

动态规划专题 第七讲

侯卫东



扫描二维码关注微信/微博
获取最新面试题及权威解答

微信: [ninechapter](#)

知乎专栏: <http://zhuanlan.zhihu.com/jiuzhang>

微博: <http://www.weibo.com/ninechapter>

官网: www.jiuzhang.com

- LintCode上较难的动态规划题目
- 综合型动态规划
- 需要辅助数据结构/算法 (字母树, 哈希表, 二分查找) 的动态规划
- 万变不离其宗

Longest Increasing Subsequence

<http://www.lintcode.com/problem/longest-increasing-subsequence/>

<http://www.jiuzhang.com/solutions/longest-increasing-subsequence/>

LintCode 76 Longest Increasing Subsequence



- 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)

题目分析

- 之前课上分析过
- 最长序列型动态规划
- $f[j]$ = 以 $a[j]$ 结尾的最长上升子序列的长度
- 转移方程: $f[j] = \max\{1, f[i] + 1 \mid i < j \text{ and } a[i] < a[j]\}$
- 时间复杂度 $O(N^2)$
- 能不能继续优化

分析方程f的值

- 转移方程: $f[j] = \max\{1, f[i] + 1 \mid i < j \text{ and } a[i] < a[j]\}$
- 每个 $f[j]$ 都在寻找前面比自己小的 $a[i]$ 里, 最大的 $f[i]$

	0	1	2	3	4	5	6	7
a	5	1	2	7	6	3	2	10
f	1	1	2	3	3	3	2	4

分析方程f的值

- 转移方程: $f[j] = \max\{1, f[i] + 1 \mid i < j \text{ and } a[i] < a[j]\}$
- 每个 $f[j]$ 都在寻找前面比自己小的 $a[i]$ 里, 最大的 $f[i]$

	0	1	2	3	4	5	6	7
a	5	1	2	7	6	3	2	10
f	1	1	?					

分析方程f的值

- 转移方程: $f[j] = \max\{1, f[i] + 1 \mid i < j \text{ and } a[i] < a[j]\}$
- 每个 $f[j]$ 都在寻找前面比自己小的 $a[i]$ 里, 最大的 $f[i]$
- $a[0]$ 和 $f[0]$ 已经沒有用, 因为 $f[1]$ 和 $f[0]$ 一样大, $a[1]$ 还比 $a[0]$ 小

	0	1	2	3	4	5	6	7
a	5	1	2	7	6	3	2	10
f	1	1	?					

分析方程f的值

- 转移方程: $f[j] = \max\{1, f[i] + 1 \mid i < j \text{ and } a[i] < a[j]\}$
- 每个 $f[j]$ 都在寻找前面比自己小的 $a[i]$ 里, 最大的 $f[i]$
- $a[4]$ 和 $f[4]$ 已经沒有用, 因为 $f[5]$ 和 $f[4]$ 一样大, $a[5]$ 还比 $a[4]$ 小

	0	1	2	3	4	5	6	7
a	5	1	2	7	6	3	2	10
f	1	1	2	3	3	3	2	?

优化要点

- 对于每个f值: 1, 2, ..., 记录拥有这个f值的**最小的**a[i]
 - $f[1] = 1, a[1] = 1$
 - $f[6] = 2, a[6] = 2$
 - $f[5] = 3, a[5] = 3$

	0	1	2	3	4	5	6	7
a	5	1	2	7	6	3	2	10
f	1	1	2	3	3	3	2	?

优化要点

- 对于每个 f 值: $1, 2, \dots$, 记录拥有这个 f 值的**最小的** $a[i]$
 - $f[1] = 1, a[1] = 1$
 - $f[6] = 2, a[6] = 2$
 - $f[5] = 3, a[5] = 3$
- 这个序列($a[1]=1, a[6]=2, a[5]=3$)中, 一定是每个数都比下一个小
- 一个新的数 $a[j]$ 来了, 它的 f 值很好算: 在序列($a[1]=1, a[6]=2, a[5]=3$)中找到比它小的数中对应 f 值最大的那个 $f[i]$, $f[j] = f[i] + 1$
 - $a[j]=10$, 找到 $a[5] = 3$, 所以 $f[j] = 3 + 1 = 4$
 - $a[j]=2$, 找到 $a[1] = 1$, 所以 $f[j] = 1 + 1 = 2$
- 然后用 $a[j]$ 替换序列中的下一个, 因为 $f[j]$ 和它值一样, 但 $f[j]$ 更小

思考：为什么

二分查找优化

- 在序列($a[1]=1, a[6]=2, a[5]=3$)中找到比它小的数中对应 f 值最大的那个
 $f[i], f[j] = f[i] + 1$
- 而序列永远是单调增的
- 所以可以二分查找
- 序列长度 $\leq N$, 因为最长上升子序列长度 $\leq N$
- 每次查找时间复杂度 $O(\log_2 N)$
- 总的时间复杂度 $O(N \log_2 N)$

K Edit Distance

<http://www.lintcode.com/problem/k-edit-distance/>
<https://www.jiuzhang.com/solutions/k-edit-distance/>

LintCode 623 K Edit Distance

- 题意：
- 给定n个字符串，以及目标字符串Target
- 问哪些字符串和Target的编辑距离不大于K
- 一次编辑包括插入一个字符或删除一个字符或修改一个字符

- 例子：
- 输入：
 - A = ["abc", "abd", "abcd", "adc"]
 - Target = "ac"
 - K = 1
- 输出： ["abc", "adc"]

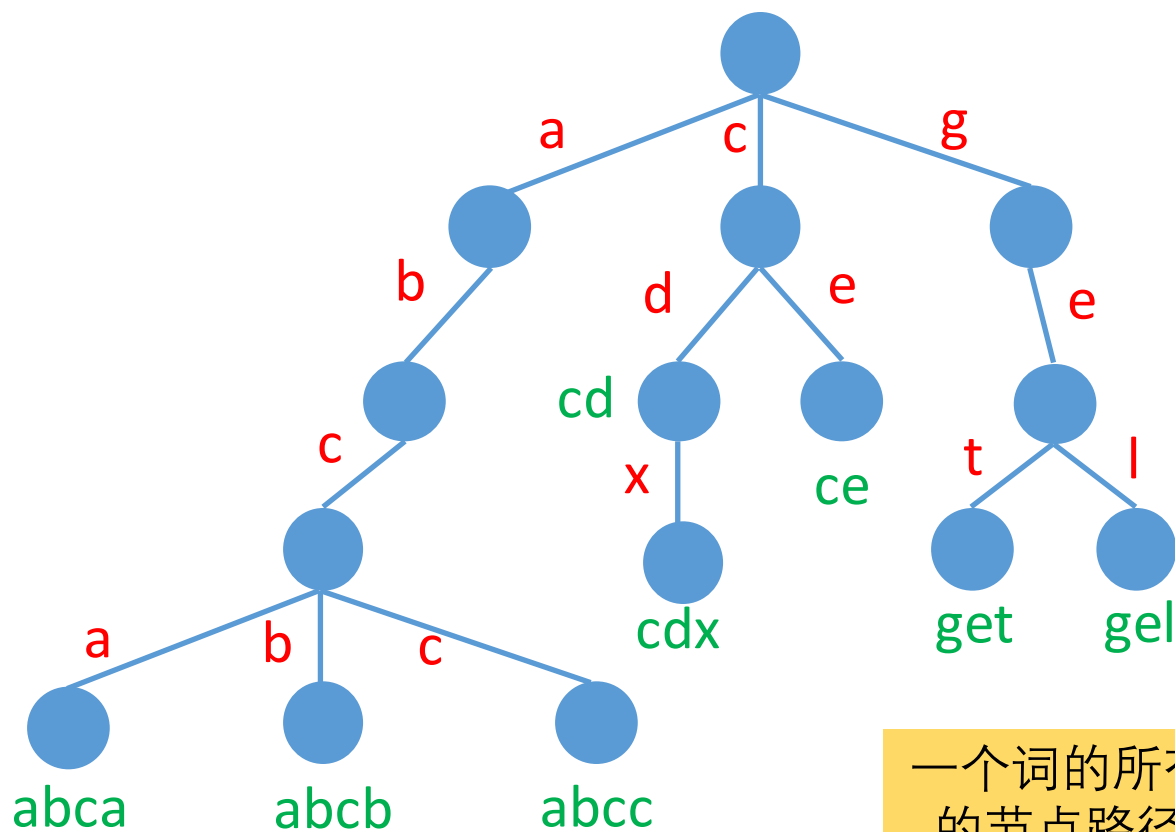
题目分析

- 这题和Edit Distance非常类似，只是要求多个字符串和Target的最小编辑距离
- 可以依次求每个字符串s和Target的最小编辑距离
 - 设 $f[i][j]$ 为s前i个字符 $s[0..i-1]$ 和Target前j个字符 $Target[0..j-1]$ 的最小编辑距离
- 存在重复计算
 - 如果给定的字符串是"abca", "abcb", "abcc"
 - 三个字符串的前3个字符都一样
 - "abcda"前0~3个字符和Target前0~n个字符的最小编辑距离
 - "abcdb"前0~3个字符和Target前0~n个字符的最小编辑距离
 - "abcdc"前0~3个字符和Target前0~n个字符的最小编辑距离

重复计算了3次

- 如何避免重复计算
- 如果几个字符串共享一段前缀，他们对应的 $f[i][j]$ 可以共享，即只计算一次
- 如何知道哪些字符串共享前缀？如何共享 $f[i][j]$ ？
- 数据结构Trie：字母树

字母树

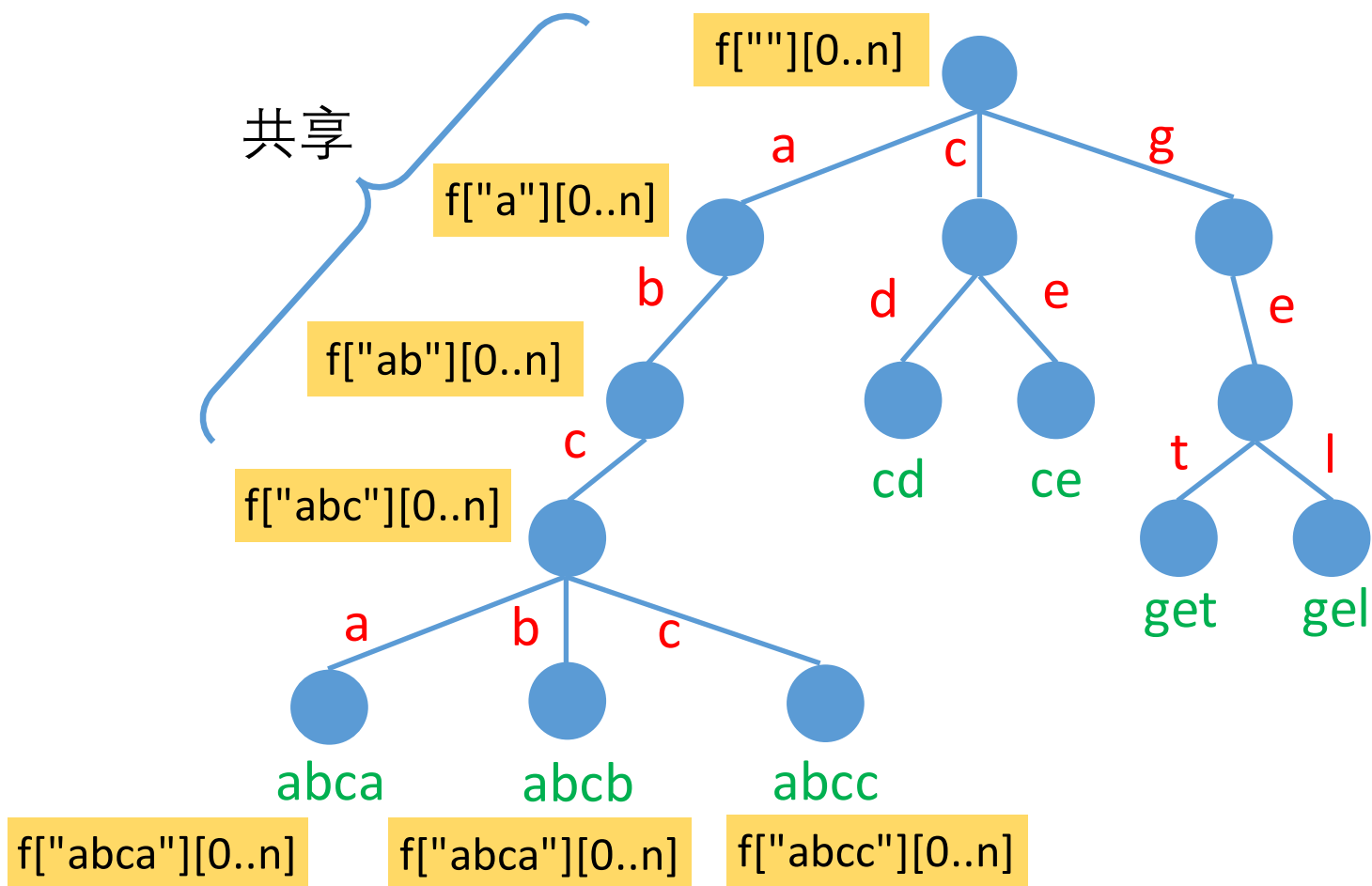


一个词的所有前缀就是从根到这个词的节点路径上形成的所有的字符串

动态规划组成部分一：确定状态

- 在Edit Distance一题中，状态是 $f[i][j]$ 为A前i个字符 $A[0..i-1]$ 和Target前j个字符 $Target[0..j-1]$ 的最小编辑距离
 - 设Target长度是n
 - A每个前缀和Target所有前缀的最小编辑距离是：
 - $f[0][0] \sim f[0][n]$
 - $f[1][0] \sim f[1][n]$
 - $f[2][0] \sim f[2][n]$
 - ...
- 现在，因为有多字符串 A_1, A_2, \dots ，我们可以将用 $f[\text{前缀}][j]$ 表示一个前缀和Target前j个字符的最小编辑距离

动态规划组成部分一：确定状态



动态规划组成部分二：转移方程

- 设 $f[s_p][j]$ 为前缀 s_p (即节点P对应的字符串)和Target前j个字符Target[0..j-1]的最小编辑距离
- 设P的父亲是Q

$$f[s_p][j] = \min\{f[s_p][j-1]+1, f[s_q][j-1]+1, f[s_q][j]+1, f[s_q][j-1] | s_p[\text{last}] = \text{Target}[j-1]\}$$

情况一： s_p 在最后插入Target[j-1]

情况二： s_p 最后一个字符替换成Target[j-1]

情况三： s_p 删掉最后一个字符

情况四： s_p 和Target最后一个字符相等

动态规划组成部分三：初始条件和边界情况

- 设 $f[s_p][j]$ 为前缀 s_p (即节点 P 对应的字符串)和 $Target$ 前 j 个字符 $Target[0..j-1]$ 的最小编辑距离
- 初始条件：一个空串和一个长度为 L 的串的最小编辑距离是 L
 - $f[s_{root}[j] = f[""][j] = j$ ($j = 0, 1, 2, \dots, n$)
 - $f[s_p][0] = \text{length}(s_p)$

动态规划组成部分四：计算顺序



- 初始化 $f[s_{\text{root}}][0] \sim f[s_{\text{root}}][n]$
- 按照字母树深度优先搜索顺序计算每个 $f[s_p][0] \sim f[s_p][n]$
- 答案是满足 $f[s_p][n] \leq K$ 且 s_p 为一个给定的单词的节点 P 的个数
- 时间复杂度（计算步数） $O(\text{前缀个数} * N)$ ，空间复杂度（数组大小） $O(\text{前缀个数} * N)$

- 在序列+状态型动态规划中，如果状态数过多，直接开数组会空间过大
- 在实际操作中，可以用哈希表来存储可能达到的状态
- 节省空间

Frog Jump

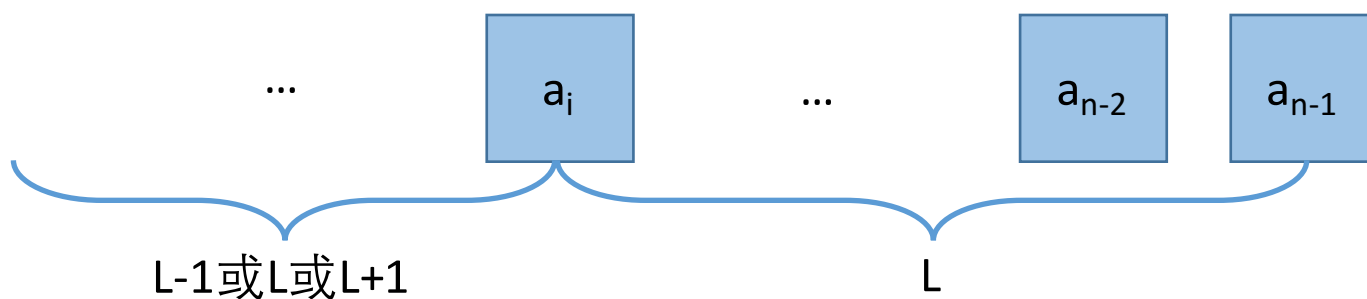
<http://www.lintcode.com/problem/frog-jump/>
<http://www.jiuzhang.com/solutions/frog-jump/>

LintCode 622 Frog Jump

- 题意：
 - 有一条小河上有N个石头，位置依次在 $a_0 < a_1 < \dots < a_{n-1}$
 - 有一只青蛙在第一个石头上
 - 青蛙一开始可以向右跳距离为1
 - 它必须一致向右跳，并且落在石头上
 - 如果上次跳的距离是L，这次跳的距离可以是L-1, L或者L
 - 问能否到达最后一个石头
-
- 例子：
 - 输入： [0,1,3,5,6,8,12,17]
 - 输出： true (0→1→3→5→8→12→17)

动态规划组成部分一：确定状态

- 最后一步：如果可以跳到最后一个石头 a_{n-1} ，考虑最后跳的一步 L
- 青蛙一定是从某块石头 $a_i = a_{n-1} - L$ 跳过来的
- 所以考虑是否能跳到 a_i
- 但是倒数第二跳只能是 $L-1, L$ 或者 $L+1$



子问题

- 要求是否能最后一跳 L 跳到最后一个石头 a_{n-1}
- 需要知道能否最后一跳 $L-1, L$ 或者 L 跳到石头 $a_i = a_{n-1} - L$
- 子问题
- 状态：设 $f[i][j]$ 表示是否能最后一跳 j 跳到石头 a_i
- 坐标+状态型动态规划

动态规划组成部分二：转移方程

- 设 $f[i][j]$ 表示是否能最后一跳 j 跳到石头 a_i
- 设上一块石头是 $a_k = a_i - j$ ，可以通过一个哈希表 ($a_k \rightarrow k$) 快速找到 k

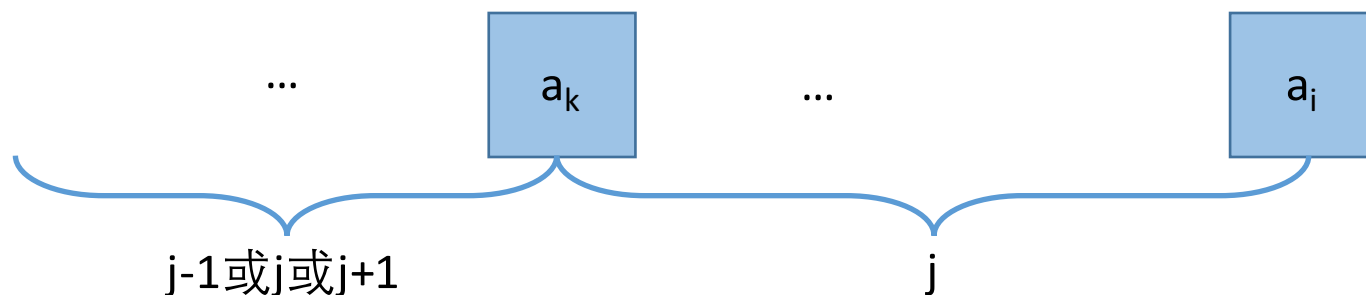
不需要枚举

$$f[i][j] = f[k][j-1] \text{ OR } f[k][j] \text{ OR } f[k][j+1]$$

能最后一跳 $j-1$ 跳到石头 a_k

能最后一跳 j 跳到石头 a_k

能最后一跳 $j+1$ 跳到石头 a_k



动态规划组成部分三：初始条件和边界情况

- 因为第一步跳的距离是1，一直向右跳，最多跳N-1步，所以一步最大跳跃距离是N-1
- 简单情况：如果只有一块石头，直接输出**TRUE**
- 如果石头1和石头0距离不是1，直接输出**FALSE**
- 第一步跳跃距离必须是1： $f[1][1] = \text{TRUE}$, $f[1][2] = \dots = f[1][N-1] = \text{FALSE}$

动态规划组成部分四：计算顺序

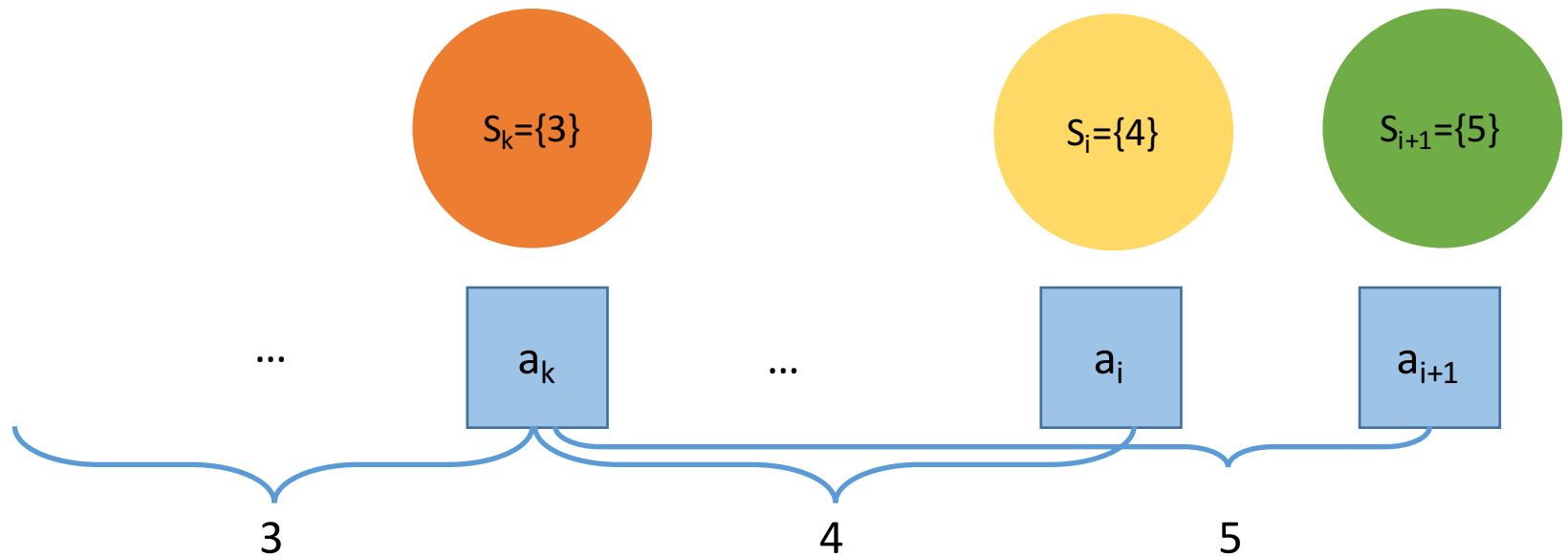
- $f[1][1], f[1][2], \dots, f[1][N-1]$
- ...
- $f[N-1][1], f[N-1][2], \dots, f[N-1][N-1]$
- 如果 $f[N-1][1], f[N-1][2], \dots, f[N-1][N-1]$ 中有任何一个是 `true`, 答案是 `true`, 否则是 `false`
- 时间复杂度 $O(N^2)$, 空间复杂度 $O(N^2)$, 不能用滚动数组优化, 因为 $f[i][j]$ 有可能依赖之前任何一个 $f[h][k]$

优化：动态规划加哈希表

- 在实际操作中，可以到达一个石头的最后一跳的值经常很少
 - 即 $f[i][1..N-1]$ 中很多都是FALSE
 - 没有必要计算，因为只关心 $f[i][j]=\text{TRUE}$ 的 i 和 j
- $f[i][j] = f[k][j-1] \text{ OR } f[k][j] \text{ OR } f[k][j+1]$
- 反过来想，如果已知 $f[k][j]=\text{TRUE}$ ，即可以最后一跳 j 到达石头 a_k
- 则可以跳到 a_k+j-1 , a_k+j 和 a_k+j+1 ，如果那里恰好有石头的话

动态规划加哈希表

- 我们用一个集合 S_i 保存能跳到一个石头 a_i 的可能的最后一跳
 - 其实就是原来的转移方程中 $f[i][j] = \text{TRUE}$ 的那些 j



优化：动态规划加哈希表

- 枚举每一个在集合 S_i 中的 L ，从石头 i 尝试往后跳 $L-1, L, L+1$
- 如果跳了 M 距离之后有一个石头 j ，则把 M 加到 S_j 中，表示可以最后一步跳 M 到达石头 j
 - 也就是 $f[j][M] = \text{TRUE}$
- 实际使用空间小

- 课间休息五分钟

Minimum Adjustment Cost

<http://www.lintcode.com/en/problem/minimum-adjustment-cost/>

<http://www.jiuzhang.com/solutions/minimum-adjustment-cost/>

LintCode 91 Minimum Adjustment Cost



- 题意：
 - 给定数组A，每个元素是不超过100的正整数
 - 将A中每个元素修改后形成数组B
 - 要求B中任意两个相邻的元素的差不能超过Target
 - 求最小修改代价，即 $|A[0]-B[0]| + \dots + |A[n-1]-B[n-1]|$
-
- 例子：
 - 输入：A=[1, 4, 2, 3], Target = 1
 - 输出：2 (B=[2, 3, 2, 3])

题目分析

- 可以证明，最优策略中**B**的每个元素也一定是不超过100的正整数
- 否则，将**B**中小于1的数改成1，大于100的数改成100
- 总的修改代价更小，且仍然满足**B**的任意两个相邻元素的差不大于Target

动态规划组成部分一：确定状态



- 最后一步：将A改成B， $A[n-1]$ 改成X，这一步代价是 $|A[n-1]-X|$
- 需要确保 $|X-B[n-2]| \leq \text{Target}$
- 前面 $n-1$ 个元素 $A[0..n-2]$ 改成 $B[0..n-2]$ ，需要知道最小代价，并确保 $B[0..n-2]$ 中任意两个相邻的元素的差不超过Target 子问题
- 但是有一个问题，改 $A[n-1]$ 时不知道 $B[n-2]$ 是多少
 - 只有知道了 $B[n-2]$ ，才能确定 $A[n-1]$ 能改成 $B[n-2]-\text{Target} \leq X \leq B[n-2]+\text{Target}$
- 不知道是多少就记录下来：序列加状态

动态规划组成部分一：确定状态

- 设状态 $f[i][j]$ 为将 A 前 i 个元素改成 B 的最小代价，确保前 i 个改好的元素中任意两个相邻的元素的差不超过 $Target$ ，并且 $A[i-1]$ 改成 j
- 这样，如果 $A[i-1]$ 改成 j ， $A[i-2]$ 就必须改成 $j-Target \leq k \leq j+Target$

动态规划组成部分二：转移方程

- 设 $f[i][j]$ 表示将 **A** 前 i 个元素改成 **B** 的最小代价，确保前 i 个改好的元素中任意两个相邻的元素的差不超过 **Target**，并且 **A**[$i-1$] 改成 j

$$f[i][j] = \min_{j-Target \leq k \leq j+Target, 1 \leq k \leq 100} \{f[i-1][k] + |j-A[i-1]|\}$$

将A前*i*个元素改成B的最小代价，A[i-1]改成j

将A前*i*-1个元素改成B的最小代价，A[i-2]改成k

A[i-1]改成j的代价

动态规划组成部分三：初始条件和边界情况

- 设 $f[i][j]$ 表示将A前 i 个元素改成B的最小代价，确保前 i 个改好的元素中任意两个相邻的元素的差不超过Target，并且 $A[i-1]$ 改成 j
- 初始条件：A的第一个元素可以变换成任意数字
 - 因为之前没有相邻的元素
 - $f[1][j] = |j - a[0]|$ ($j = 1, 2, \dots, 100$)

动态规划组成部分四：计算顺序

- $f[1][1], f[1][2], \dots, f[1][100]$
- ...
- $f[N][1], f[N][2], \dots, f[N][100]$
- 答案是 $\min\{f[N][1], f[N][2], \dots, f[N][100]\}$
- 时间复杂度 $O(100^2N)$, 空间复杂度 $O(100N)$, 可以用滚动数组优化至 $O(100)$

K-Sum

<http://www.lintcode.com/en/problem/k-sum/>

<http://www.jiuzhang.com/solutions/k-sum/>

LintCode 89 K Sum



- 题意：
- 给定数组A，包含n个互不相等的正整数
- 问有多少种方式从中找出K个数，使得它们的和是Target
- 例子：
- 输入：A=[1, 2, 3, 4], K=2, Target = 5
- 输出：2 (1 + 4 = 5, 2 + 3 = 5)

题目分析

- 要求从一些正整数中选出一些，使得和是Target
- 背包问题
- 数组A：各个物品的重量
- Target：背包最大称重
- 使得和是Target：背包正好装满

动态规划组成部分一：确定状态



- 最后一步：最后一个数 A_{n-1} 是否选入这 K 个数
- 情况一（ A_{n-1} 不选入）：需要在前 $n-1$ 个数中选 K 个数，使得它们的和是Target
- 情况二（ A_{n-1} 选入）：需要在前 $n-1$ 个数中选 $K-1$ 个数，使得它们的和是Target - A_{n-1}
- 要知道还有几个数可选，以及它们的和需要是多少：序列加状态
- 状态： $f[i][k][s]$ 表示有多少种方法可以在前 i 个数中选出 k 个，使得它们的和是 s

动态规划组成部分二：转移方程

- $f[i][k][s]$ 表示有多少种方法可以在前 i 个数中选出 k 个，使得它们的和是 s

$$f[i][k][s] = f[i-1][k][s] + f[i-1][k-1][s-A_{i-1}] \quad |s \geq A_{i-1}$$

有多少种方法可以在前 i 个数中选出 k 个，使得它们的和是 s

有多少种方法可以在前 $i-1$ 个数中选出 k 个，使得它们的和是 s

有多少种方法可以在前 $i-1$ 个数中选出 $k-1$ 个，使得它们的和是 $s-A_{i-1}$

动态规划组成部分三：初始条件和边界情况

- $f[i][k][s]$ 表示有多少种方法可以在前 i 个数中选出 k 个，使得它们的和是 s
- $f[i][k][s] = f[i-1][k][s] + f[i-1][k-1][s-A_{i-1}] | s \geq A_{i-1}$
- 初始条件：
 - $f[0][0][0] = 1$
 - $f[0][0][s] = 0, s = 1, 2, \dots, \text{Target}$
- 边界条件：
 - 如果 $s < A_{i-1}$, 只考虑情况一 $f[i-1][k][s]$

动态规划组成部分四：计算顺序

- $f[0][0 \sim K][0 \sim \text{Target}]$
- $f[1][0 \sim K][0 \sim \text{Target}]$
- ...
- $f[N][0 \sim K][0 \sim \text{Target}]$
- 答案是 $f[N][K][\text{Target}]$
- 时间复杂度 $O(N * K * \text{Target})$, 空间复杂度 $O(N * K * \text{Target})$, 可以用滚动数组优化至 $O(K * \text{Target})$

Maximal Square

<http://www.lintcode.com/problem/maximal-square/>
<http://www.jiuzhang.com/solutions/maximal-square/>

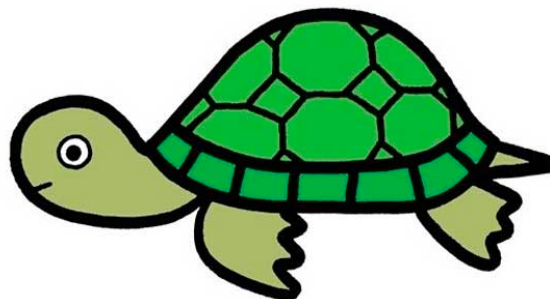
LintCode 436: Maximal Square

- 题意:
- 给定一个 $m \times n$ 的网格，每个格子里都是0或者1
- 求一块最大的全由1组成的正方形
- 输出面积

	0	1	2	3	4
0	0	1	1	1	0
1	0	1	1	1	1
2	1	1	1	1	0
3	0	0	1	0	0

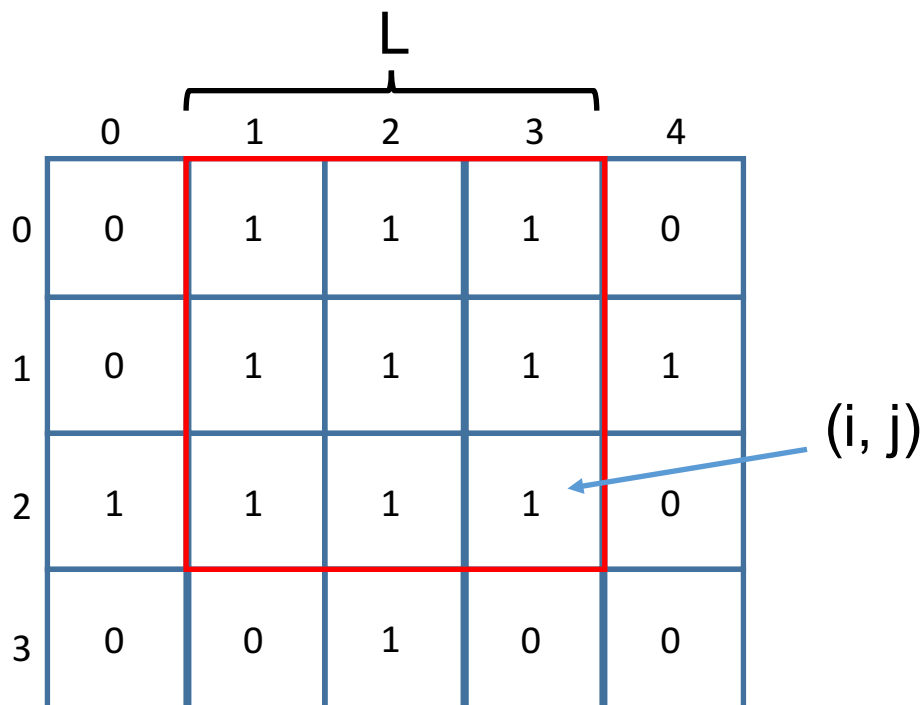
- 枚举左上角，枚举右下角，检查内部是否全是1
- 左上角和右下角各有 $O(M*N)$ 种可能性，内部大小也是 $O(M*N)$ 级别
- 时间复杂度 $O(M*N*M*N*M*N)$

太慢



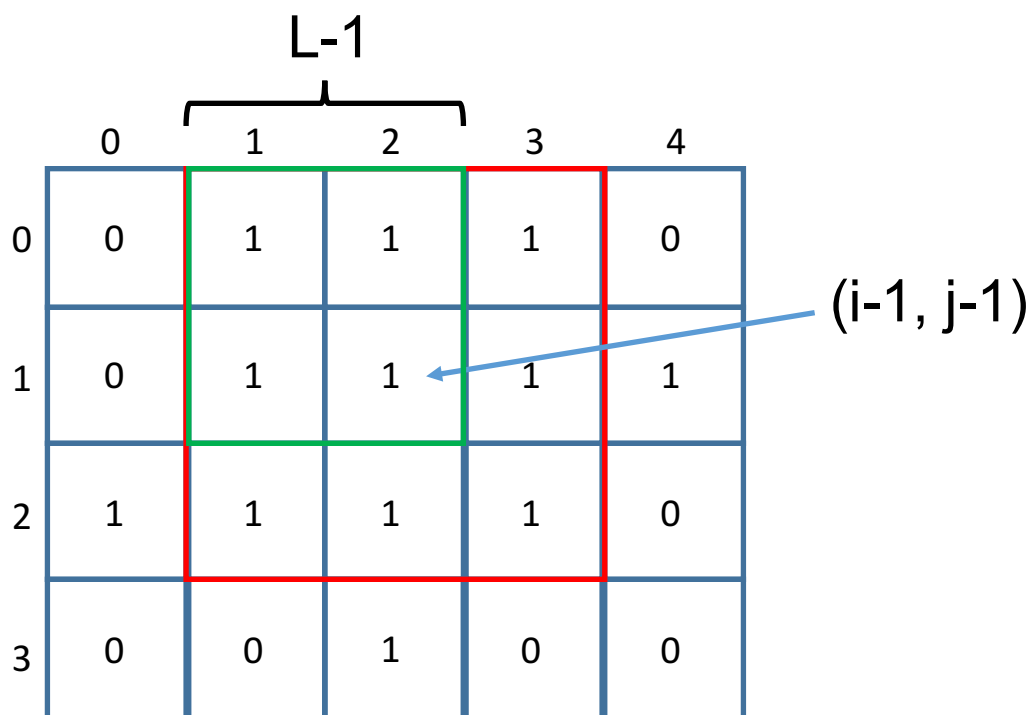
动态规划组成部分一：确定状态

- 最大的全1正方形，要么是边长为1，要么边长是 $L > 1$
- 右下角 (i, j) 肯定是1



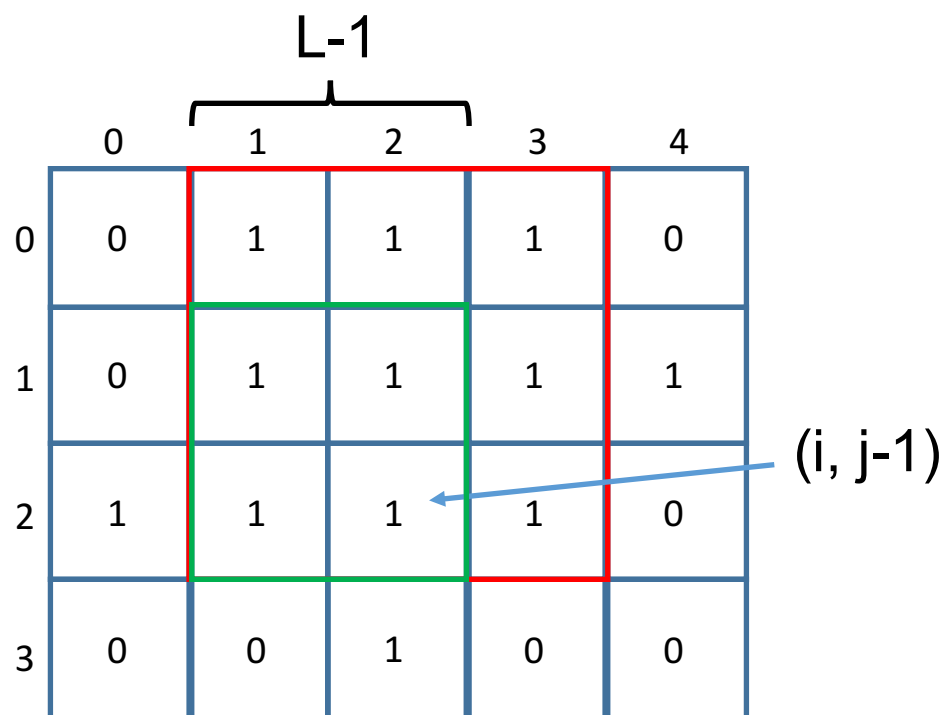
动态规划组成部分一：确定状态

- 以 $(i-1, j-1)$ 为右下角的最大全1正方形边长至少是 $L-1$



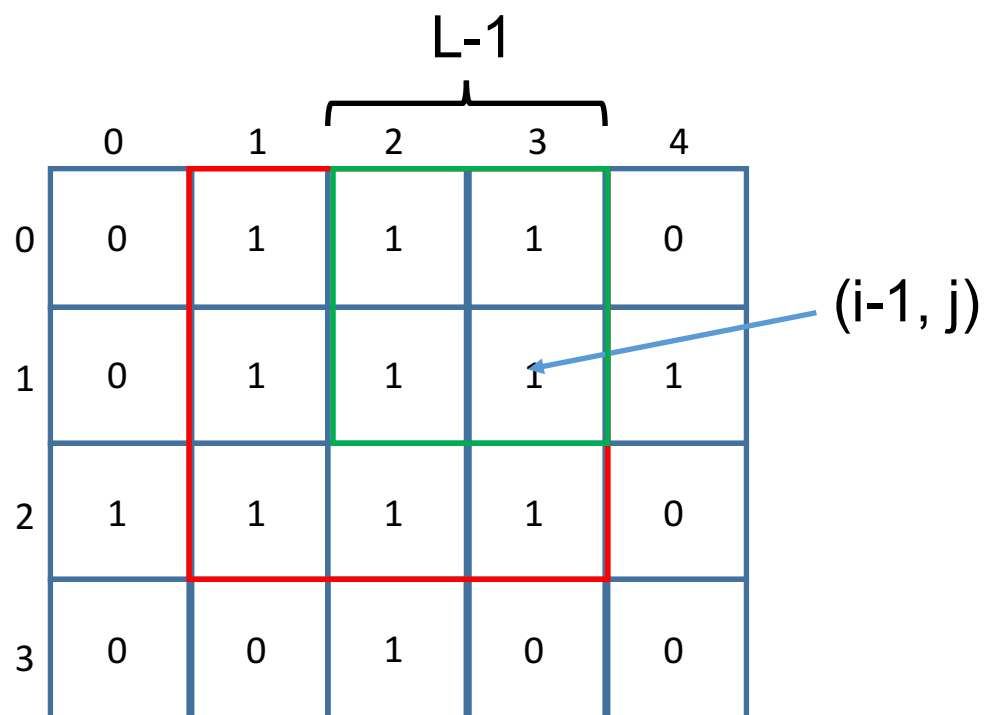
动态规划组成部分一：确定状态

- 以 $(i, j-1)$ 为右下角的最大全1正方形边长至少是 $L-1$



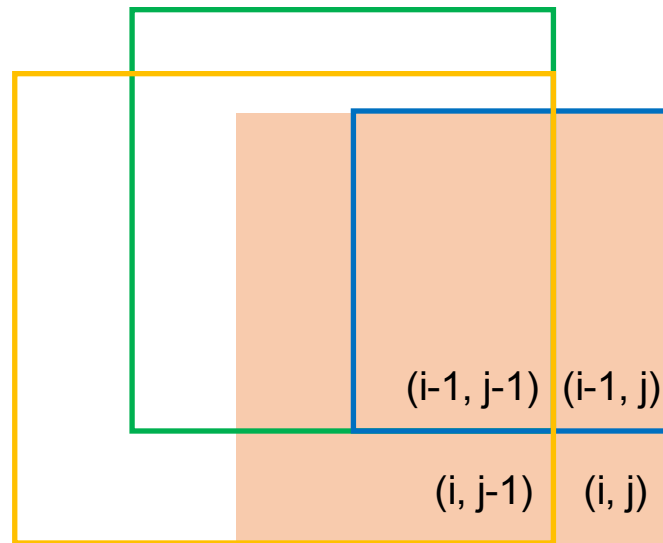
动态规划组成部分一：确定状态

- 以 $(i-1, j)$ 为右下角的最大全1正方形边长至少是 $L-1$



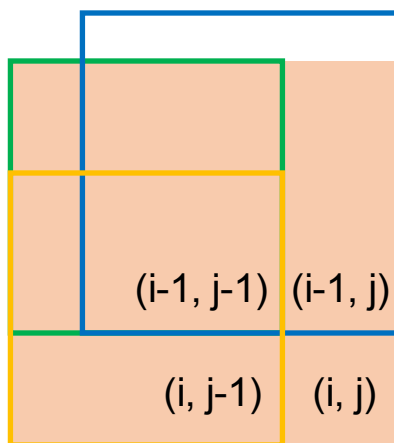
动态规划组成部分一：确定状态

- 换个角度想，如果以 $(i-1, j-1)$, $(i-1, j)$, $(i, j-1)$ 为右下角的最大全1正方形的边长分别是 L_1, L_2 和 L_3 ，而 (i, j) 格子里是1，那么以 (i, j) 为右下角的最大全1正方形的边长应该是 $\min\{L_1, L_2, L_3\} + 1$



动态规划组成部分一：确定状态

- 换个角度想，如果以 $(i-1, j-1)$, $(i-1, j)$, $(i, j-1)$ 为右下角的最大全1正方形的边长分别是 L_1 , L_2 和 L_3 ，而 (i, j) 格子里是1，那么以 (i, j) 为右下角的最大全1正方形的边长应该是 $\min\{L_1, L_2, L_3\} + 1$



- 于是，需要求以 $(i-1, j-1)$, $(i-1, j)$, $(i, j-1)$ 为右下角的最大全1正方形的边长
- 而原来是求以 (i, j) 为右下角的最大全1正方形的边长
- 子问题
- 状态：设 $f[i][j]$ = 以 (i, j) 为右下角的最大全1正方形的边长
- 坐标型动态规划

动态规划组成部分二：转移方程

- 设 $f[i][j]$ = 以 (i, j) 为右下角的最大全1正方形的边长

$$f[i][j] = \begin{cases} 0, & \text{如果}(i, j)\text{格是}0 \\ \min\{f[i-1][j], f[i][j-1], f[i-1][j-1]\} + 1, & \text{如果}(i, j)\text{格是}1 \end{cases}$$

动态规划组成部分三：初始条件和边界情况

- $i=0$ 或者 $j=0$ ，即最上边一行或最左边一列

	0	1	2	3	4
0	0	1	1	1	0
1	0	1	1	1	1
2	1	1	1	1	0
3	0	0	1	0	0

$$f[i][j] = \begin{cases} 0, & \text{如果}(i,j)\text{格是}0 \\ 1, & \text{如果}(i,j)\text{格是}1\text{且}i=0\text{或}j=0 \\ \min\{f[i-1][j], f[i][j-1], f[i-1][j-1]\} + 1, & \text{如果}(i,j)\text{格是}1 \end{cases}$$

- 初始条件：空

动态规划组成部分四：计算顺序

- $f[0][0], f[0][1], \dots, f[0][n-1]$
- $f[1][1], f[1][2], \dots, f[1][n-1]$
- ...
- $f[m-1][0], f[m-1][1], \dots, f[m-1][n-1]$
- 答案是 $\max_{i,j}\{f[i][j]^2\}$
- 时间复杂度（计算步数）： $O(MN)$, 空间复杂度（数组大小）： $O(MN)$

Maximal Rectangle

<http://www.lintcode.com/problem/maximal-rectangle/>
<http://www.jiuzhang.com/solutions/maximal-rectangle/>

LintCode 510 Maximum Rectangle



- 题意：
- 给定一个01矩阵，找到其中面积最大的全1长方形
- 例子：
- 输入：
 - 11001
 - 01001
 - 00111
 - 00111
 - 00001
- 输出： 6

LintCode 510 Maximum Rectangle



- 题意：
- 给定一个01矩阵，找到其中面积最大的全1长方形
- 例子：
- 输入：
 - 11001
 - 01001
 - 00111
 - 00111
 - 00001
- 输出： 6

动态规划组成部分一：确定状态

- 最优策略产生**最大的全1**长方形
- 最后一步：观察它的最后一行的1
- 每一个1向上延伸一直到边界或0
- 一定至少有一根**顶梁柱**
 - 只能到达长方形上边缘

思考：为什么？

	0	1	2	3	4	5
0	0	1	0	1	0	0
1	0	1	1	1	1	1
2	1	1	1	1	0	0
3	0	0	1	0	0	0

动态规划组成部分一：确定状态

- 这根顶梁柱向左最多到达长方形左边缘

	0	1	2	3	4	5
0	0	1	0	1	0	0
1	0	1	1	1	1	1
2	1	1	1	1	0	0
3	0	0	1	0	0	0

动态规划组成部分一：确定状态

- 这根顶梁柱向左最多到达长方形左边缘
- 向右最多到达长方形右边缘

	0	1	2	3	4	5
0	0	1	0	1	0	0
1	0	1	1	1	1	1
2	1	1	1	1	0	0
3	0	0	1	0	0	0

动态规划组成部分一：确定状态

- 所以，最大长方形一定有一个1作为顶梁柱的基石
- 向上找到顶梁柱的长度
- 向左移动到极限，向右移动到极限
 - 一直是全1
- 找到所有的顶梁柱，左右移动
- 就可以找到最大长方形！

	0	1	2	3	4	5
0	0	1	0	1	0	0
1	0	1	1	1	1	1
2	1	1	1	1	0	0
3	0	0	1	0	0	0

子问题

- 求一个格子 (i, j) 的顶梁柱长度 $up[i][j]$
 - 即往上最多延伸多少个1
- 如果 (i, j) 是0, 则 $up[i][j] = 0$
- 如果 (i, j) 是1, 则 $up[i][j] = up[i-1][j] + 1$

	0	1	2	3	4	5
0	0	1	0	1	0	0
1	0	1	1	1	1	1
2	1	1	1	1	0	0
3	0	0	1	0	0	0

子问题

- 求一个格子 (i, j) 的顶梁柱最多往左延伸多久
– $\text{left}[i][j]$
- 如果 (i, j) 是0, 则 $\text{left}[i][j] = 0$
- 如果上面一个格子 $(i-1, j)$ 是0, 则 $\text{left}[i][j] = (i, j)$ 往左最多连续有多少个1
- 否则 $\text{left}[i][j] = \min\{(i, j)$ 往左最多连续有多少个1, $(i-1, j)$ 的顶梁柱最多往左延伸多久 $\}$
- 同理求出 $\text{right}[i][j]$

	0	1	2	3	4	5
0	0	1	0	1	0	0
1	0	1	1	1	1	1
2	1	1	1	1	0	0
3	0	0	1	0	0	0

子问题

- 求 (i, j) 往左最多连续有多少个1
- 如果 (i, j) 是0, 答案是0
- 如果 (i, j) 是1, 答案是 $1 + (i, j-1)$ 往左最多连续有多少个1
- 同理求出 (i, j) 往右最多连续有多少个1

	0	1	2	3	4	5
0	0	1	0	1	0	0
1	0	1	1	1	1	1
2	1	1	1	1	0	0
3	0	0	1	0	0	0

动态规划组成部分四：计算顺序



- 对于up, 逐行求, 每一行从左到右
- 对于left, 同理
- 对于right, 每一行从右到左
- 答案是 $\max_{i,j}\{up[i][j] * (left[i][j] + right[i][j] - 1)\}$
- 时间复杂度 $O(MN)$, 空间复杂度 $O(MN)$, 可以滚动数组优化至 $O(N)$

- 常见动态规划类型
 - 坐标型动态规划 (20%)
 - 序列型动态规划 (20%)
 - 划分型动态规划 (20%)
 - 区间型动态规划 (15%)
 - 背包型动态规划 (10%)
 - 最长序列型动态规划 (5%)
 - 博弈型动态规划 (5%)
 - 综合性动态规划 (5%)

- 确定状态
 - 研究最优策略的最后一步
 - 化为子问题
- 转移方程
 - 根据子问题定义直接得到
- 初始条件和边界情况
 - 细心，考虑周全
- 计算顺序
 - 利用之前的计算结果

- 谢谢大家！
- 祝各位同学面试顺利，拿到自己理想的offer