

## 双序列型动态规划

九章算法

- 顾名思义，有两个序列/字符串，需要进行一些操作
- 每个序列本身是一维的
- 可以转化为二维动态规划



## LintCode 77 Longest Common Subsequence

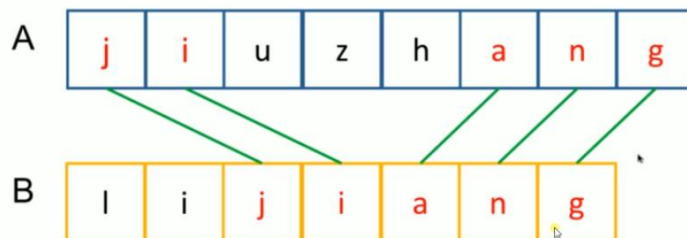
九章算法

- 题意：
- 给定两个字符串A, B
- 一个字符串的子串是这个字符串去掉某些字符（可能0个）之后剩下的字符串
- 找到两个字符串的最长公共子串的长度
- 例子：
- 输入：A="jiuzhang", B="lijiang"
- 输出：5（最长公共子串是jiang）

## 题目分析

九章算法

- 公共子串一定是对应的字符按顺序都相等
- 找到最长的对应子，且对子连线不能相交



## 动态规划组成部分一：确定状态

九章算法

- 设A长度是m, B长度是n
- 现在我们考虑最优策略产生出的最长公共子串（虽然还不知道是什么）
- 最后一步：观察A[m-1]和B[n-1]这两个字符是否作为一个对子在最优策略中

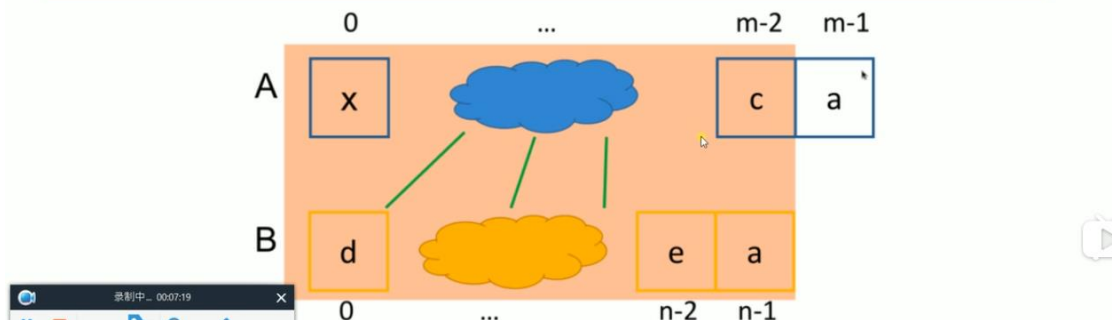
## 动态规划组成部分一：确定状态

- 最长公共子串也是公共子串：长度是 $L \rightarrow$ 选定了 $L$ 个对应的对子

最长公共子串

情况一：对子中没有 $A[m-1]$

推论：A和B的最长公共子串就是A前 $m-1$ 个字符和B前 $n$ 个字符的最长公共子串



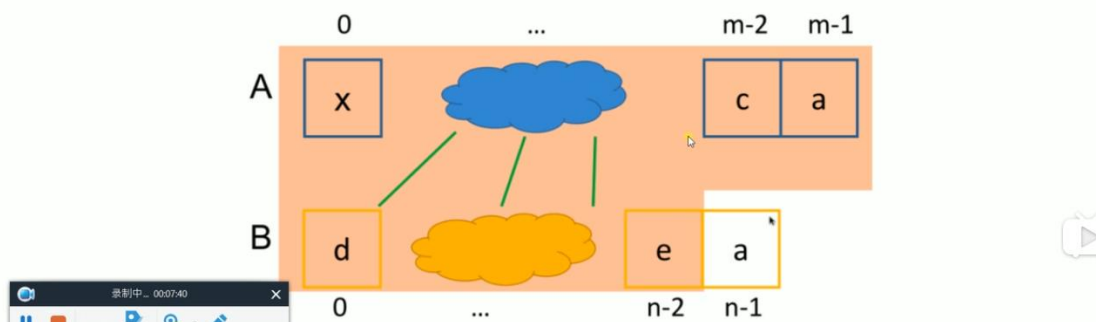
## 动态规划组成部分一：确定状态

- 最长公共子串也是公共子串：长度是 $L \rightarrow$ 选定了 $L$ 个对应的对子

最长公共子串

情况二：对子中没有 $B[n-1]$

推论：A和B的最长公共子串就是A前 $m$ 个字符和B前 $n-1$ 个字符的最长公共子串



## 动态规划组成部分一：确定状态

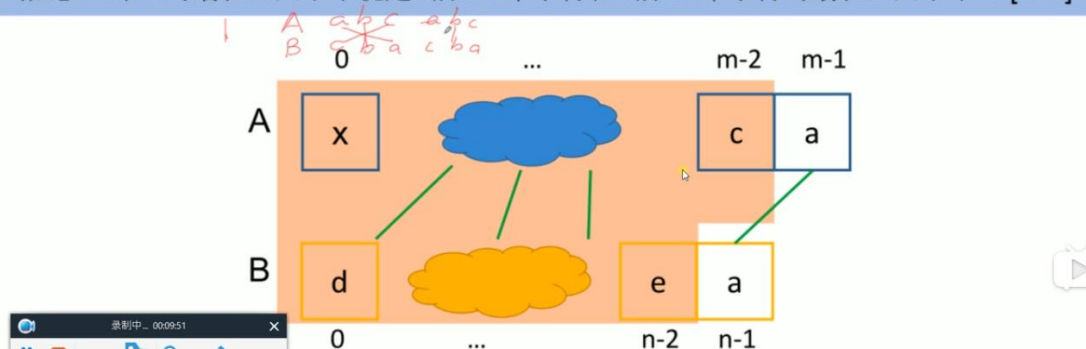
九章算法

- 最长公共子串也是公共子串：长度是 $L \rightarrow$ 选定了 $L$ 个对应的对子

最长公共子串

情况三：对子中有 $A[m-1]-B[n-1]$

推论：A和B的最长公共子串就是A前 $m-1$ 个字符和B前 $n-1$ 个字符的最长公共子串+A[ $m-1$ ]



## 子问题

九章算法

- 要求 $A[0..m-1]$ 和 $B[0..n-2]$ 的最长公共子串， $A[0..m-2]$ 和 $B[0..n-1]$ 的最长公共子串和 $A[0..m-2]$ 和 $B[0..n-2]$ 的最长公共子串
- 原来是求 $A[0..m-1]$ 和 $B[0..n-1]$ 的最长公共子串
- 子问题
- 状态：设 $f[i][j]$ 为A前 $i$ 个字符 $A[0..i-1]$ 和B前 $j$ 个字符 $B[0..j-1]$ 的最长公共子串的长度

## 动态规划组成部分二：转移方程

九章算法

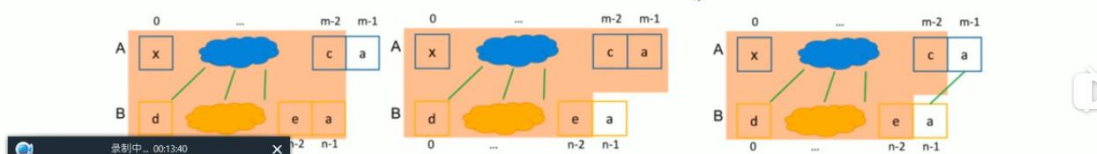
- 设 $f[i][j]$ 为A前 $i$ 个字符 $A[0..i-1]$ 和B前 $j$ 个字符 $B[0..j-1]$ 的最长公共子串的长度
- 要求 $f[m][n]$

$$f[i][j] = \max\{f[i-1][j], f[i][j-1], f[i-1][j-1]+1 | A[i-1]=B[j-1]\}$$

情况一：A[0..i-2]和B[0..j-1]的最长公共子串

情况二：A[0..i-1]和B[0..j-2]的最长公共子串

情况三：A[0..i-2]和B[0..j-2]的最长公共子串+A[i-1]



## 动态规划组成部分三：初始条件和边界情况

- $f[i][j]$  为 A 前  $i$  个字符  $A[0..i-1]$  和 B 前  $j$  个字符  $B[0..j-1]$  的最长公共子串的长度
- 转移方程： $f[i][j] = \max\{f[i-1][j], f[i][j-1], f[i-1][j-1]+1 | A[i-1]=B[j-1]\}$
- 初始条件：空串和任何串的最长公共子串长度是 0
  - $f[0][j] = 0, j=0..n$
  - $f[i][0] = 0, i=0..m$

## 动态规划组成部分四：计算顺序

- $f[0][0], f[0][1], \dots, f[0][n]$
- $f[1][0], f[1][1], \dots, f[1][n]$
- ...
- $f[m][0], f[m][1], \dots, f[m][n]$
- 答案是  $f[m][n]$
- 时间复杂度（计算步数） $O(MN)$ ，空间复杂度（数组大小） $O(MN)$

计算顺序是由转移方程决定的。

## LintCode 29 Interleaving String

- 题意：
- 给定三个字符串 A, B, X
- 判断 X 是否是由 A, B 交错在一起形成
  - 即 A 是 X 的子序列，去掉 A 后，剩下的字符组成 B
- 例子：
- 输入：A="aabcc" B="dbbac", X="aadbcbcbac"
- 输出：True (X="aadbcbcbac")

## 动态规划组成部分一：确定状态



- 首先，如果X的长度不等于A的长度+B的长度，直接输出False

- 设A长度是m, B长度是n, X的长度是m+n

- 最后一步：假设X是由A和B交错形成的，那么X的最后一个字符X[m+n-1]
  - 要么是A[m-1]
    - 那么X[0..m+n-2]是由A[0..m-2]和B[0..n-1]交错形成的
  - 要么是B[n-1]
    - 那么X[0..m+n-2]是由A[0..m-1]和B[0..n-2]交错形成的

## 子问题



- 要求X[0..m+n-1]是否由A[0..m-1]和B[0..n-1]交错形成
- 需要知道X[0..m+n-2]是否由A[0..m-2]和B[0..n-1]交错形成，以及X[0..m+n-2]是否由A[0..m-1]和B[0..n-2]交错形成
- 子问题
- 状态：设f[s][i][j]为X前s个字符是否由A前i个字符A[0..i-1]和B前j个字符B[0..j-1]交错形成
- 但是s=i+j，所以可以简化为：设f[i][j]为X前i+j个字符是否由A前i个字符A[0..i-1]和B前j个字符B[0..j-1]交错形成

## 动态规划组成部分二：转移方程



- 设f[i][j]为X前i+j个字符是否由A前i个字符A[0..i-1]和B前j个字符B[0..j-1]交错形成

$$f[i][j] = (f[i-1][j] \text{ AND } X[i+j-1] == A[i-1]) \text{ OR } (f[i][j-1] \text{ AND } X[i+j-1] == B[j-1])$$

X前i+j个字符是否由A前i个字符和B前j个字符交错形成

情况一：X前i+j-1个字符由A前i-1个字符和B前j个字符交错形成，X第i+j个字符等于A第i个字符

情况二：X前i+j-1个字符由A前i个字符和B前j-1个字符交错形成，X第i+j个字符等于B第j个字符



### 动态规划组成部分三：初始条件和边界情况



- 设 $f[i][j]$ 为X前 $i+j$ 个字符是否由A前 $i$ 个字符 $A[0..i-1]$ 和B前 $j$ 个字符 $B[0..j-1]$ 交错形成
- 转移方程
  - $f[i][j] = (f[i-1][j] \text{ AND } X[i+j-1] == A[i-1]) \text{ OR } (f[i][j-1] \text{ AND } X[i+j-1] == B[j-1])$
- 初始条件：空串由A的空串和B的空串交错形成  $\rightarrow f[0][0] = \text{True}$
- 边界情况：如果 $i=0$ ，不考虑情况一；如果 $j=0$ ，不考虑情况二

### 动态规划组成部分四：计算顺序



- $f[0][0], f[0][1], \dots, f[0][n]$
- $f[1][0], f[1][1], \dots, f[1][n]$
- ...
- $f[m][0], f[m][1], \dots, f[m][n]$
- 答案是 $f[m][n]$
- 时间复杂度（计算步数） $O(MN)$ ，空间复杂度（数组大小） $O(MN)$ ，可以用滚动数组优化空间至 $O(N)$

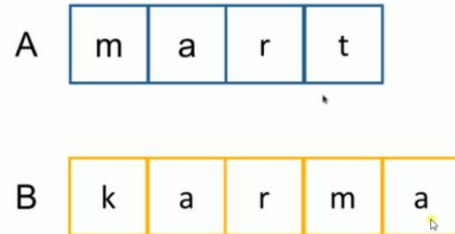
### LintCode 119 Edit Distance



- 题意：
- 给定两个字符串A, B
- 要求把A变成B，每次可以进行下面一种操作：
  - 增加一个字符
  - 去掉一个字符
  - 替换一个字符
- 最少需要多少次操作，即最小编辑距离
- 例子：
- 输入：A="mart", B="karma"
- 输出：3 (m换成k, t换成m, 加上a)

## 题目分析

- 可以进行的操作很多，直接下手比较困难



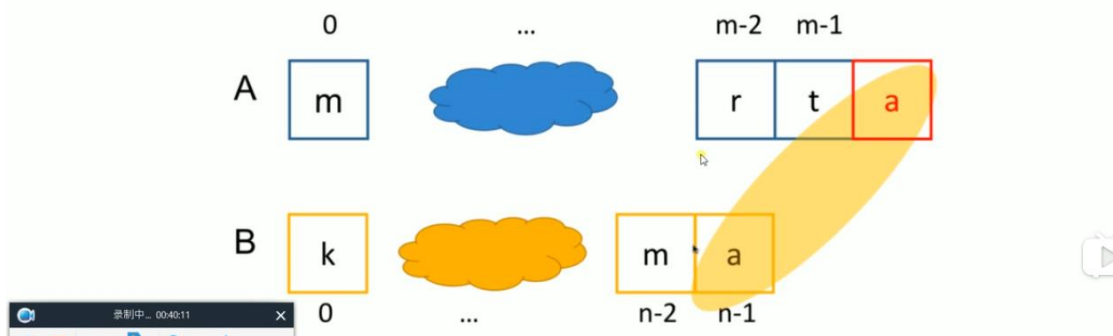
## 动态规划组成部分一：确定状态

- 设A长度是m, B长度是n
- 全部操作完成后A的长度也是n, 并且 $A[n-1]=B[n-1]$
- 于是最优策略（以及所有合法策略）最终都是让A的最后一个字符变成B的最后一个字符

## 动态规划组成部分一：确定状态

情况一：A在最后插入B[n-1]

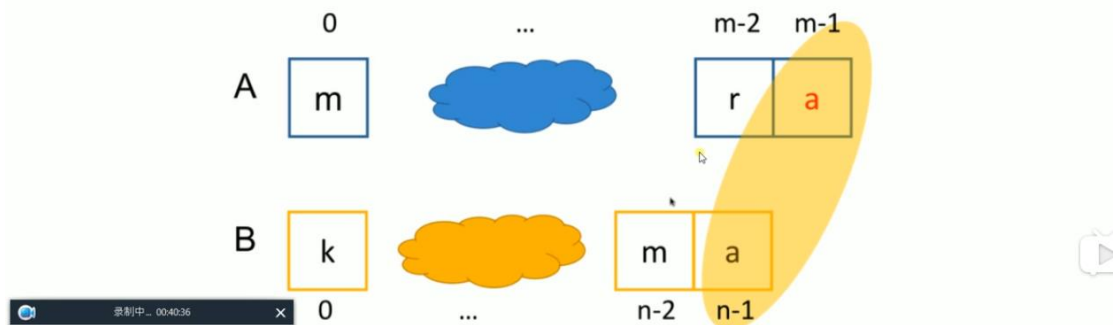
- 要将A[0..m-1]变成B[0..n-2]



## 动态规划组成部分一：确定状态

九章算法

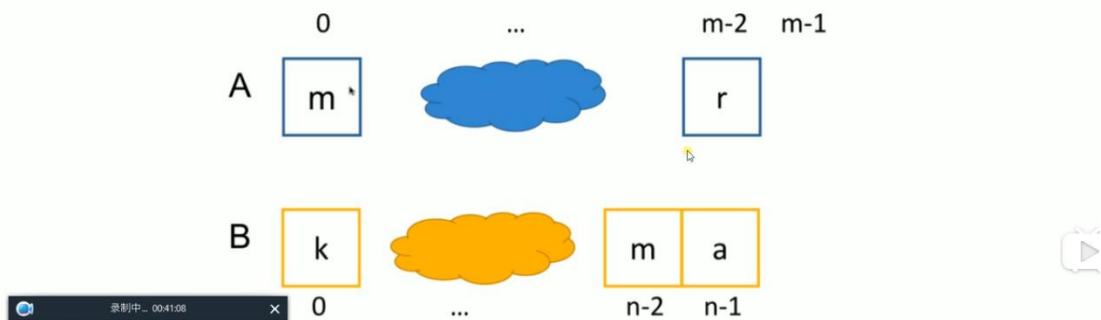
情况二：A最后一个字符替换成B[n-1] • 要将A[0..m-2]变成B[0..n-2]



## 动态规划组成部分一：确定状态

九章算法

情况三：A删掉最后一个字符 • 要将A[0..m-2]变成B[0..n-1]



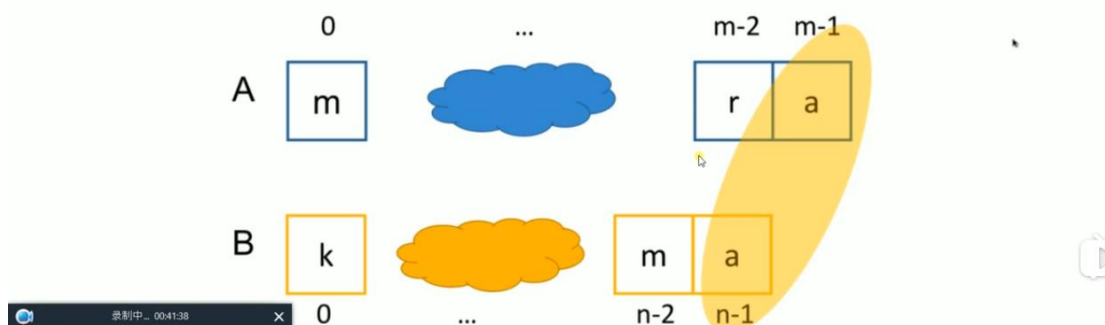


## 动态规划组成部分一：确定状态

九章算法

情况四：A和B最后一个字符相等

- 要将A[0..m-2]变成B[0..n-2]



## 子问题

九章算法

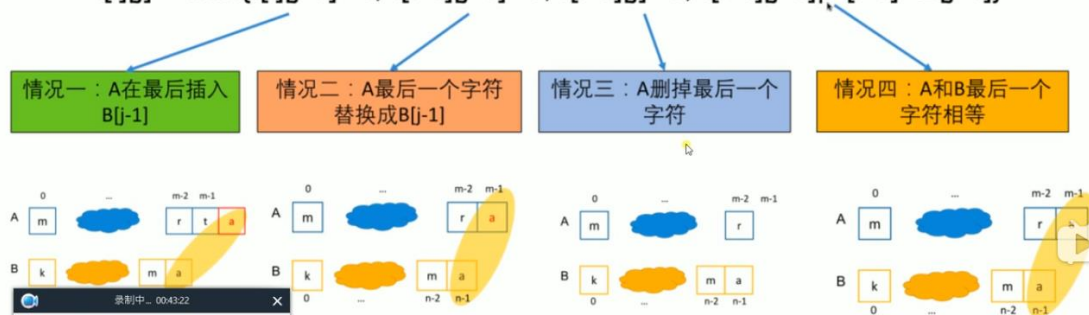
- 要求A[0..m-1]和B[0..n-2]的最小编辑距离, A[0..m-2]和B[0..n-1]的最小编辑距离和A[0..m-2]和B[0..n-2]的最小编辑距离
- 原来是求A[0..m-1]和B[0..n-1]的最小编辑距离
- 子问题
- 状态：设 $f[i][j]$ 为A前i个字符A[0..i-1]和B前j个字符B[0..j-1]的最小编辑距离

## 动态规划组成部分二：转移方程

九章算法

- 设 $f[i][j]$ 为A前i个字符A[0..i-1]和B前j个字符B[0..j-1]的最小编辑距离
- 要求 $f[m][n]$

$$f[i][j] = \min\{f[i][j-1]+1, f[i-1][j-1]+1, f[i-1][j]+1, f[i-1][j-1] | A[i-1]=B[j-1]\}$$



### 动态规划组成部分三：初始条件和边界情况



- 设  $f[i][j]$  为 A 前 i 个字符  $A[0..i-1]$  和 B 前 j 个字符  $B[0..j-1]$  的最小编辑距离
- 转移方程： $f[i][j] = \min\{f[i][j-1]+1, f[i-1][j-1]+1, f[i-1][j]+1, f[i-1][j-1] \mid A[i-1]=B[j-1]\}$
- 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
  - $f[0][j] = j$  ( $j = 0, 1, 2, \dots, n$ )
  - $f[i][0] = i$  ( $i = 0, 1, 2, \dots, m$ )

### 动态规划组成部分四：计算顺序



- $f[0][0], f[0][1], \dots, f[0][n]$
- $f[1][0], f[1][1], \dots, f[1][n]$
- ...
- $f[m][0], f[m][1], \dots, f[m][n]$
- 答案是  $f[m][n]$
- 时间复杂度（计算步数） $O(MN)$ ，空间复杂度（数组大小） $O(MN)$

录制中... 00:43:54

优化空间至  $O(N)$   
可以用滚动数组优化空间至  $O(N)$

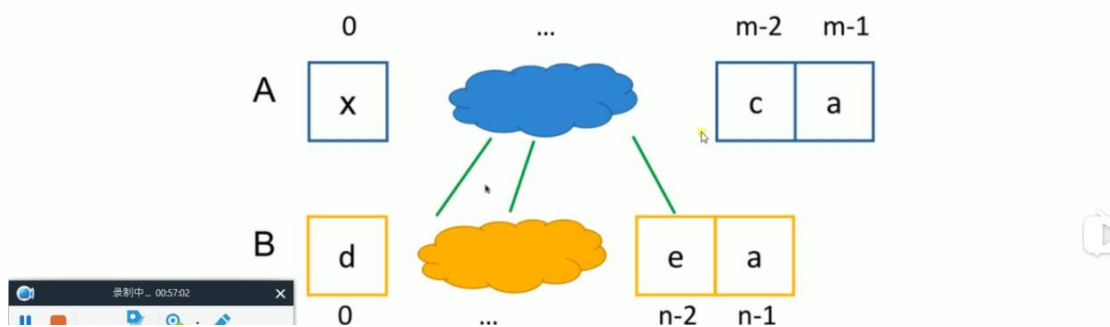
### LintCode 118 Distinct Subsequences



- 题意：
- 给定两个字符串  $A[0..m-1]$ ,  $B[0..n-1]$
- 问 B 在 A 中出现多少次（可以不连续）
- 例子
- 输入：A="rabbbit", B="rabbit"
- 输出：3
  - rabbbit
  - rabbbit
  - rabbbit

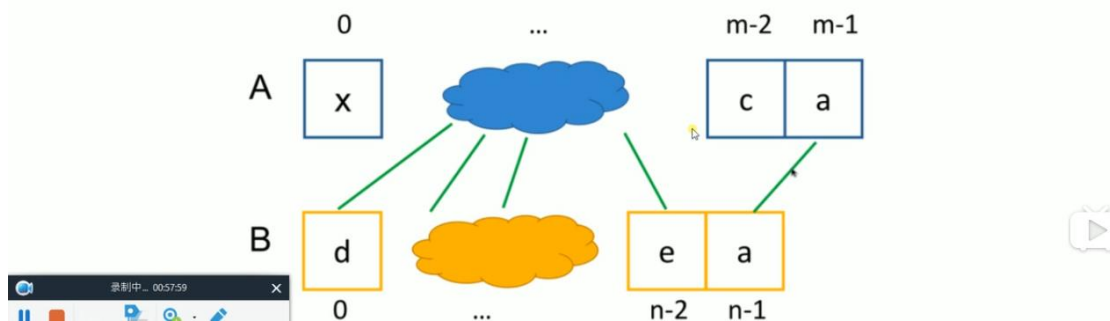
## 题目分析

- 双序列型动态规划
- **B**在**A**中出现多少次（可以不连续）
- 如果至少出现一次，那么**A**和**B**的最长公共子串就是**B**，而且也不能更长
- 用最长公共子串的思路：对应对子



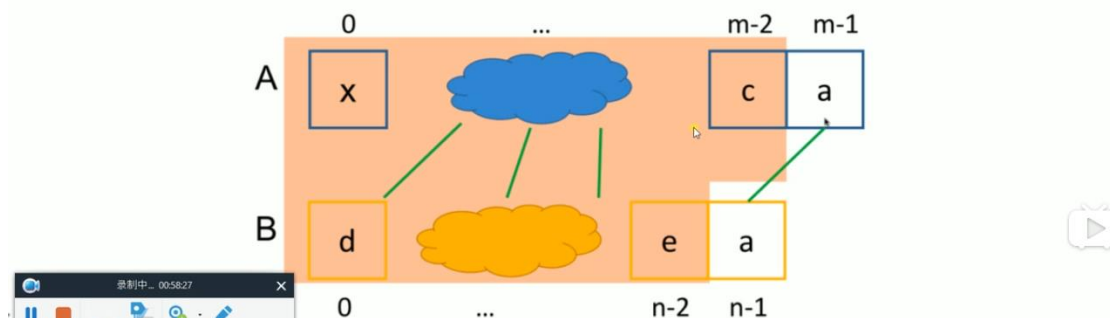
## 题目分析

- 但不同的是，**B**的每一个字符都必须出现在一个对子里
- 所以这题的“最后一步”以**B**为出发点



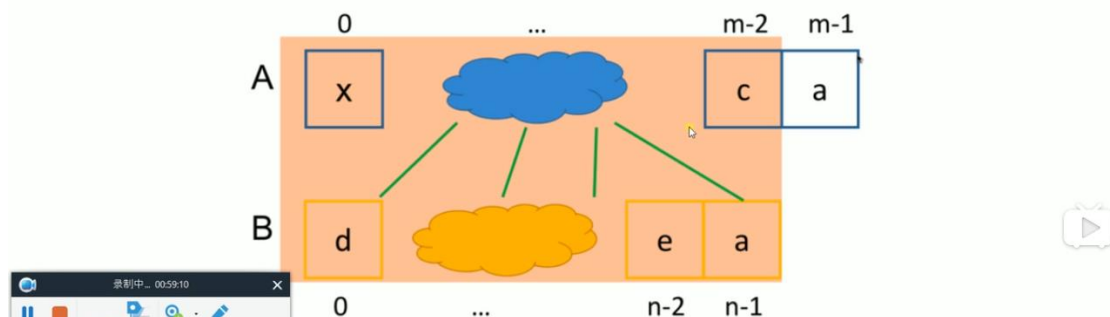
情况1： $B[n-1] = A[m-1]$ ，结成对子

求 $B[0..n-2]$ 在 $A[0..m-2]$ 中出现多少次



情况2： $B[n-1]$ 不和 $A[m-1]$ 结成对子

求 $B[0..n-1]$ 在 $A[0..m-2]$ 中出现多少次



- 要求 $B[0..n-1]$ 在 $A[0..m-1]$ 中出现多少次
- 需要知道 $B[0..n-1]$ 在 $A[0..m-2]$ 中出现多少次，以及 $B[0..n-2]$ 在 $A[0..m-2]$ 中出现多少次
- 子问题
- 状态：设 $f[i][j]$ 为B前j个字符 $B[0..j-1]$ 在A前i个字符 $A[0..i-1]$ 中出现多少次

## 动态规划组成部分二：转移方程

九章算法

- 设  $f[i][j]$  为 B 前 j 个字符  $B[0..j-1]$  在 A 前 i 个字符  $A[0..i-1]$  中出现多少次
- 要求  $f[m][n]$

$$f[i][j] = f[i-1][j-1] | A[i-1]=B[j-1] + f[i-1][j]$$

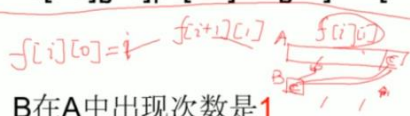
情况1：  $B[j-1] = A[i-1]$ ，结成对子

情况2：  $B[j-1]$  不和  $A[i-1]$  结成对子

## 动态规划组成部分三：初始条件和边界情况

九章算法

- 设  $f[i][j]$  为 B 前 j 个字符  $B[0..j-1]$  在 A 前 i 个字符  $A[0..i-1]$  中出现多少次
- 转移方程： $f[i][j] = f[i-1][j-1] | A[i-1]=B[j-1] + f[i-1][j]$
- 初始条件：
  - 如果 B 是空串，B 在 A 中出现次数是 1
  - $f[i][0] = 1$  ( $i = 0, 1, 2, \dots, m$ )
  - 如果 A 是空串而 B 不是空串，B 在 A 中出现次数是 0
  - $f[0][j] = 0$  ( $j = 1, 2, \dots, n$ )



## 动态规划组成部分四：计算顺序

九章算法

- 初始化  $f[0][0], f[0][1], \dots, f[0][n]$
- $f[1][0], f[1][1], \dots, f[1][n]$
- ...
- $f[m][0], f[m][1], \dots, f[m][n]$
- 答案是  $f[m][n]$
- 时间复杂度（计算步数）  $O(MN)$ ，空间复杂度（数组大小）  $O(MN)$



## LintCode 154 Regular Expression Matching



- 题意：
- 给定两个字符串A, B
- B是一个正则表达式，里面可能含有'.'和'\*'
  - '.' 可以匹配任何单个字符
  - '\*' 可以匹配0个或多个前一个字符
- 问A和B是否匹配
- 例子：
  - isMatch("aa","a") → false
  - isMatch("aa","aa") → true
  - isMatch("aaa","aa") → false
  - isMatch("aa","a\*") → true
  - isMatch("aa",".\*") → true
  - isMatch("ab",".\*") → true
  - isMatch("aab","c\*a\*b") → true


### 动态规划组成部分一：确定状态



- 双序列型动态规划
- 设A长度是m, B长度是n
- 现在我们考虑A和B如何匹配
- 最后一步：关注最后的字符
- 主要取决于正则表达式B中最后的字符B[n-1]是什么

### 动态规划组成部分一：确定状态



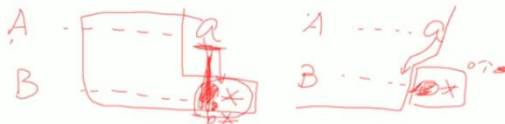
- 如果B[n-1]是一个正常字符（非.非\*），则如果A[m-1]=B[n-1]，能否匹配取决于A[0..m-2]和B[0..n-2]是否匹配；否则不能匹配
- 如果B[n-1]是'.'，则A[m-1]一定是和'.'匹配，之后能否匹配取决于A[0..m-2]和B[0..n-2]是否匹配  

- 如果B[n-1]是'\*'，它代表B[n-2]=c可以重复0次或多次，它们是一个整体c\*，需要考虑A[m-1]是0个c，还是多个c中的最后一个
  - A[m-1]是0个c，能否匹配取决于A[0..m-1]和B[0..n-3]是否匹配
  - A[m-1]是多个c中的最后一个，能否匹配取决于A[0..m-2]和B[0..n-1]是否匹配
    - 这种情况必须A[m-1]=c或者c='.'



## 子问题

九章算法

- 要求A前m个字符和B前n个字符能否匹配，需要知道A前m个字符和B前n-1个字符，A前m-1个字符和B前n个字符以及A前m个字符和B前n-2个字符能否匹配



- 子问题

- 状态：设 $f[i][j]$ 为A前i个字符 $A[0..i-1]$ 和B前j个字符 $B[0..j-1]$ 能否匹配

## 动态规划组成部分二：转移方程

九章算法

- 设 $f[i][j]$ 为A前i个字符 $A[0..i-1]$ 和B前j个字符 $B[0..j-1]$ 能否匹配

$$f[i][j] = \begin{cases} f[i-1][j-1], & \text{如果 } B[j-1]='.' \text{ 或者 } A[i-1]=B[j-1] \\ f[i][j-2] \text{ OR } (f[i-1][j] \text{ AND } (B[j-2]='.' \text{ OR } B[j-2]=A[i-1])), & \text{如果 } B[j-1]='*' \end{cases}$$

## 动态规划组成部分三：初始条件和边界情况

九章算法

- 设 $f[i][j]$ 为A前i个字符 $A[0..i-1]$ 和B前j个字符 $B[0..j-1]$ 能否匹配
- 空串和空正则表达式匹配： $f[0][0] = \text{TRUE}$
- 空的正则表达式不能匹配长度 $>0$ 的串  
-  $f[1][0] = \dots = f[m][0] = \text{FALSE}$
- 注意： $f[0][1..n]$ 也用动态规划计算，但是因为没有 $A[-1]$ ，所以只能用第二种情况中的 $f[i][j-2]$

- $f[0][0], f[0][1], \dots, f[0][n]$
- $f[1][0], f[1][1], \dots, f[1][n]$
- ...
- $f[m][0], f[m][1], \dots, f[m][n]$
- 答案是  $f[m][n]$
- 时间复杂度（计算步数） $O(MN)$ ，空间复杂度（数组大小） $O(MN)$

优化空间至 $O(N)$

## LintCode 192 Wildcard Matching

- 题意：
- 给定两个字符串A, B
- B是一个正则表达式，里面可能含有'?'和'\*'
  - '?' 可以匹配任何单个字符
  - '\*' 可以匹配0个或多个任意字符
- 问A和B是否匹配
- 例子：
  - `isMatch("aa","a")` → false
  - `isMatch("aa","aa")` → true
  - `isMatch("aaa","aa")` → false
  - `isMatch("aa","*")` → true
  - `isMatch("aa","a*")` → true
  - `isMatch("ab","?")` → true
  - `isMatch("ab","c*a*b")` → false

## 动态规划组成部分一：确定状态

- 双序列型动态规划
- 和Regular Expression Matching很类似，因为'.'和'?'作用相同，但是这题中'\*'可以匹配0个或多个任意字符
- 设A长度是m, B长度是n
- 现在我们考虑A和B如何匹配
- 关注最后的字符
- 主要取决于Wildcard中B[n-1]是什么

## 动态规划组成部分一：确定状态

九章算法

- 如果 $B[n-1]$ 是一个正常字符（非?非\*），则如果 $A[m-1]=B[n-1]$ ，能否匹配取决于 $A[0..m-2]$ 和 $B[0..n-2]$ 是否匹配；否则不能匹配
- 如果 $B[n-1]$ 是'?'，则 $A[m-1]$ 一定是和'?'匹配，之后能否匹配取决于 $A[0..m-2]$ 和 $B[0..n-2]$ 是否匹配
- 如果 $B[n-1]$ 是'\*'，它可以匹配0个或任意多个字符，需要考虑 $A[m-1]$ 有没有被这个'\*'匹配
  - $A[m-1]$ 不被'\*'匹配，能否匹配取决于 $A[0..m-1]$ 和 $B[0..n-2]$ 是否匹配
  - $A[m-1]$ 被'\*'匹配，能否匹配取决于 $A[0..m-2]$ 和 $B[0..n-1]$ 是否匹配

## 子问题

九章算法

- 要求A前m个字符和B前n个字符能否匹配，需要知道A前m-1个字符和B前n-1个字符，A前m个字符和B前n-1个字符以及A前m-1个字符和B前n个字符能否匹配
- 子问题
- 状态：设 $f[i][j]$ 为A前i个字符 $A[0..i-1]$ 和B前j个字符 $B[0..j-1]$ 能否匹配

## 动态规划组成部分二：转移方程

九章算法

- 设 $f[i][j]$ 为A前i个字符 $A[0..i-1]$ 和B前j个字符 $B[0..j-1]$ 能否匹配

$$f[i][j] = \begin{cases} f[i-1][j-1], & \text{如果 } B[j-1]='?' \text{ 或者 } A[i-1]=B[j-1] \\ f[i-1][j] \text{ OR } f[i][j-1], & \text{如果 } B[j-1]='*' \end{cases}$$

### 动态规划组成部分三：初始条件和边界情况



- 设 $f[i][j]$ 为A前i个字符 $A[0..i-1]$ 和B前j个字符 $B[0..j-1]$ 能否匹配
- 空串和空Wildcard匹配： $f[0][0] = \text{TRUE}$
- 空的Wildcard不能匹配长度 $>0$ 的串
  - $f[1][0] = \dots = f[m][0] = \text{FALSE}$
- $f[0][1..n]$ 也用动态规划计算，但是因为缺少 $A[-1]$ ，所以只能用第二种情况中的 $f[i][j-1]$

### 动态规划组成部分四：计算顺序



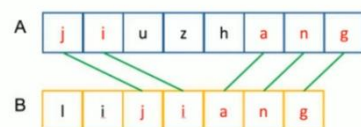
- $f[0][0], f[0][1], \dots, f[0][n]$
- $f[1][0], f[1][1], \dots, f[1][n]$
- ...
- $f[m][0], f[m][1], \dots, f[m][n]$
- 答案是 $f[m][n]$
- 时间复杂度（计算步数） $O(MN)$ ，空间复杂度（数组大小） $O(MN)$

录制中... 01:31:57  
优化空间至 $O(N)$

### 双序列型动态规划总结



- 两个一维序列/字符串
- 突破口
  - 串A和串B的最后一个字符是否匹配
  - 是否需要串A/串B的最后一个字符
  - 缩减问题规模
- 数组下标表示序列A前i个，序列B前j个： $f[i][j]$
- 初始条件和边界情况
  - 空串如何处理
  - 计数型(情况1+情况2+...)以及最值型( $\min/\max\{\text{情况1}, \text{情况2}, \dots\}$ )
- 匹配的情况下别忘了+1(操作数多1次，匹配长度多1)



录制中... 01:39:14  
匹配的情况下别忘了+1(操作数多1次，匹配长度多1)

- 题意：
- 给定T个01串 $S_1, S_2, \dots, S_{T-1}$
- 现有m个0, n个1
- 问最多能组成多少个给定01串
- 每个串最多组成一次
- 例子：
- 输入：["10", "0001", "111001", "1", "0"], m = 5, n = 3
- 输出：4 ("10", "0001", "1", "0")

### 动态规划组成部分一：确定状态

- 最后一步：最优策略组成了最多的01串，其中有没有最后一个字符串 $S_{T-1}$
- 情况1：没有 $S_{T-1}$ 
  - 需要知道前T-1个01串中，用m个0和n个1最多能组成多少个01串
- 情况2：有 $S_{T-1}$ 
  - 设第T-1个01串中有 $a_{T-1}$ 个0,  $b_{T-1}$ 个1
  - 需要知道前T-1个01串中，用 $m-a_{T-1}$ 个0和 $n-b_{T-1}$ 个1最多能组成多少个01串
- 子问题
- 0和1的个数在变化，如何记录？
  - 直接放入状态

状态：设 $f[i][j][k]$ 为前i个01串最多能有多少个被j个0和k个1组成

### 动态规划组成部分二：转移方程

- 设 $f[i][j][k]$ 为前i个01串最多能有多少个被j个0和k个1组成
- 设 $S_i$ 中有 $a_i$ 个0,  $b_i$ 个1

$$f[i][j][k] = \max\{f[i-1][j][k], f[i-1][j-a_{i-1}][k-b_{i-1}] + 1 \mid j \geq a_{i-1} \text{ AND } k \geq b_{i-1}\}$$

前i个01串最多能有多少个被j个0和k个1组成

前i-1个01串最多能有多少个被j个0和k个1组成

前i-1个01串最多能有多少个被j- $a_{i-1}$ 个0和k- $b_{i-1}$ 个1组成，再加上 $S_{i-1}$



### 动态规划组成部分三：初始条件和边界情况



- 设  $f[i][j][k]$  为前  $i$  个 01 串最多能有多少个被  $j$  个 0 和  $k$  个 1 组成
- 设  $S_i$  中有  $a_i$  个 0,  $b_i$  个 1
- $f[i][j][k] = \max\{f[i-1][j][k], f[i-1][j-a_{i-1}][k-b_{i-1}] + 1 \mid j \geq a_{i-1} \text{ AND } k \geq b_{i-1}\}$
- 初始条件： $f[0][0 \sim m][0 \sim n] = 0$ 
  - 无论有多少 0 和 1, 前 0 个 01 串中最多能组成 0 个
- 边界情况： $f[i-1][j-a_{i-1}][k-b_{i-1}] + 1$  必须  $j \geq a_{i-1} \text{ AND } k \geq b_{i-1}$

### 动态规划组成部分四：计算顺序



- $f[0][0][0], f[0][0][1], \dots, f[0][0][n], f[0][1][0], \dots, f[0][1][n], \dots, f[0][m][n]$
- $f[1][0][0], f[1][0][1], \dots, f[1][0][n], f[1][1][0], \dots, f[1][1][n], \dots, f[1][m][n]$
- ...
- $f[T][0][0], f[T][0][1], \dots, f[T][0][n], f[T][1][0], \dots, f[T][1][n], \dots, f[T][m][n]$
- 答案是  $f[T][m][n]$
- 时间复杂度： $O(Tmn)$ , 空间复杂度： $O(Tmn)$ , 可以用滚动数组优化至  $O(mn)$