二分法

oooooooooooooxxxxxxxxxx

找最后一个o 找第一个x

动态规划

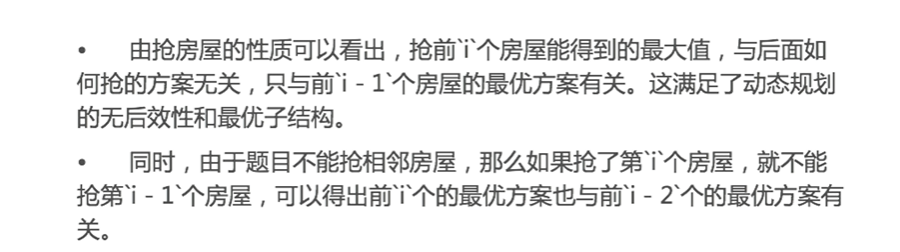
## 一道例题

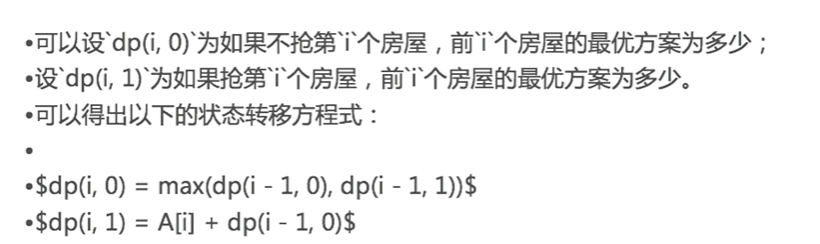
求最值

求可行性

方案数

打家劫舍





1 数据有一定不可交换性 方向性

2 最大和

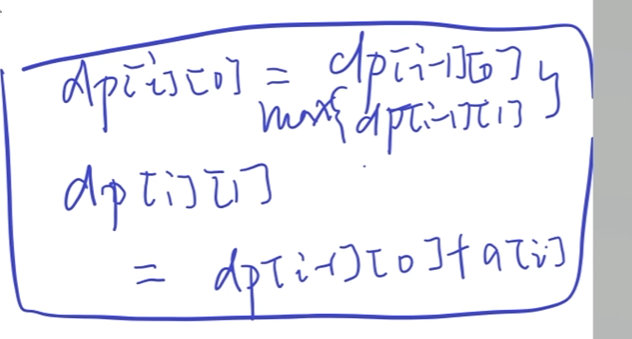
坐标动态规划

dp[坐标] = 行走到这个位置的最优值

上一个坐标是哪里来的

dp[i][0] 不打劫

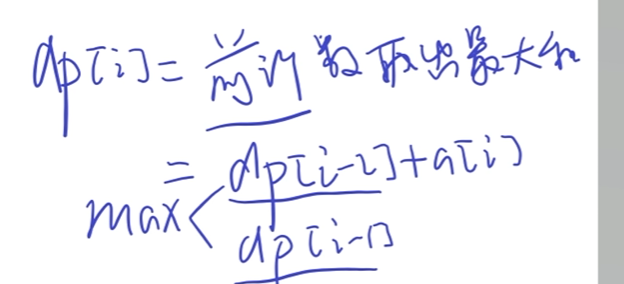
dp[i][1] 打劫



前缀动态规划

前i个数怎么样

dp[i] 前i个数取出来的最大值，不关心 是否打劫



## Triangle

遍历法 全局最优 有一个全局变量

dfs找所有的方法

自底向上：

public int minimumTotal(int[][] triangle) {

// 自底向上

int[][] dp = new int[triangle.length][triangle.length];

int n = triangle.length;

// 初始化

for(int i=0;i<triangle.length;i++){

for(int j=0;j<triangle.length;j++){

dp[i][j] = 0;

}

}

// 最后面一行要初始化

for(int i=0;i<n;i++){

dp[n-1][i] = triangle[n-1][i];

}

// 循环递推求解

for(int i=n-2;i>=0;i--){

for(int j=0;j<=i;j++){

dp[i][j] = Math.min(dp[i+1][j], dp[i+1][j+1]) + triangle[i][j];

}

}

return dp[0][0];

}

top-down

public class Solution {

/\*\*

\* @param triangle: a list of lists of integers

\* @return: An integer, minimum path sum

\*/

public int minimumTotal(int[][] triangle) {

// 自顶向下

int[][] dp = new int[triangle.length][triangle.length];

int n = triangle.length;

// 先初始化三角形的左边和右边

for(int i=0;i<triangle.length;i++){

for(int j=0;j<triangle.length;j++){

if(i==0&&j==0) dp[i][j] = triangle[i][j];

else dp[i][j] = 0;

}

}

for(int i=1;i<n;i++){

dp[i][0] = dp[i-1][0]+triangle[i][0];

dp[i][i] = dp[i-1][i-1]+triangle[i][i];

}

// top-down

for(int i=2;i<n;i++){

for(int j=1;j<i;j++){

dp[i][j] = triangle[i][j] + Math.min(dp[i-1][j], dp[i-1][j-1]);

}

}

int min = Integer.MAX\_VALUE;

for(int i=0;i<n;i++){

min = Math.min(min, dp[n-1][i]);

}

return min;

}

}

## 题目特征

求最大值 最小值 bfs再简单图可以求最短路径

判断是否可行

统计方案个数

什么时候不能用动态规划

• 求出所有 具体 的方案而非方案 个数

• http://www.lintcode.com/problem/palindrome-partitioning/

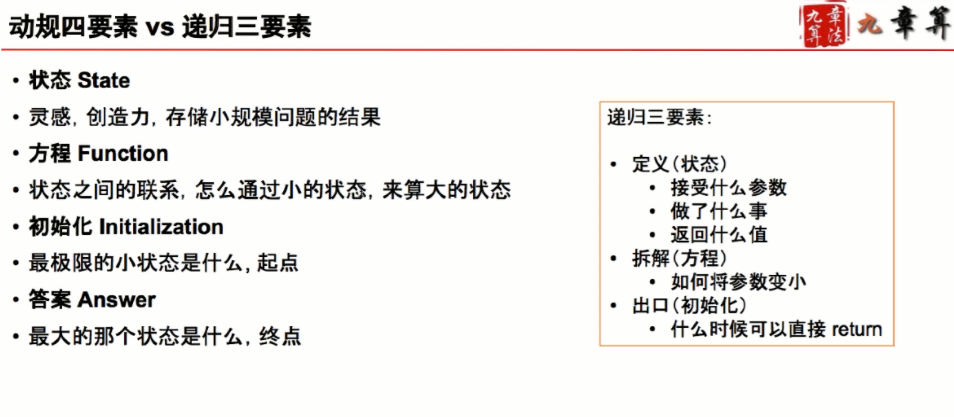
• 输入数据是一个 集合 而不是 序列

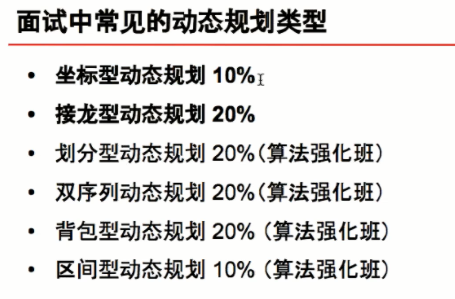
• http://www.lintcode.com/problem/longest-consecutive-sequence/

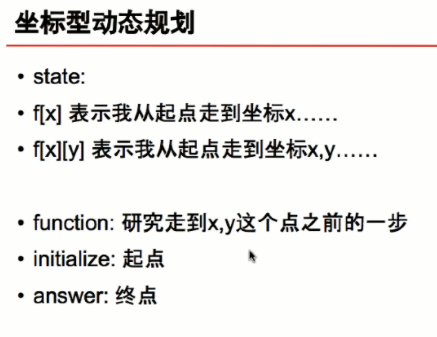
• 暴力算法的复杂度已经是多项式级别

• 动态规划擅长与优化指数级别复杂度(2^n,n!)到多项式级别复杂度(n^2,n^3) • 不擅长优化n^3到n^2

• 则 极不可能 使用动态规划求解







• state: f[x][y]从起点到x,y的路径数

• function: (研究倒数第一步) f[x][y] = f[x - 1][y] + f[x][y - 1]

initialize: f[0][i] = 1

• f[i][0] = 1

answer: f[n-1][m-1]

Related Question: Unique Path II

• Unique Path

state: f[x][y]从起点走到x,y的最短路径

• function: f[x][y] = min(f[x-1][y], f[x][y-1]) + A[x][y]

intialize: f[i][0] = sum(0,0 ~ i,0)

• f[0][i] = sum(0,0 ~ 0,i)

answer: f[n-1][m-1]

Climbing Stairs

• state: f[i]表示跳到第i个位置的方案总数

function: f[i] = f[i-1] + f[i-2]

initialize: f[0] = 1

answer: f[n] // index from 0~n

跳跃游戏 I && II http://www.lintcode.com/en/problem/jump-game/ http://www.lintcode.com/en/problem/jump-game-ii/ 这个题最优的方法是使用“贪心法”，动态规划复杂度较高

我们已与LintCode协商，将数据范围调低，让你也可以通过动态规划算法拿到Accepted。

## 接龙型动态规划

最长子序列

将n个数看做n个木桩，目的是从某个木桩出发，从前向后，从低往高，看做多能踩多少个木桩。

• state: f[i] 表示（从任意某个木桩）跳到第i个木桩，最多踩过多少根木桩

function: f[i] = max{f[j] + 1}, j必须满足 j < i && nums[j] < nums[i]

initialize: f[0..n-1] = 1

answer: max{f[0..n-1]}

Perfect Squares

我们用f[i](https://www.jiuzhang.com/solutions/perfect-squares/)表示i最少能被几个完全平方数来表示。

* 首先我们对dp数组赋予初值，对于每个完全平方数的f=1。
* 利用记忆化搜索来完成查找。对于i，我们考虑i的前继状态，也就是哪几个状态可以加上一个完全平方数抵达i。
* 对于所有能够抵达i的状态，取他们的最小值+1即可。

上述过程的状态转移方程为 f[i](https://www.jiuzhang.com/solutions/perfect-squares/) = min{f[i](https://www.jiuzhang.com/solutions/perfect-squares/), f[i - j \* j] + 1} (j\*j <= i)

边界: f[i\*i] = 1

http://www.lintcode.com/en/problem/perfect-squares/

http://www.jiuzhang.com/solutions/perfect

squares/

动态规划的实质

•

记忆化搜索

•

避免重复的中间结果计算

•

动态规划与递归的关系

•

动规四要素 vs 递归三要素

•

什么时候使用动态规划

•

最优，可行，方案数

•

什么时候不用动态规划

•

所有方案而不是方案数

•

集合而非序列

•

暴力算法已经是多项式级别复杂度

•

动态规划擅长优化指数级别(2

^

n)到多项式级别(n

^

•

动态规划四要素

•

状态，方程，初始化，答案

•

动态规划经典题

•

Longest Increasing Subsequence (LIS

动态规划题目特点

1 计数

有多少种方式走到右下角

有多少种方式选出k个数使得和是sum

2 求最大值最小值

从左上角走到右下角路径最大的数字和

最长上升子序列长度

3 求存在性

取石子游戏 先手是否必胜

能不能选出k个数使得和是sum

## 换硬币问题

要保证之前的问题是最优解

f(x) 用最少的硬币数拼出规模为x

最后的硬币只可能是2 5 7

如果最后的硬币是2 f(27) = f(27-2)+1

….类推

f(27) = min(….)

## 最大乘积 连续子序列

## 动态规划

子问题

转移方程

初始条件和边界情况

计算顺序

编程

## 坐标型动态规划

最简单的动态规划

给定一个序列或网格

需要找到序列中某个/些子序列或网格中的某条路径

某种性质最大/最小

计数

存在性

动态规划方程中f[i]中的下标i 表示以ai为结尾的满足条件的子序列的性质，

f[i][j]中的下标[i][j]表示以格子(i,j)为结尾的满足条件的路径的性质

最大值/最小值

个数

是否存在

坐标型动态规划的初始条件f[0]就是指以a0为结尾的子序列的性质

坐标类型动态规划可以用滚动数组进行优化

## 位操作类型动态规划

<https://www.lintcode.com/problem/counting-bits/description>

常见的位操作有：

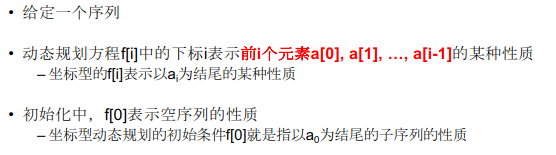
& 与 | 或 ^异或 !非

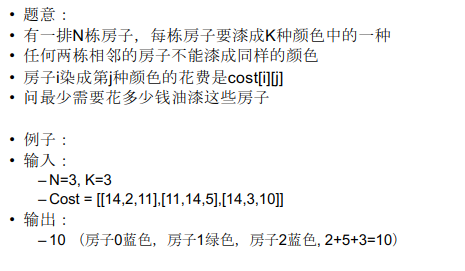
逐位操作

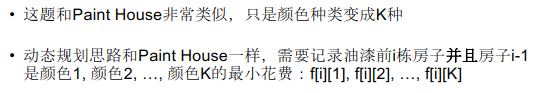
和位操作相关的动态规划一般用值作状态

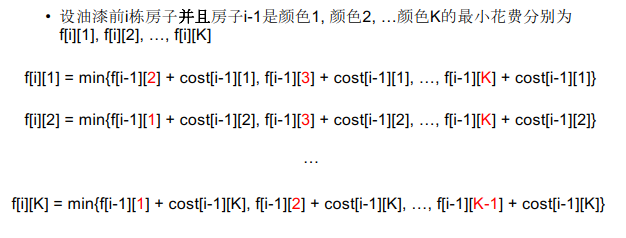
## 序列动态规划

<https://www.lintcode.com/problem/paint-house-ii/description>

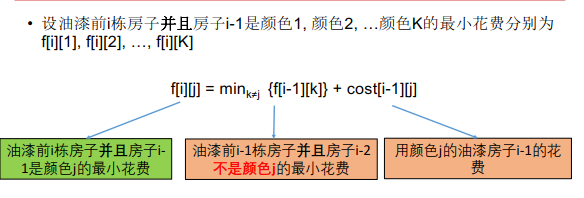


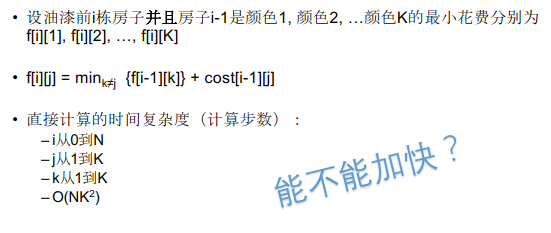






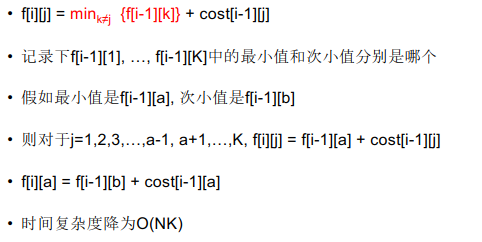
转移方程：





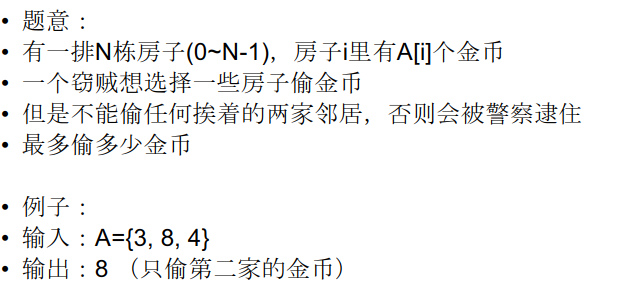
滚动数组优化



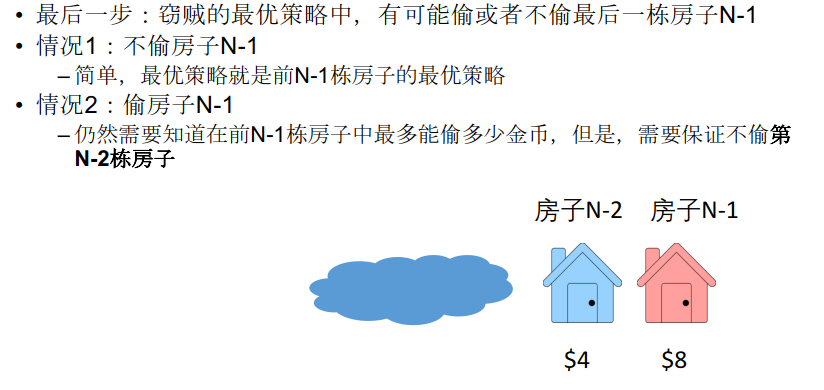


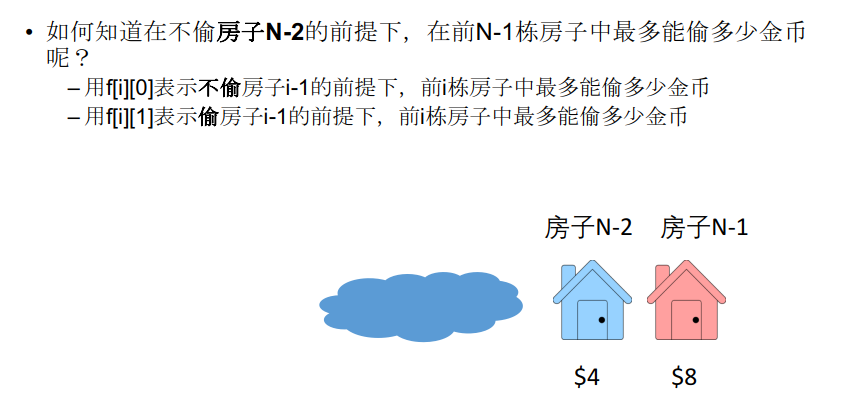
打家劫舍

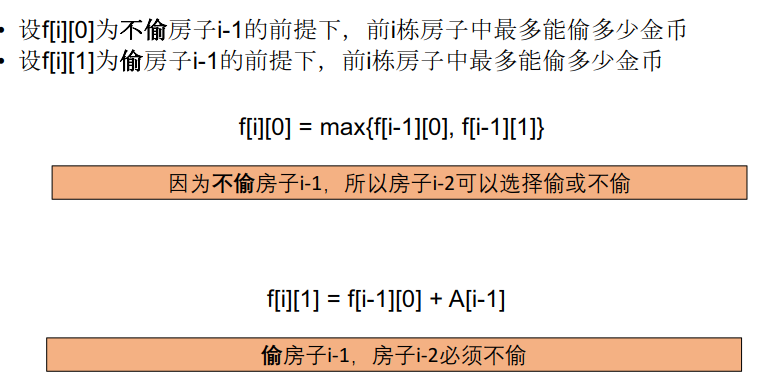
<https://www.lintcode.com/problem/house-robber/>



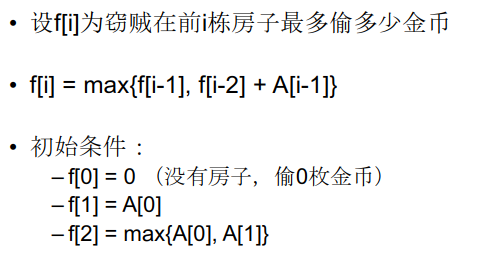
确定状态



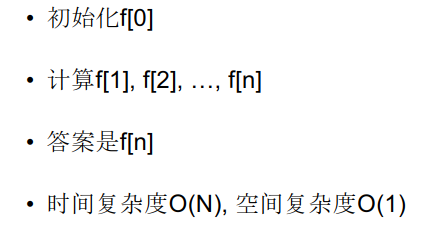




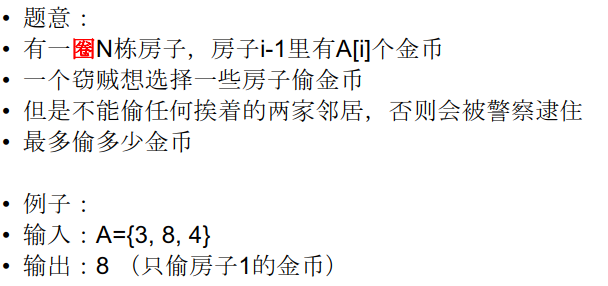
初始条件和边界情况

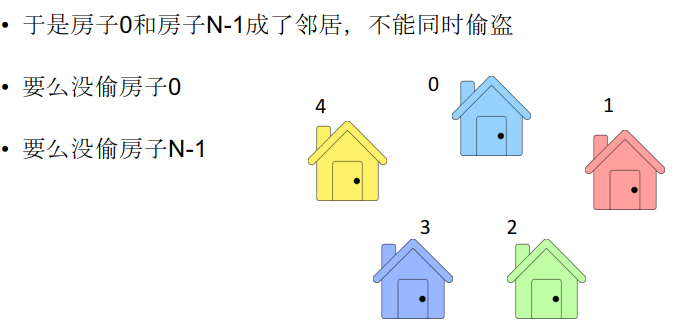


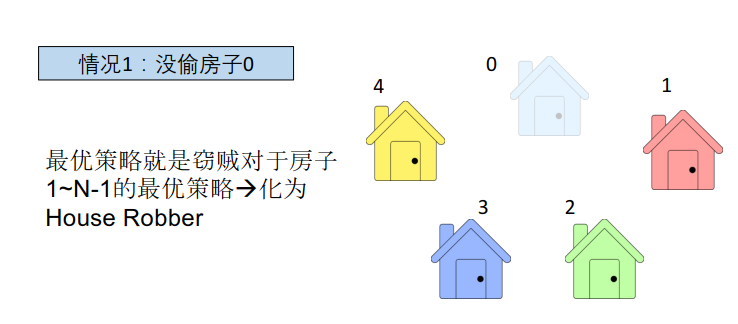
计算顺序

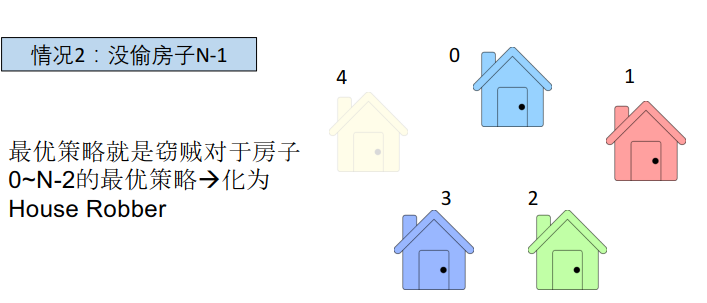


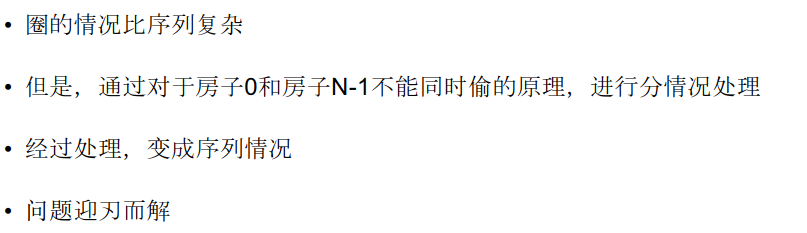
<https://www.lintcode.com/problem/house-robber-ii/>





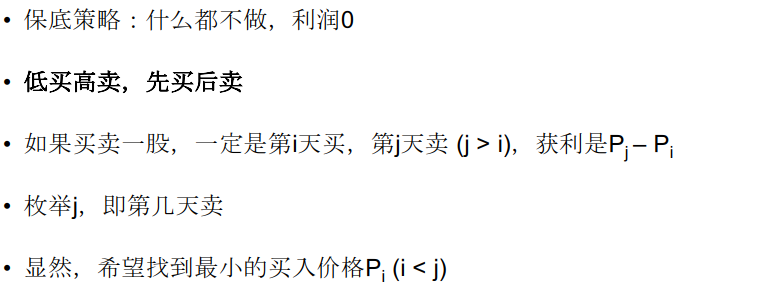


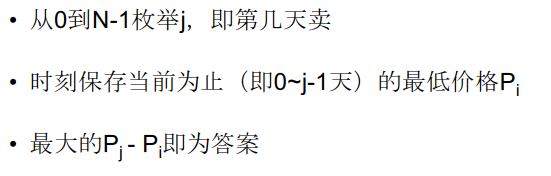




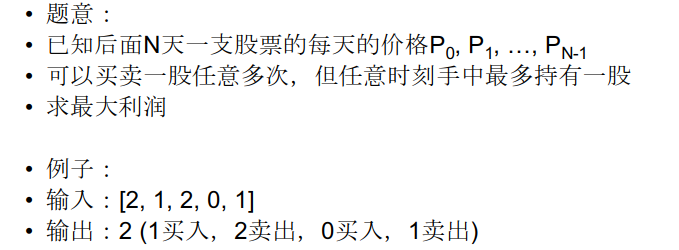
股票问题是重点！！！

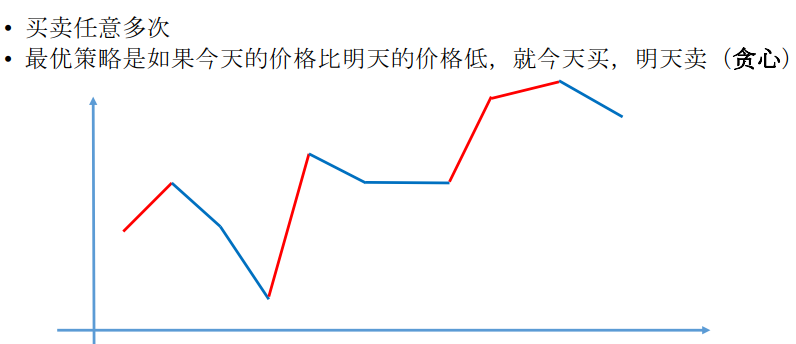
<https://www.lintcode.com/en/problem/best-time-to-buy-and-sell-stock/>

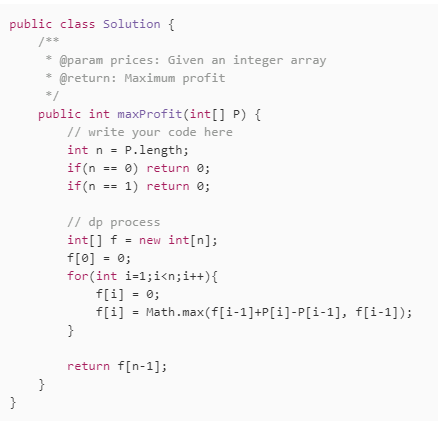




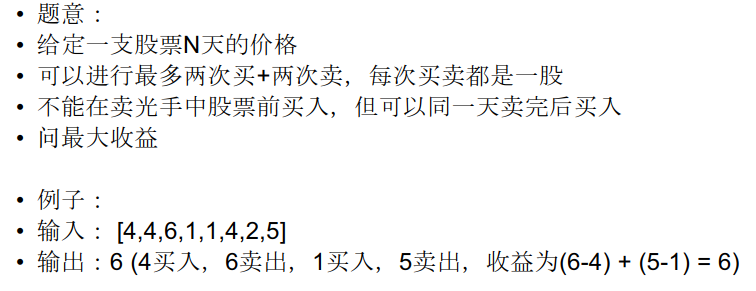
<https://www.lintcode.com/en/problem/best-time-to-buy-and-sell-stock-ii/>

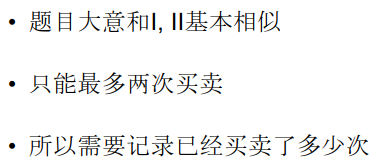


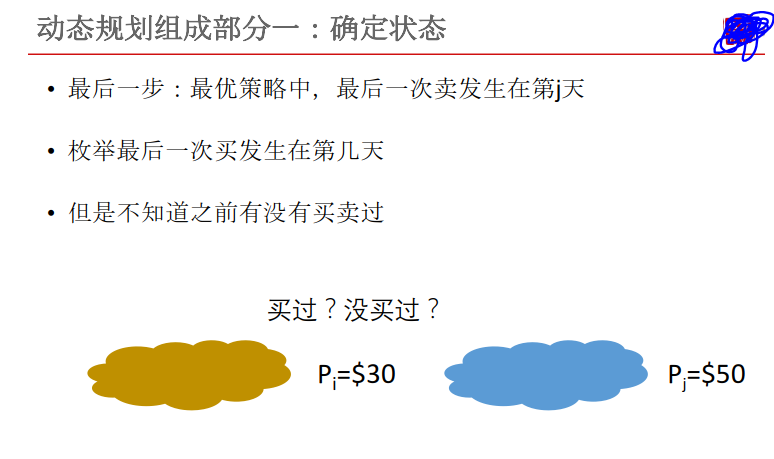


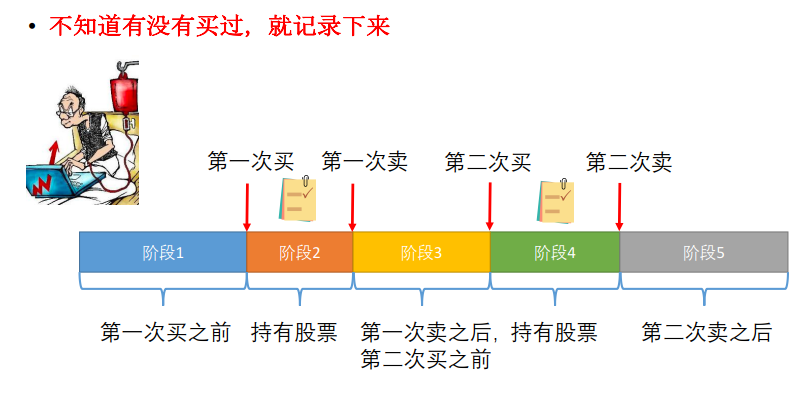


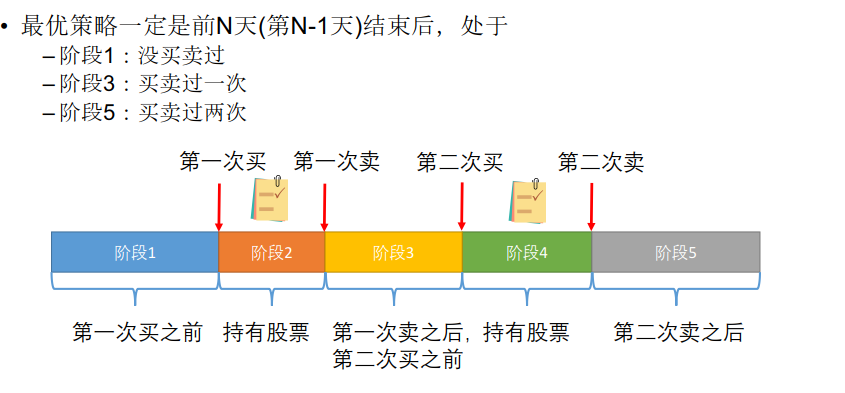
<https://www.lintcode.com/problem/best-time-to-buy-and-sell-stock-iii/>

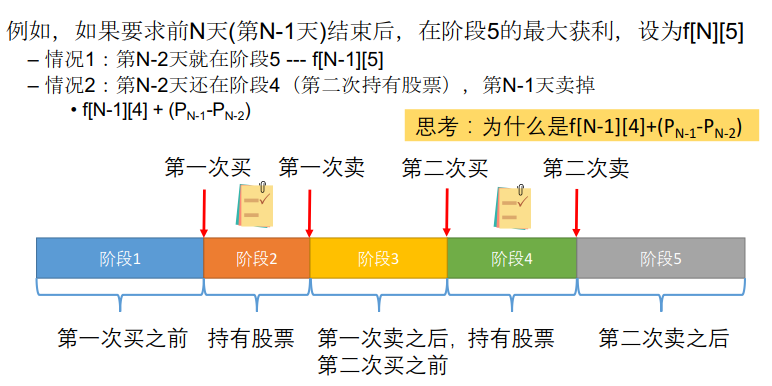


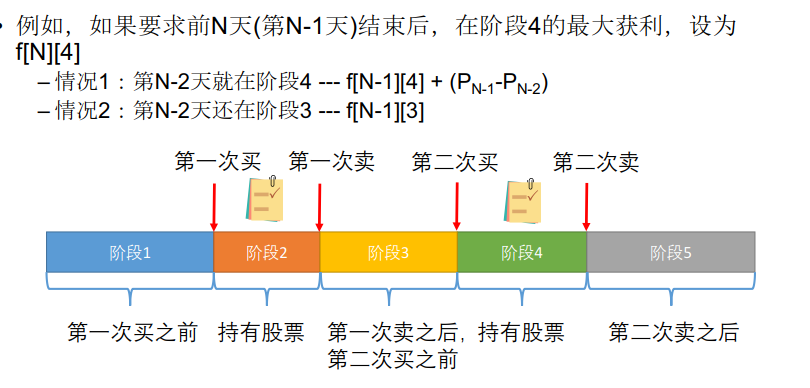


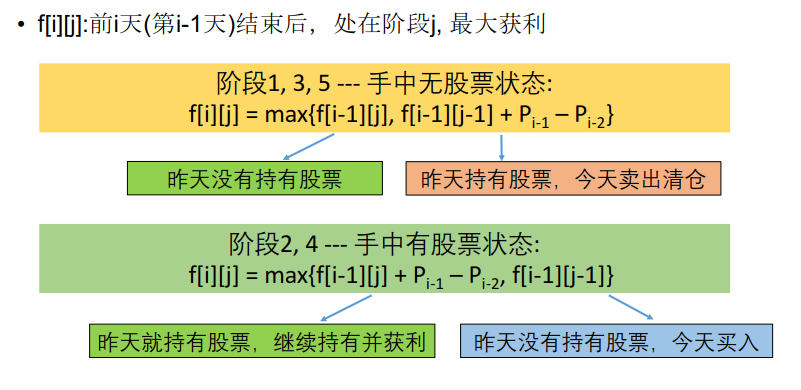
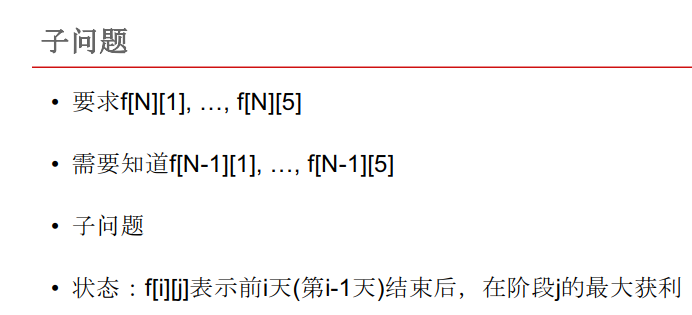




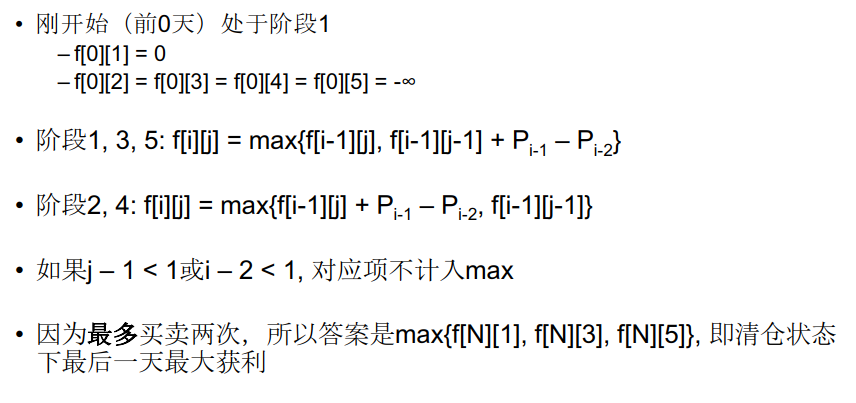




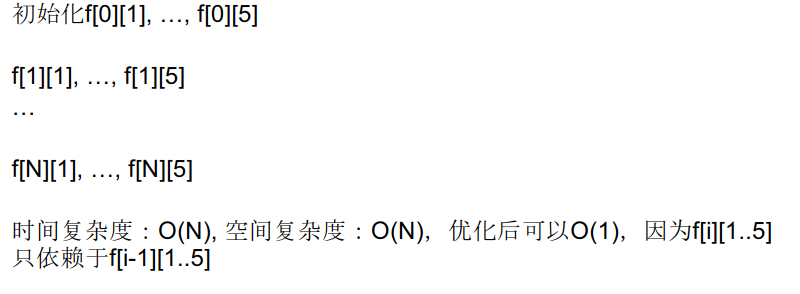




初始条件和边界情况

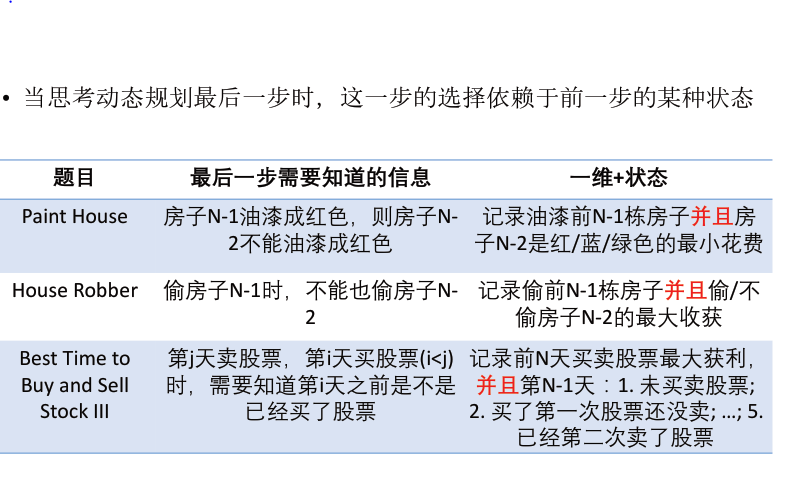


计算顺序





k次



DFS

N皇后问题可以看作dfs

2413 每个皇后排在第几列

最短变化序列规则

简单图 最短路径 拓扑排序