难题专场

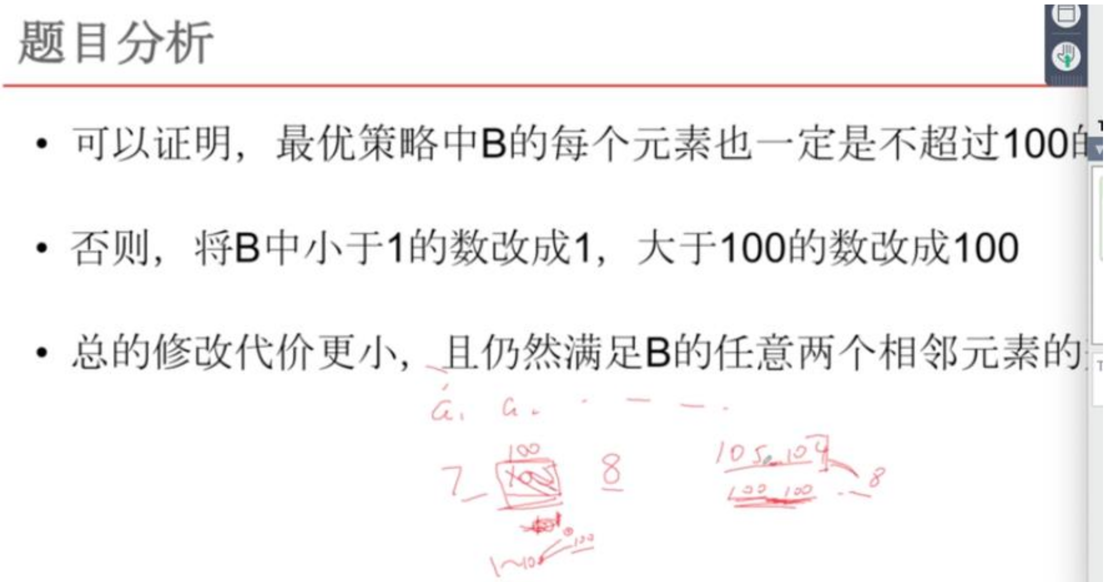
难题专场

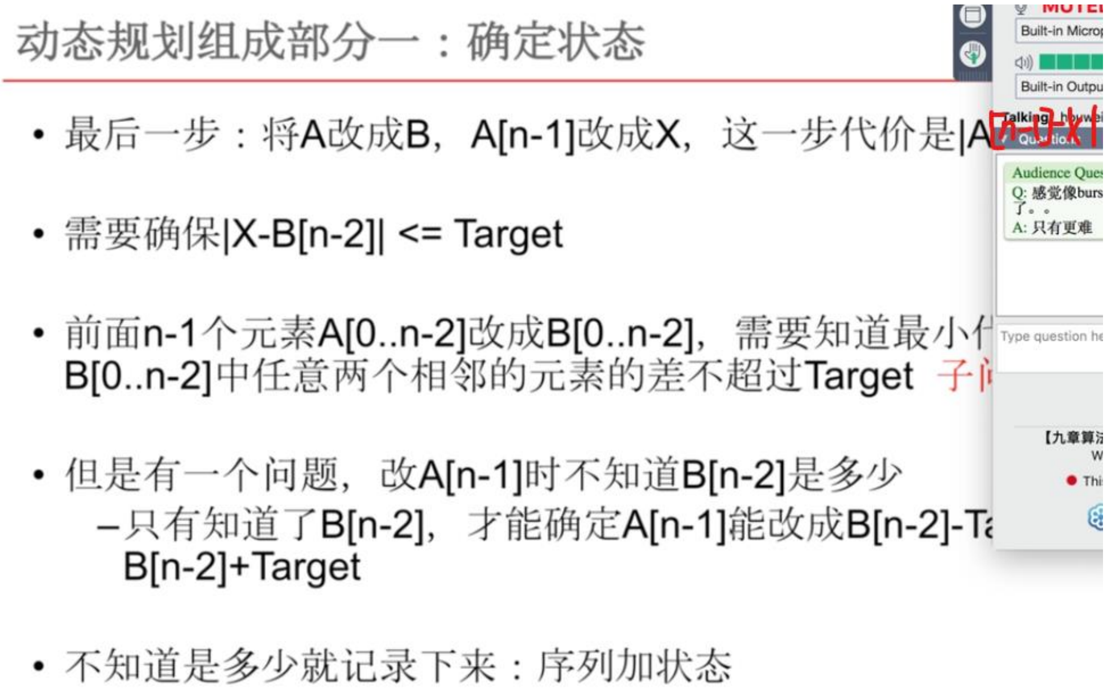
- LintCode上较难的动态规划题目
- 综合型动态规划
- 需要辅助数据结构 /算法 (字母树，哈希表，二分查找) 的
- 万变不离其宗

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]| + … + |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])





- 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 <=k <=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**

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

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, …, 100)

动态规划组成部分四：计算顺序

- f[1][1], f[1][2], …, f[1][100]

- …

- f[N][1], f[N][2], …, f[N][100]

- 答案是min{f[N][1], f[N][2], …, 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)

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)

与背包问题很类似。

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

$$f[i][k][s] = f[i-1][k][s] + f[i-1][k-1][s-A_{i-1}] | s>=A_{i-1}$$

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

九章算法

- 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
- $f[i][k][s] = f[i-1][k][s] + f[i-1][k-1][s-A_{i-1}] | s>=A_{i-1}$
- 初始条件：
  - f[0][0][0] = 1
  - f[0][0][s] = 0, s = 1, 2, …, 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

九章算法

- f[0][0~K][0~Target]
- f[1][0~K][0~Target]
- …
- f[N][0~K][0~Target]
- 答案是f[N][K][Target]
- 时间复杂度O(N*K*Target)，空间复杂度O(N*K*Target)，可以用滚动数组优化至O(K*Target)

LintCode 76 Longest Increasing Subsequence

九章算法

- 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）

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| i < j 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| i < j 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]

f[0] f[1]

	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| i < j 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]已经没有用，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| i < j 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]已经没有用，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, …, 记录当前为止拥有这个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, …, 记录当前为止拥有这个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

f=1 3 f=2 5 f=3

思考：为什么

- 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)中找到最后一个比它小的数a[i]，f[j] 就是 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a[i]的下一个，因为f[j]和它值一样，但f[j]更小

二分查找优化

九章算法

- 在序列(a[1]=1, a[6]=2, a[5]=3)中找到最后一个比它小的数a[i]，f[j] 就是 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<=N，因为最长上升子序列长度<=N
- 每次查找时间复杂度$O(\log_2 N)$
- 总的时间复杂度$O(N\log_2 N)$

LintCode 623 K Edit Distance

九章算法

- 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

九章算法

- 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

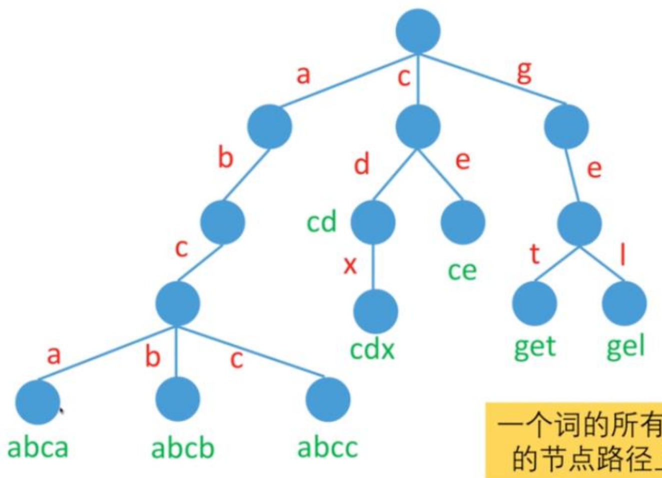
- 存在重复计算
  - 如果给定的字符串是"abca", "abcb", "abcc"
  - 三个字符串的前3个字符都一样
  - "abca"前0~3个字符和Target前0~n个字符的最小编辑距离
  - "abcb"前0~3个字符和Target前0~n个字符的最小编辑距离
  - "abcc"前0~3个字符和Target前0~n个字符的最小编辑距离

重复计算了3次

题目分析

- 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

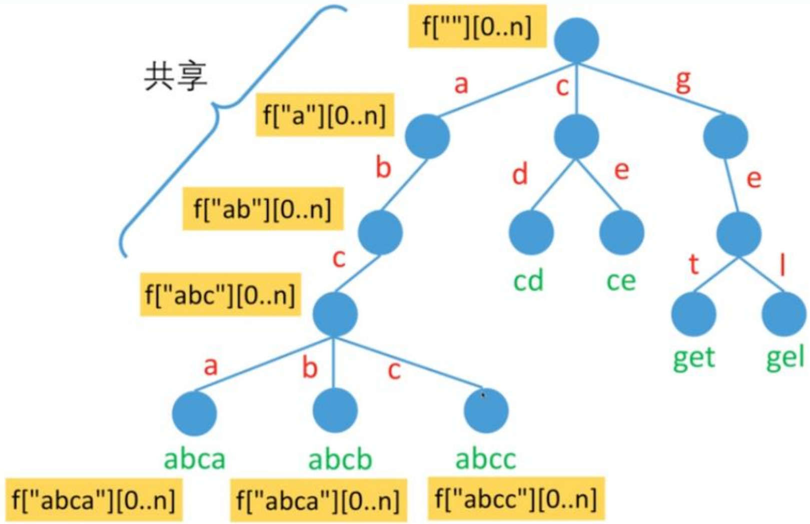


一个词的所有前缀就是从根到这个词的节点路径上形成的所有的字符串

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]~f[0][n]
    - f[1][0]~f[1][n]
    - f[2][0]~f[2][n]
    - …
- 现在，因为有多个字符串A_1, A_2, …，我们可以将用f[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

$$f[s_P][j] = \min\{f[s_P][j-1]+1,\ f[s_Q][j-1]+1,\ f[s_Q][j]+1,\ f[s_Q][j-1] \mid s_P[last]=Target[j-1]\}$$

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第j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
  - $f[s_{root}][j] = f[""][j] = j$ $(j = 0, 1, 2, \ldots, n)$
  - $f[s_p][0] = \text{length}(s_p)$

动态规划组成部分四：计算顺序

- 初始化$f[s_{root}][0]$~$f[s_{root}][n]$
- 按照字母树深度优先搜索顺序计算每个$f[s_P][0]$~$f[s_P][n]$
- 答案是满足$f[s_P][n]<=K$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(前缀个数*N)，空间复杂度（数组大小）O(前缀个数*N)

序列+哈希表

- 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

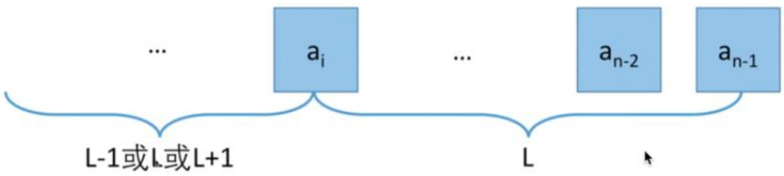
LintCode 622 Frog Jump

- 题意：
- 有一条小河上有N个石头，位置依次在$a_0<a_1<\ldots<a_{n-1}$
- 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
- 它必须一直向右跳，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+1
- 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

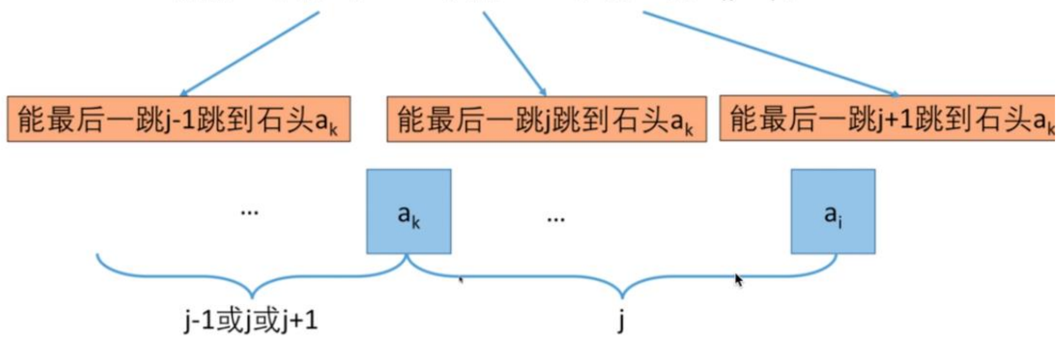
- 要求是否能最后一跳L跳到最后一个石头a_{n-1}
- 需要知道能否最后一跳L-1, L或者L+1跳到石头$a_i = a_{n-1}$-L
- 子问题
- 状态：设f[i][j]表示是否能最后一跳长度j跳到石头a_i
- 坐标+状态型动态规划

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

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



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] = TRUE, f[1][2] = … = f[1][N-1] = FALSE

动态规划组成部分四：计算顺序

- f[1][1], f[1][2], …, f[1][N-1]
- …
- f[N-1][1], f[N-1][2], …, f[N-1][N-1]
- 如果f[N-1][1], f[N-1][2], …, 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]=TRUE的i和j
- f[i][j] = f[k][j-1] OR f[k][j] OR f[k][j+1] | $a_k=a_i-j$
- 反过来想，如果已知f[k][j]=TRUE，即可以最后一跳j到达石头a_k

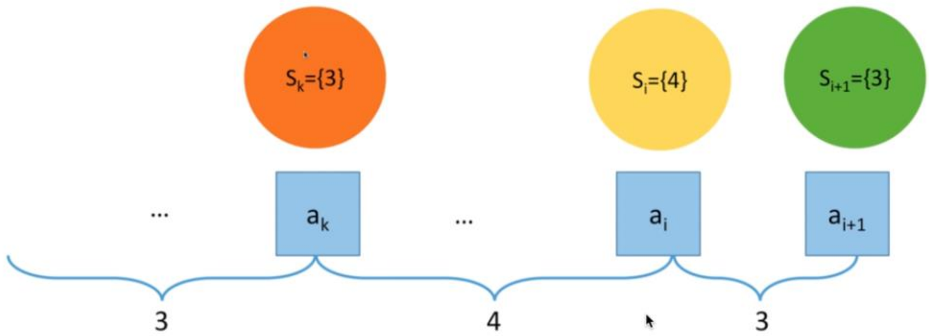
  j+1

  $f[a_k+j-1][j-1]=T$ 索取 贡献
- 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

索取型与贡献型

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] = 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] = TRUE

- 实际使用空间小

LintCode 436: Maximal Square

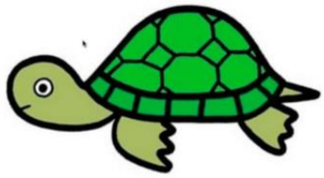
- 题意：
- 给定一个mxn的网格，每个格子里都是0或者1
- 求一块最大的全由1组成的正方形
- 输出**面积**

	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

直觉

- 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

L

	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

(i, j)

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

L-1

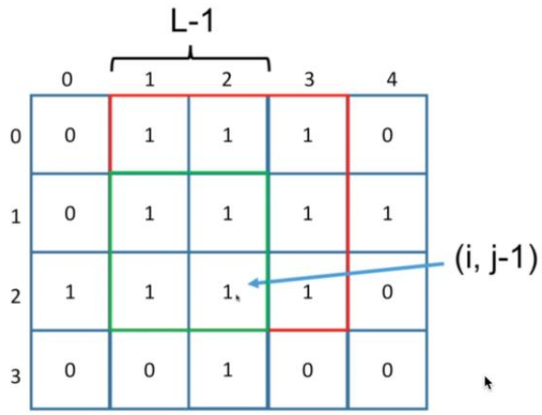
	0	1	2	3	4
0	0	1	1	1	0
1	0	1	1	1	1
2	1	1	1	1	0
3	0	0	1	0	0

(i-1, j-1)

动态规划组成部分一：确定状态

九章算法

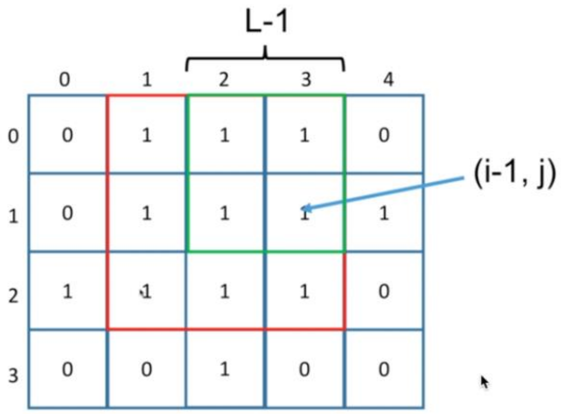
- 以(i, j-1)为右下角的最大全1正方形边长至少是L-1



动态规划组成部分一：确定状态

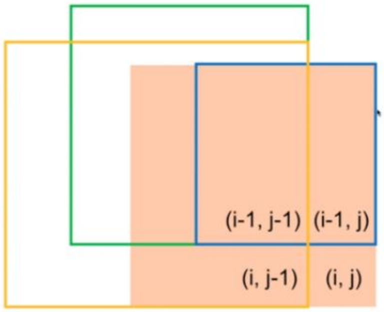
九章算法

- 以(i-1, j)为右下角的最大全1正方形边长至少是L-1



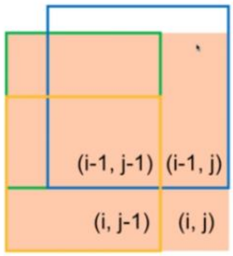
动态规划组成部分一：确定状态

- 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$\min\{L_1, L_2, L_3\} + 1$



子问题

- 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

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

- 初始条件：空

动态规划组成部分四：计算顺序

- f[0][0], f[0][1], …, f[0][n-1]
- f[1][1], f[1][2], …, f[1][n-1]
- …
- f[m-1][0], f[m-1][1], …, 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)

课程总结

- 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%)