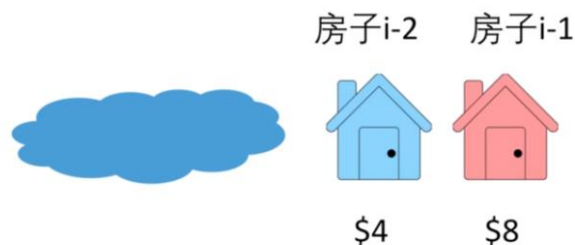
$$f[i][1] = f[i-1][0] + A[i-1]$$

偷房子 $i-1$ ，房子 $i-2$ 必须不偷

简化

九章算法

- 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

九章算法

- 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）
 - $f[1] = A[0]$
 - $f[2] = \max\{A[0], A[1]\}$

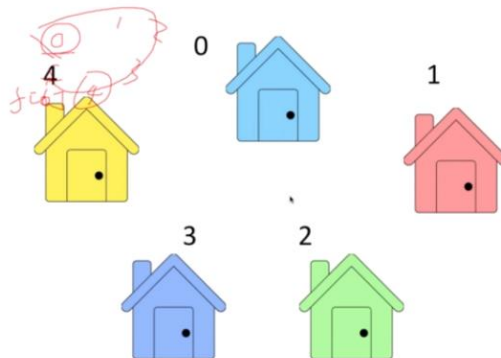
- 初始化 $f[0]$
- 计算 $f[1], f[2], \dots, f[n]$
- 答案是 $f[n]$
- 时间复杂度 $O(N)$, 空间复杂度 $O(1)$

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 $N-1$

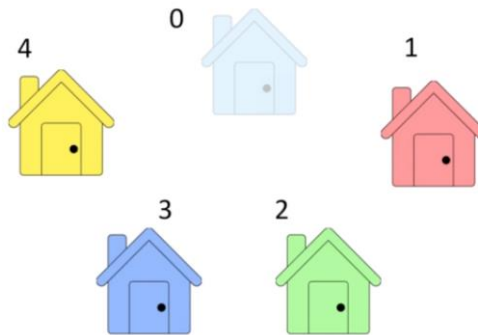


题意分析

- 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

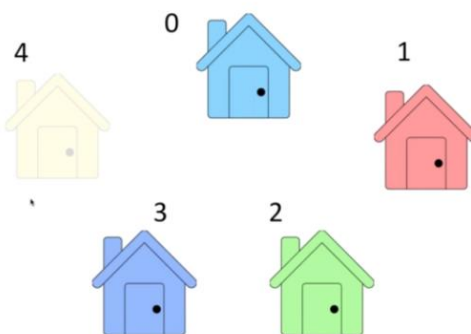


题意分析

- 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

- 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, \dots, P_{N-1}
- 可以最多买一股卖一股
- 求最大利润

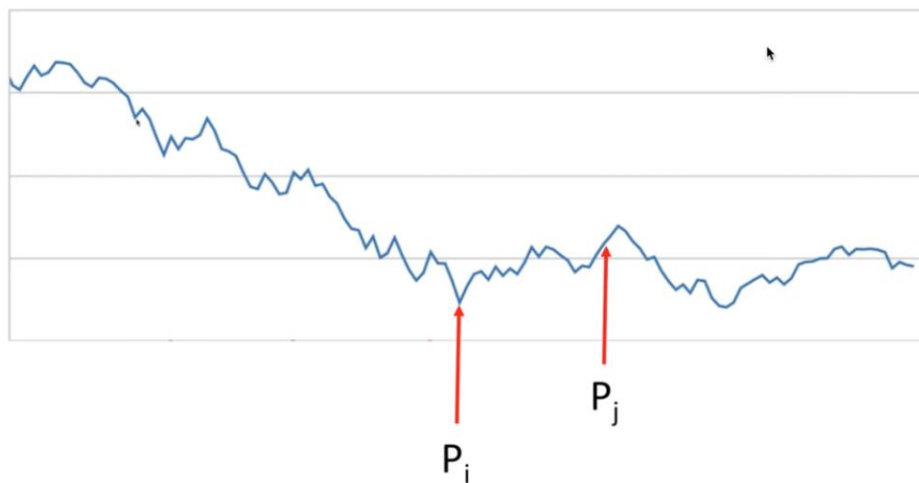
- 例子：
- 输入： $[3, 2, 3, 1, 2]$
- 输出： 1 (2买入, 3卖出)

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, \dots, 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

九章算法

- 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

九章算法

- 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

九章算法

- 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

九章算法

- 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

九章算法

- 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

$f(i)$ $f(j)$ $f(i, j)$
买过？没买过？



$P_i = \$30$

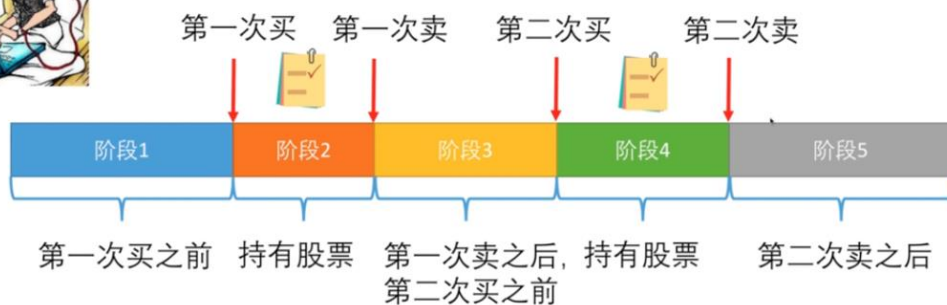


$P_j = \$50$

记录阶段

九章算法

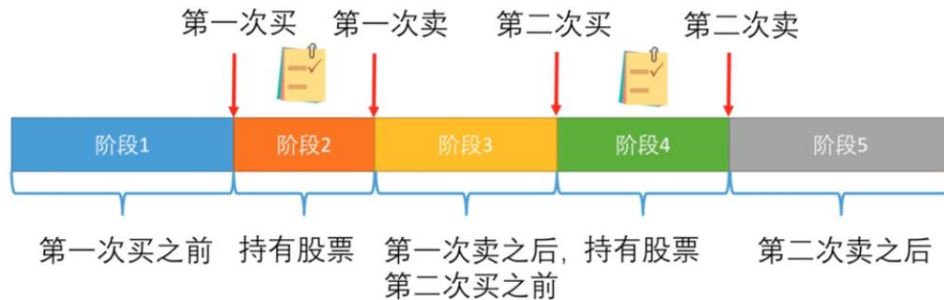
- 不知道有没有买过，就记录下来



记录阶段

九章算法

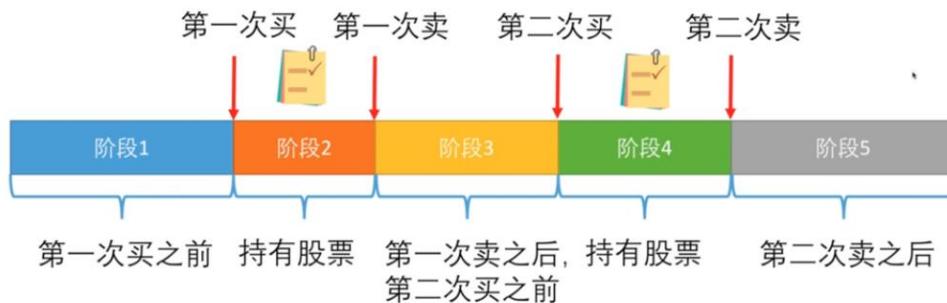
- 阶段可以保持：即不进行买卖操作
- 阶段可以变化：买或卖
 - 在阶段2，卖了一股后，进入阶段3 *第二天*
 - 在阶段2，卖了一股后当天买一股，进入阶段4



动态规划组成部分一：确定状态

九章算法

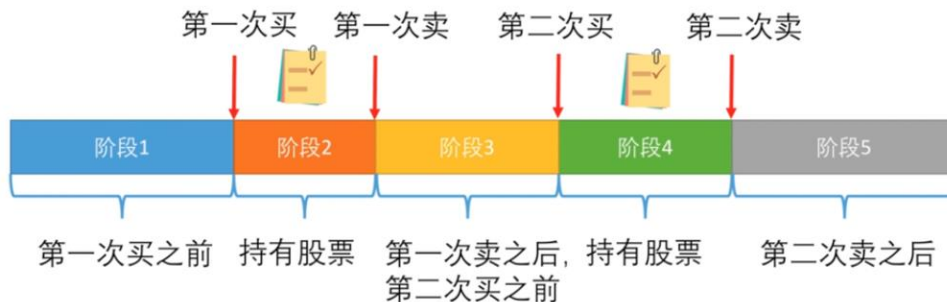
- 最优策略一定是前N天(第N-1天)结束后，处于
 - 阶段1：没买卖过
 - 阶段3：买卖过一次
 - 阶段5：买卖过两次



动态规划组成部分一：确定状态

九章算法

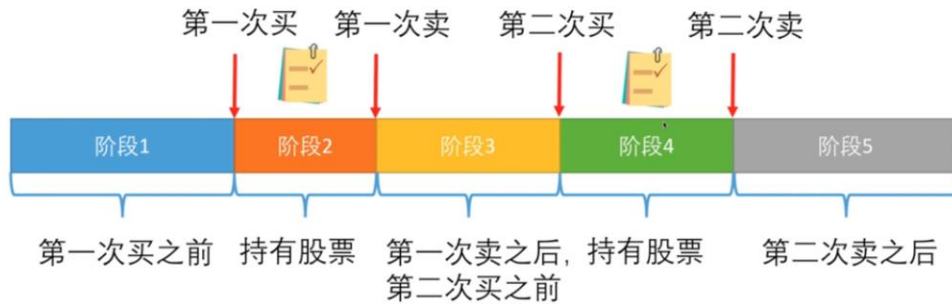
- 例如，如果要求前N天(第N-1天)结束后，在阶段5的最大获利，设为 $f[N][5]$
 - 情况1：第N-2天就在阶段5 --- $f[N-1][5]$
 - 情况2：第N-2天还在阶段4（第二次持有股票），第N-1天卖掉
 - $f[N-1][4] + (P_{N-1} - P_{N-2})$
代表承担这一天的盈亏



动态规划组成部分一：确定状态

九章算法

- 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

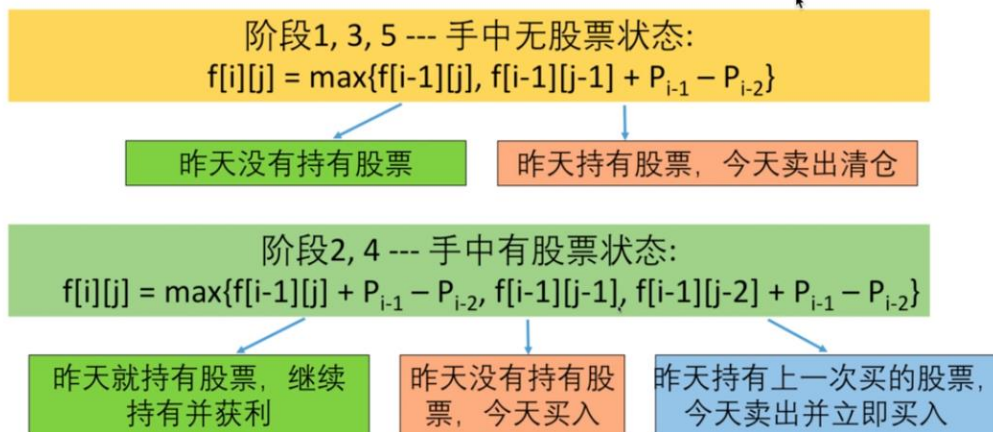
九章算法

- 要求 $f[N][1], \dots, f[N][5]$
- 需要知道 $f[N-1][1], \dots, f[N-1][5]$
- 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

九章算法

- $f[i][j]$: 前 i 天(第 $i-1$ 天)结束后，处在阶段 j ，最大获利



动态规划组成部分三：初始条件和边界情况

九章算法



- 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

九章算法

- 初始化 $f[0][1], \dots, f[0][5]$
- $f[1][1], \dots, f[1][5]$
- ...
- $f[N][1], \dots, f[N][5]$
- 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

九章算法

- 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

九章算法

- 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前一天低

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

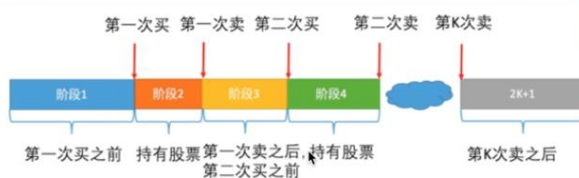
九章算法



记录阶段

九章算法

- 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

九章算法

- $f[i][j]$: 前 i 天(第 $i-1$ 天)结束后, 处在阶段 j , 最大获利



动态规划组成部分三：初始条件和边界情况

九章算法

- 刚开始 (前0天) 处于阶段1
 - $f[0][1] = 0$
 - $f[0][2] = f[0][3] = \dots = f[0][2K+1] = +\infty$
- 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 $\max\{f[N][1], f[N][3], \dots, f[N][2K+1]\}$, 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

九章算法

- 初始化 $f[0][1], \dots, f[0][2K+1]$
- $f[1][1], \dots, f[1][2K+1]$
- ...
- $f[N][1], \dots, f[N][2K+1]$
- 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



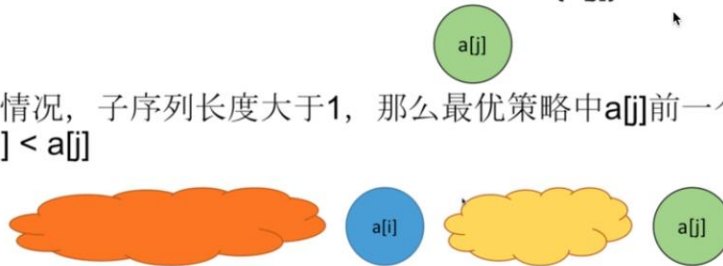
- 题目给定一个序列
- 要求找出符合条件的最长子序列
- 方法
 - 记录以每个元素i结尾的最长子序列的长度
 - 计算时，在i之前枚举子序列上一个元素是哪个

最长序列型动态规划

- 题意：
- 给定 $a[0], \dots, a[n-1]$
- 找到最长的子序列 $0 \leq i_1 < i_2 < \dots < i_k < n$, 使得 $a[i_1] < a[i_2] < \dots < a[i_k]$, 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

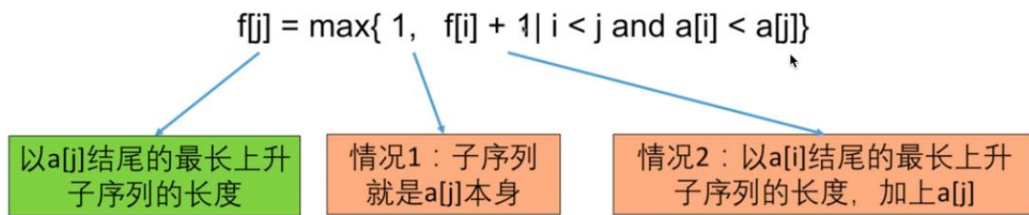
- 因为不确定最优策略中 $a[j]$ 前一个元素 $a[i]$ 是哪个，需要枚举每个 i
- 求以 $a[j]$ 结尾的最长上升子序列
- 本来是求以 $a[j]$ 结尾的最长上升子序列
- 化为子问题: $i < j$
- 状态：设 $f[j]$ = 以 $a[j]$ 结尾的最长上升子序列的长度



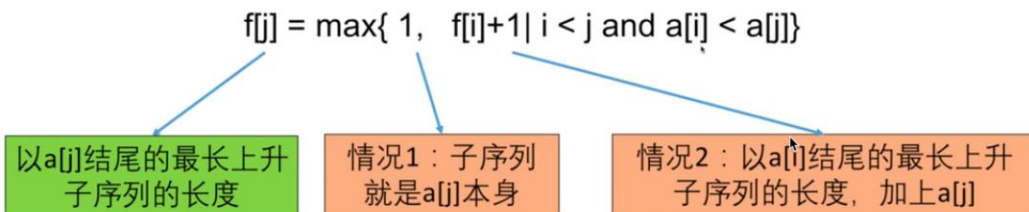
动态规划组成部分二：转移方程



- $f[j]$ = 以 $a[j]$ 结尾的最长上升子序列的长度



动态规划组成部分三：初始条件和边界情况



- 情况2必须满足：
 - $i \geq 0$
 - $a[i] < a[j]$, 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 $f[0], f[1], f[2], \dots, f[n-1]$
- 答案是 $\max\{f[0], f[1], f[2], \dots, f[n-1]\}$
- 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 $O(n \log n)$

LintCode 602 Russian Doll Envelopes

九章算法

- 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

九章算法

- 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, \dots, 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

九章算法

- 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

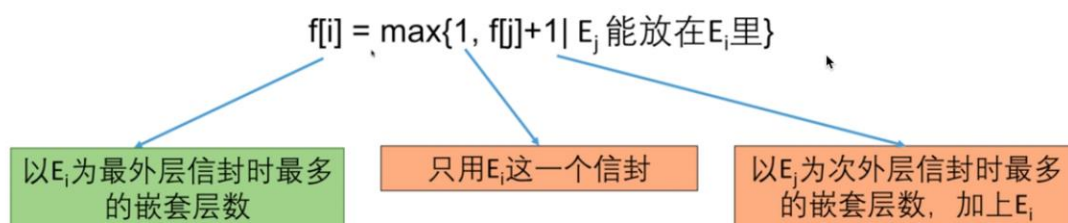
九章算法

- 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



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



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