最小编辑距离

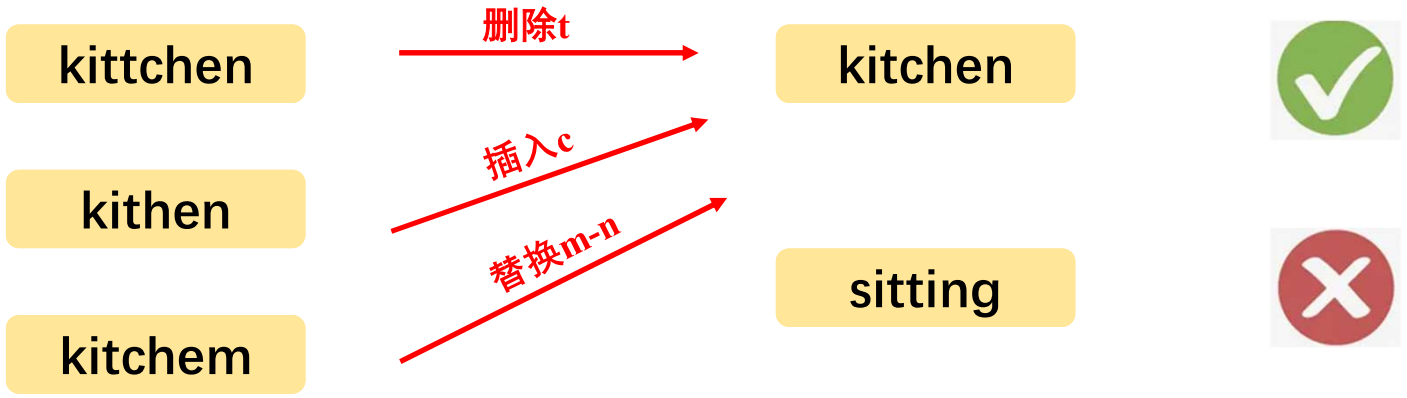
编辑距离是针对二个字符串的差异程度的量化量测，量测方式是看至少需要多少次的处理才能将一个字符串变成另一个字符串。编辑操作中，许可的操作一般包括将一个字符**替换**成另一个字符，**插入**一个字符，**删除**一个字符。一般来说，编辑距离越小，两个串的相似度越大。由一个字符串转换成另外一个字符串所需的最少编辑次数，也被称之为最小编辑距离。

编辑距离由俄罗斯科学家 Vladimir Levenshtein 在1965年提出，也因此而得名 Levenshtein Distance。可以用在信息论、语言学和计算机科学领域，用于DNA分析、拼字检测等等，核心是比较两者的相似性。

最小编辑距离

问题背景：

输入法自动更正

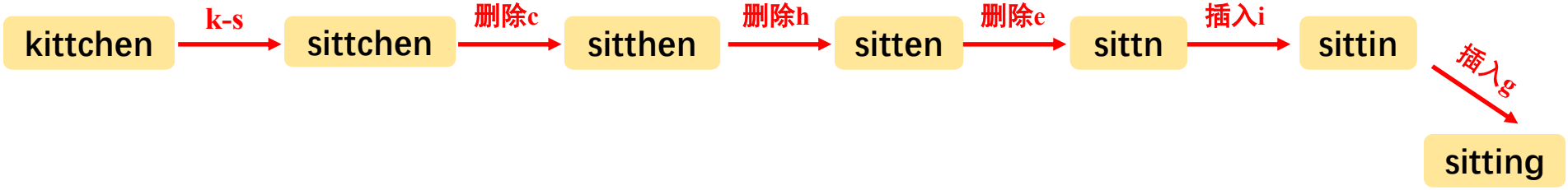


问题：如何衡量序列的相似程度

最小编辑距离

方案1

编辑操作：6次



方案2

编辑操作：5次

问题：如何求解最少的编辑操作次数（最小编辑距离）

最小编辑距离

形式化表示：

输入：长度为n的字符串X：$\{a_1, a_2, \dots, a_n\}$，长度为m的字符串Y：$\{b_1, b_2, \dots, b_m\}$

输出：求出一组编辑操作$\mathbf{D}=< d_1, d_2, \dots, d_l >$

令

优化目标

$$\mathbf{min|D|}$$

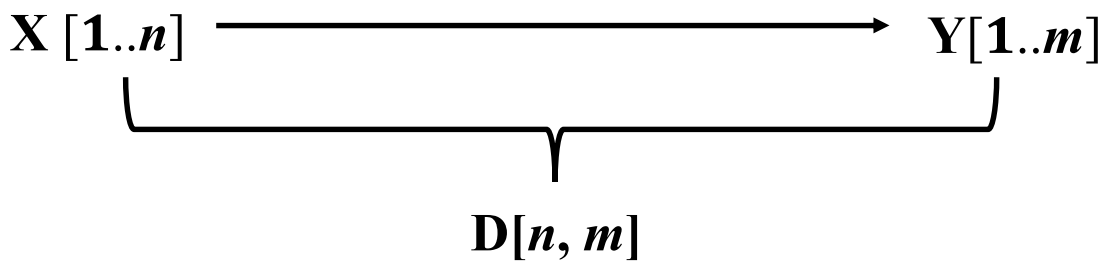
约束条件

s.t. **字符串经过了D次的操作之后满足**

$$\{a_1, a_2, \dots, a_n\}= \{b_1, b_2, \dots, b_m\}$$

递归结构

数组D[i, j]：表示字符串X [1..i]变成字符串Y[1..j]的最少操作次数



$X=\{a_1, a_2, \dots, a_n\}$ $(1 \le i_1 < i_2, \dots, i_l \le n)$

$Y=\{b_1, b_2, \dots, b_m\}$ $(1 \le j_1 < j_2, \dots, j_l \le m)$

X=Y

递归结构

最长公共子序列

对于任意两条序列X [1..*m*]和Y[1..*n*]的最长公共子序列的长度

$X=\{x_1, x_2, \dots, x_m\}$
$Y=\{y_1, y_2, \dots, y_n\}$ → $X=\{x_1, x_2, \dots, x_{i-1}, x_i\}$ $Y=\{y_1, y_2, \dots, y_{j-1}, y_j\}$ → $C[m, n]$

$\max|Z|$

- $X=\{x_1, x_2, \dots, x_m\}$ $Y=\{y_1, y_2, \dots, y_n\}$ → $C[m, n]=C[m-1, n-1]+1$ ，$x_m = y_n$
- $X=\{x_1, x_2, \dots, x_m\}$ $Y=\{y_1, y_2, \dots, y_{n-1}, y_n\}$ → $C[m, n]=C[m, n-1]+0$ ，$x_m \neq y_n$
- $X=\{x_1, x_2, \dots, x_{m-1}, x_m\}$ $Y=\{y_1, y_2, \dots, y_n\}$ → $C[m, n]=C[m-1, n]+0$ ，$x_m \neq y_n$

最优子结构性质

最小编辑距离

1. 考察末尾元素

2. 操作步骤：

   插入
   删除
   替换 → 递归式？

在最小编辑距离问题中，只操作其中的一条序列，另一条序列固定不变。

递归结构

固定Y序列，只操作X序列

X={A, B, C, B, D, A, B}

Y={B, D, C, A, B, A}

min

A={A, B, C, B, D, A, ~~B~~}

B={B, D, C, A, B, A}

删除B

A={A, B, C, B, D, A, B, A}

B={B, D, C, A, B, A}

插入A

A={A, B, C, B, D, A, A}

B={B, D, C, A, B, A}

替换B-A

递归结构

对于任意两条序列A=$\{a_1, a_2, \dots, a_i\}$和B=$\{b_1, b_2, \dots, b_j\}$

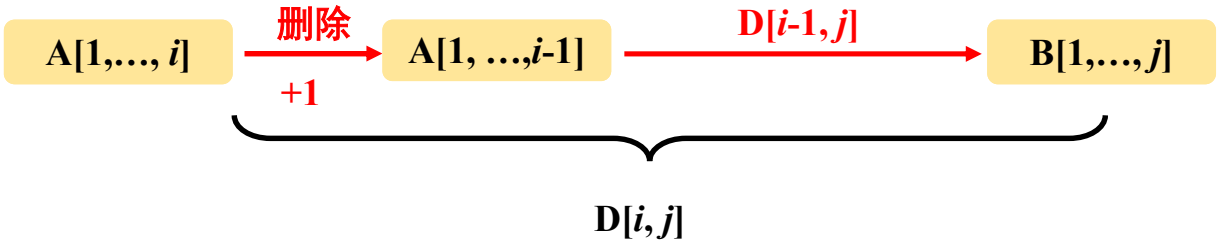
D[*i*, *j*]

A=$\{a_1, a_2, \dots, a_{i-1}, a_i\}$

B=$\{b_1, b_2, \dots, b_j\}$

子问题：

D[*i*, *j*] ⟶ **D[*i*-1, *j*]**



在删除操作中：**D[*i*, *j*]** = **D[*i*-1, *j*]** + 1

最优子结构性质

递归结构

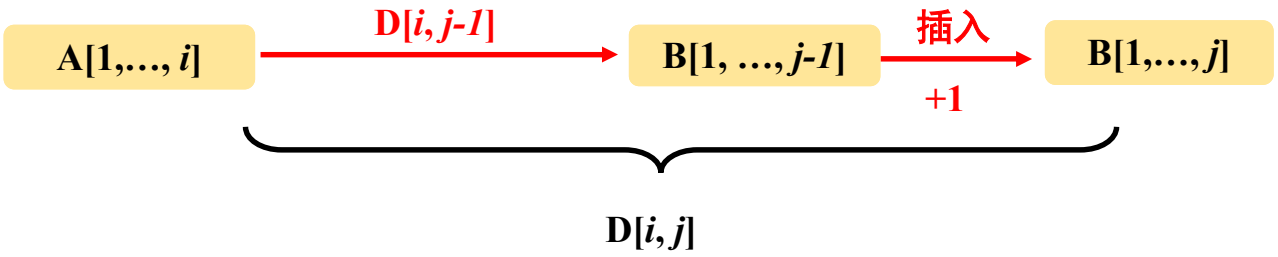
D[*i*, *j*]

A={a_1, a_2, … ,a_{i-1}, a_i, b_j}

B={b_1, b_2, … , b_{j-1}, b_j}

子问题：

D[*i*, *j*] ⟶ **D[*i*, *j-1*]**



在插入操作中：**D[*i*, *j*] = D[*i*, *j-1*] + 1**

最优子结构性质

递归结构

$D[i,j]$

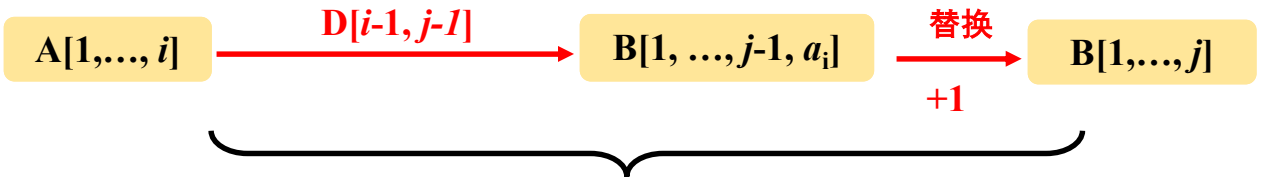
$A=\{a_1, a_2, \cdots, a_{i-1}, a_i\}$

$B=\{b_1, b_2, \cdots, b_{j-1}, b_j\}$

子问题：

$D[i,j] \longrightarrow D[i-1,j-1]$

$$\begin{cases} a_i \neq b_j, & \text{操作为1} \\ a_i = b_j, & \text{操作为0} \end{cases}$$



$D[i,j]$

在替换操作中$D[i,j] = D[i-1,j-1] +$

$$\begin{cases} 1, \text{ if } a_i \neq b_j, \\ 0, \text{ if } a_i = b_j \end{cases}$$

最优子结构性质

如果$a_i = b_j$时，则不需要替换，操作步骤为0，否则则需要替换，操作步骤为1

递归结构

综上所述：最小编辑距离的递归式

$$D[i,j] = \min \begin{cases} D[i-1,j] + 1 & \text{删除} \\ D[i,j-1] + 1 & \text{插入} \\ D[i-1,j-1] + \begin{cases} 1, \text{ if } a_i \neq b_j, \\ 0, \text{ if } a_i = b_j \end{cases} & \text{替换} \end{cases}$$

X \ Y	$j=0$	$j=1$	$j=2$	…	$j=m$
$i=0$	0	1	2		m
$i=1$	1				
$i=2$	2				
…				D[i-1, j-1]	D[i-1, j]
$i=n$	n			D[i, j-1]	D[i, j]

插入

删除

- m次操作将空串变成长度为j的字符串
- n次操作将长度为i的字符串变成空串

实例

X={A, G, G, C, A, T, A, G, C, T}

Y={C, A, G, T, A, T, C, C, T}

***m*=length(X)=10**

***n*=length(Y)=9**

求解将序列X变成序列Y的最小编辑距离？

实例

构建D[*i*, *j*]数组

D[1,1] =min { D[0, 1] +1=2 ; D[1, 0] +1=2 ; D[0, 0] +1=1 }

$A \neq C,\ x_1 \neq y_1$

D[1,2] =min { D[0, 2] +1=3 ; D[1, 1] +1=2 ; D[0, 1] +0=1 }

$A = A,\ x_1 = y_2$

D[1,3] =min { D[0, 3] +1=4 ; D[1, 2] +1=2 ; D[0, 2] +1=3 }

$A \neq G,\ x_1 \neq y_3$

	Y	0	C	A	G	T	A	T	C	C	T
X	D[*i*, *j*]	*j* = 0	*j* = 1	*j* = 2	*j* = 3	*j* = 4	*j* = 5	*j* = 6	*j* = 7	*j* = 8	*j* = 9
0	*i* = 0	0	1	2	3	4	5	6	7	8	9
A	*i* = 1	1	C[1,1]	C[1,2]	C[1,3]						
G	*i* = 2	2									
G	*i* = 3	3									
C	*i* = 4	4									
A	*i* = 5	5									
T	*i* = 6	6									
A	*i* = 7	7									
G	*i* = 8	8									
C	*i* = 9	9									
T	*i* = 10	10									

以此类推……

$$D[i,j] = \min \begin{cases} D[i-1,j] + 1 & \text{删除} \\ D[i,j-1] + 1 & \text{插入} \\ D[i-1,j-1] + \begin{cases} 1, \text{if } a_i \neq b_j, \\ 0, \text{if } a_i = b_j \end{cases} & \text{替换} \end{cases}$$

实例

构建D[*i*, *j*]数组

	Y	0	C	A	G	T	A	T	C	C	T
X	D[*i*, *j*]	*j* = 0	*j* = 1	*j* = 2	*j* = 3	*j* = 4	*j* = 5	*j* = 6	*j* = 7	*j* = 8	*j* = 9
0	*i* = 0	0	1	2	3	4	5	6	7	8	9
A	*i* = 1	1	1	1	2	3	4	5	6	7	8
G	*i* = 2	2	2	2	1	2	3	4	5	6	7
G	*i* = 3	3	3	3	2	2	3	4	5	6	7
C	*i* = 4	4	3	4	3	3	3	4	4	5	6
A	*i* = 5	5	4	3	4	4	3	4	5	5	6
T	*i* = 6	6	5	4	4	4	4	3	4	5	5
A	*i* = 7	7	6	5	5	5	4	4	4	5	6
G	*i* = 8	8	7	6	5	6	5	5	5	5	6
C	*i* = 9	9	8	7	6	6	6	6	5	5	6
T	*i* = 10	10	9	8	7	6	7	6	6	6	5

实例

回溯最小编辑路径　D[*i*, *j*] = 5，即：X和Y序列的最小编辑距离为5

	Y	0	C	A	G	T	A	T	C	C	T
X	D[*i*, *j*]	*j* = 0	*j* = 1	*j* = 2	*j* = 3	*j* = 4	*j* = 5	*j* = 6	*j* = 7	*j* = 8	*j* = 9
0	*i* = 0	0	1	2	3	4	5	6	7	8	9
A	*i* = 1	1	1	1	2	3	4	5	6	7	8
G	*i* = 2	2	2	2	1	2	3	4	5	6	7
G	*i* = 3	3	3	3	2	2	3	4	5	6	7
C	*i* = 4	4	3	4	3	3	3	4	4	5	6
A	*i* = 5	5	4	3	4	4	3	4	5	5	6
T	*i* = 6	6	5	4	4	4	4	3	4	5	5
A	*i* = 7	7	6	5	5	5	4	4	4	5	6
G	*i* = 8	8	7	6	5	6	5	5	5	5	6
C	*i* = 9	9	8	7	6	6	6	6	5	5	6
T	*i* = 10	10	9	8	7	6	7	6	6	6	5

↖：替换；

←：插入；

↑：删除；

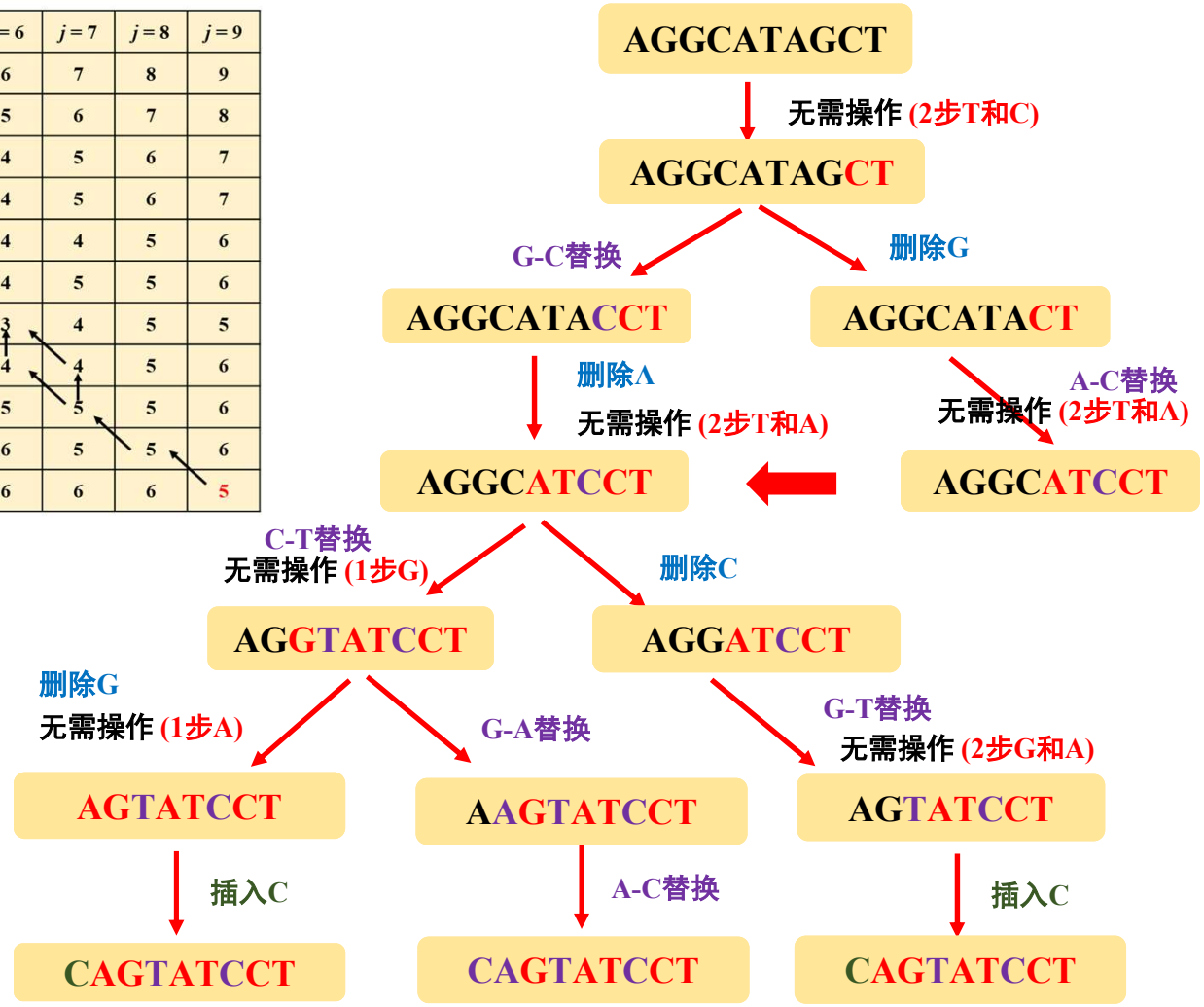
实例 回溯最小编辑路径

	Y	0	C	A	G	T	A	T	C	C	T
X	D[i,j]	j=0	j=1	j=2	j=3	j=4	j=5	j=6	j=7	j=8	j=9
0	i=0	0	1	2	3	4	5	6	7	8	9
A	i=1	1	1	1	2	3	4	5	6	7	8
G	i=2	2	2	2	1	2	3	4	5	6	7
G	i=3	3	3	3	2	2	3	4	5	6	7
C	i=4	4	3	4	3	3	3	4	4	5	6
A	i=5	5	4	3	4	4	3	4	5	5	6
T	i=6	6	5	4	4	4	4	3	4	5	5
A	i=7	7	6	5	5	5	4	4	4	5	6
G	i=8	8	7	6	5	6	5	5	5	5	6
C	i=9	9	8	7	6	6	6	6	5	5	6
T	i=10	10	9	8	7	6	7	6	6	6	5

编辑操作：5次

X={A, G, G, C, A, T, A, G, C, T}

Y={C, A, G, T, A, T, C, C, T}



拓展--带权最小编辑距离

最小编辑距离只求解了需要操作的次数，如果对删除、插入和替换三种不同的操作，赋予不同的代价权重呢？

分别用w_1, w_2和w_3表示进行一次删除、插入和替换的代价

$$D[i,j] = \min \begin{cases} D[i\text{-}1,j] + 1 & \text{删除} \\ D[i,j\text{-}1] + 1 & \text{插入} \\ D[i\text{-}1,j\text{-}1] + \begin{cases} 1, \text{if } a_i \neq b_j, \\ 0, \text{if } a_i = b_j \end{cases} & \text{替换} \end{cases}$$

递归式

$$s[0,0]=0$$

$$s[i,0]=w_1 * i,\ 1 \le i \le n$$

$$s[0,j]=w_2 * j,\ 1 \le j \le m$$

$$s[i,j]=\min \begin{cases} s[i\text{-}1,j] + w_1 \\ s[i,j\text{-}1] + w_2 \\ s[i\text{-}1,j\text{-}1] + w_3/0 \end{cases}$$

总结--动态规划算法设计要点

- 引入参数来界定子问题的边界，注意子问题的重叠程度。
- 给出带界参数的优化函数定义与优化函数的递推关系，找到递推关系的初值。
- 判断该优化问题是否满足优化原则。
- 考虑是否需要标记函数。
- 采用自向上的实现技术，从最小的子问题开始迭代计算，计算中用备忘录保留优化函数和标记函数的值。
- 动态规划算法的时间复杂度是对所有子问题(备忘录)的计算工作量求和(可能需要追踪解的工作量)。
- 动态规划算法一般使用较多的存储空间，这往往成为限制动态规划算法使用的瓶颈因素。