Design and Analysis of Algorithms

Part II: Dynamic Programming

Lecture 14: Minimum Edit Distance

童咏昕

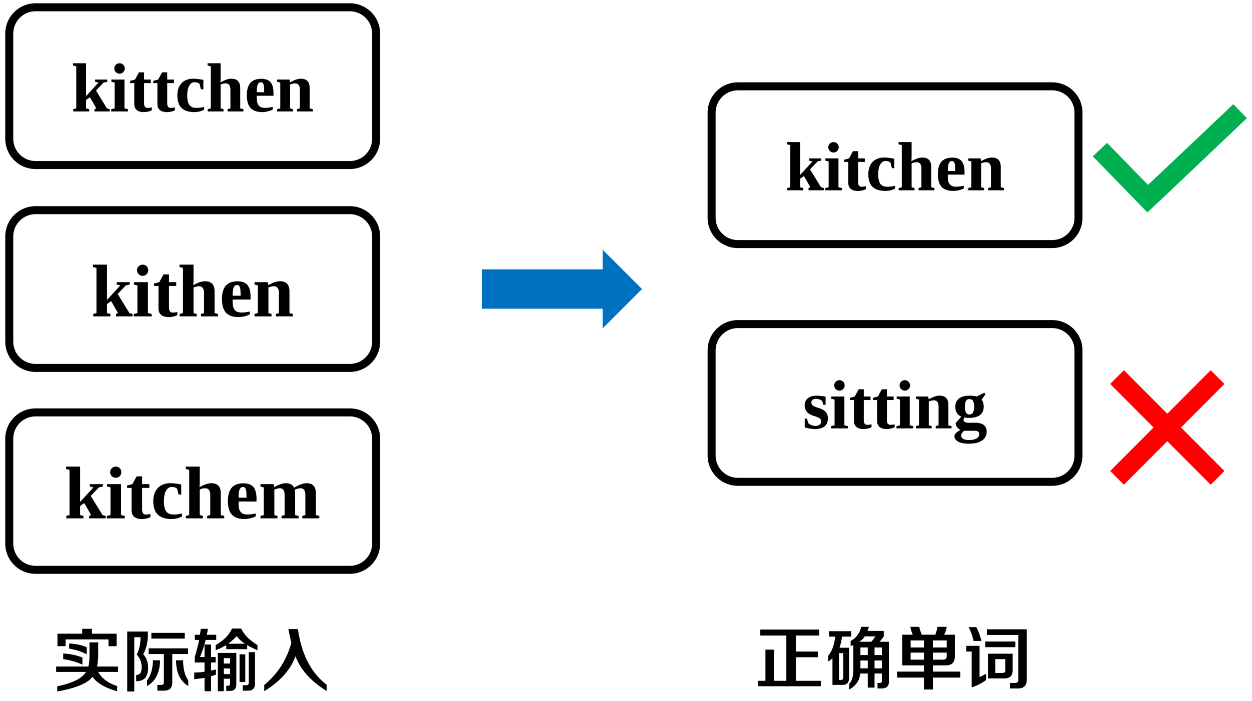
北京航空航天大学
计算机学院

动态规划篇概述

- 在算法课程第二部分“动态规划”主题中，我们将主要聚焦于如下经典问题：
  - 0-1 Knapsack (0-1背包问题)
  - Maximum Contiguous Subarray II (最大连续子数组 II)
  - Longest Common Subsequences (最长公共子序列)
  - Longest Common Substrings (最长公共子串)
  - Minimum Edit Distance (最小编辑距离)
  - Rod-Cutting (钢条切割)
  - Chain Matrix Multiplication (矩阵链乘法)

问题背景

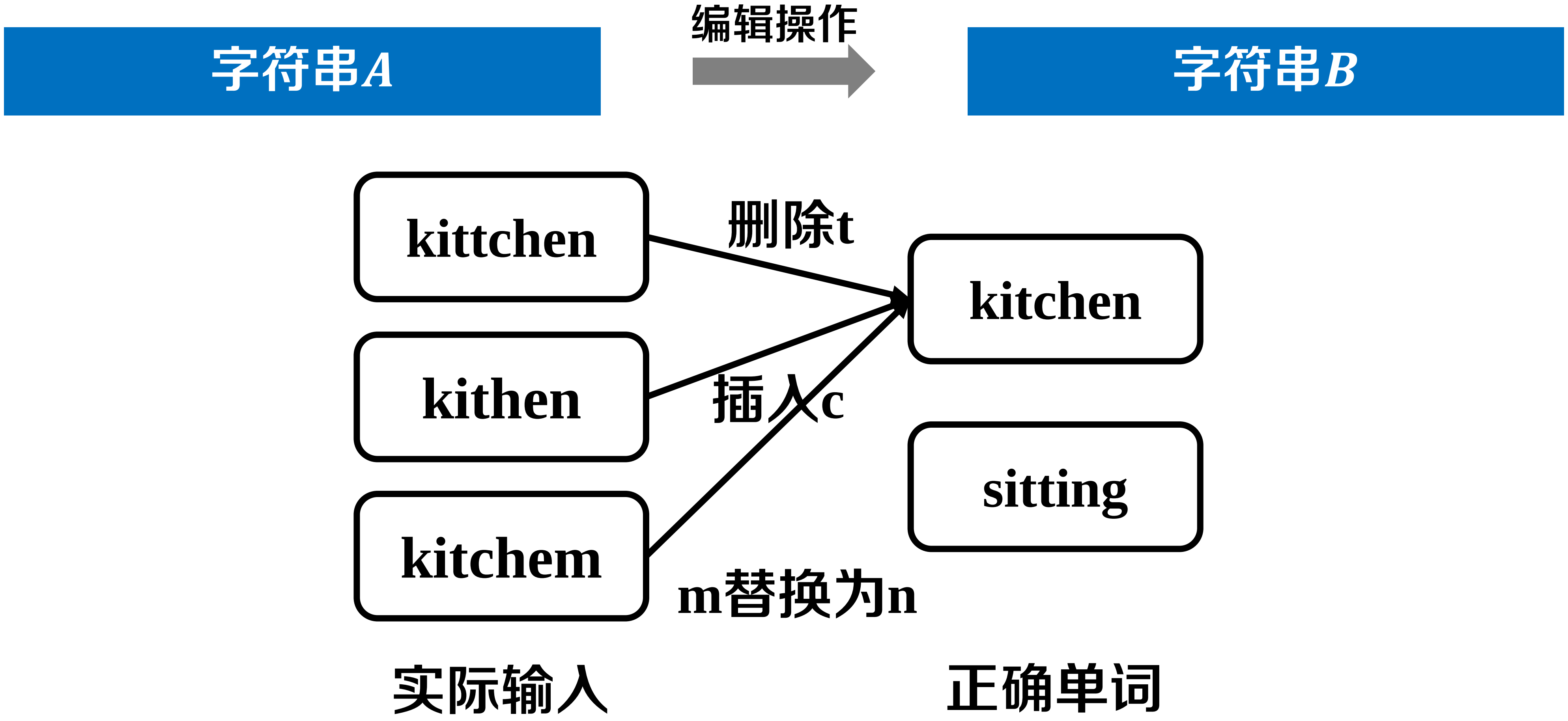
- 输入法自动更正



问题：如何衡量序列的相似程度？

编辑操作

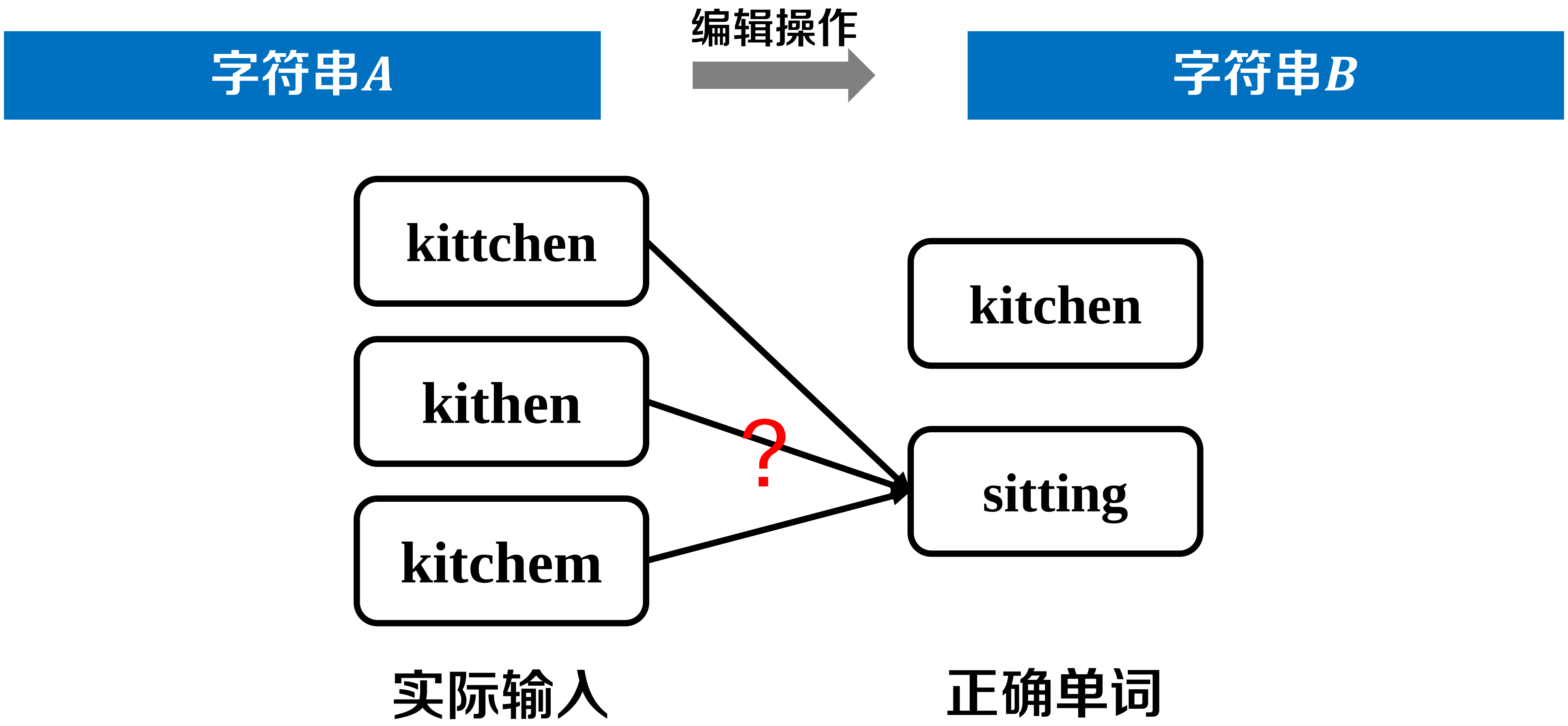
- 基本思想



- 编辑操作：删除、插入、替换

编辑操作

- 基本思想



- 编辑操作：删除、插入、替换

编辑操作示例

A = kittchen

编辑操作

B = sitting

操作名称	操作示例
删除	kittchen → kitchen
插入	kithen → kitchen
替换	kitchem → kitchen

- 方案1 kittchen $\xrightarrow{k\to s}$ sittchen $\xrightarrow{删除c}$ sitthen $\xrightarrow{删除h}$ sitten $\xrightarrow{删除e}$ sittn $\xrightarrow{插入i}$ sittin $\xrightarrow{插入g}$ sitting
- 方案2 kittchen $\xrightarrow{k\to s}$ sittchen $\xrightarrow{删除c}$ sitthen $\xrightarrow{删除h}$ sitten $\xrightarrow{e\to i}$ sittin $\xrightarrow{插入g}$ sitting

问题：如何求出最少的编辑操作数（最小编辑距离）？

问题定义

编辑距离问题

Minimum Edit Distance, MED

输入

- 长度为n的字符串s，长度为m的字符串t

输出

- 求出一组编辑操作$O = < e_1, e_2, \ldots e_d >$，令

优化目标

$$\min |O|$$

约束条件

$$s.t.\ 字符串s经过O的操作后满足s = t$$

问题结构分析

- 给出问题表示
  - $D[i,j]$：字符串$s[1..i]$变为$t[1..j]$的最小编辑距离

$$\boxed{s[1..i]}\ \underbrace{\xrightarrow{\hspace{6em}}}_{D[i,j]}\ \boxed{t[1..j]}$$

- 明确原始问题
  - $D[n,m]$：字符串$s[1..n]$变为$t[1..m]$的最小编辑距离

问题结构分析

↓

递推关系建立

↓

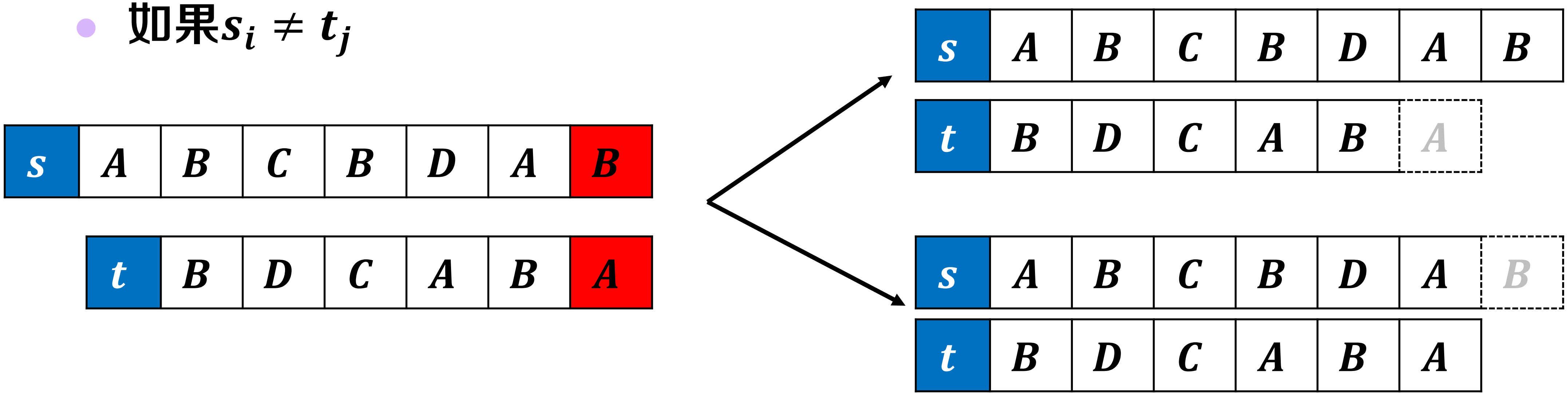
自底向上计算

↓

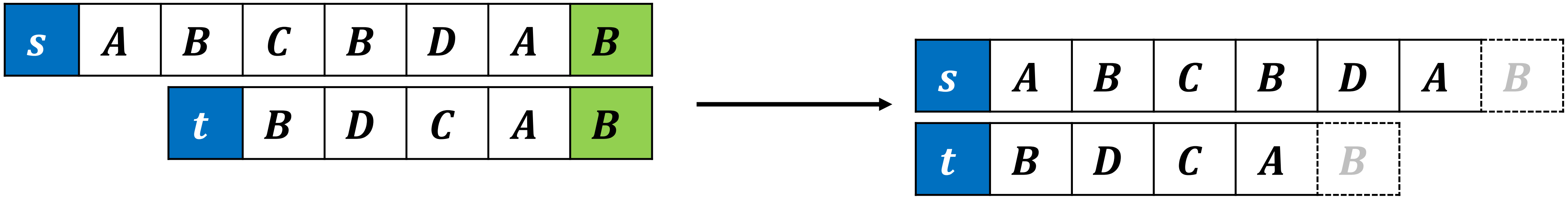
最优方案追踪

递推关系建立：回顾与启发

- 最长公共子序列
  - 如果$s_i \neq t_j$



  - 如果$s_i = t_j$



考察末尾元素

递推关系建立

- 考察末尾元素
  - 删除

s	A	B	C	B	D	A	B

t	B	D	C	A	B	A

→ ?

  - 插入

s	A	B	C	B	D	A	B	?

t	B	D	C	A	B	A

→ ?

  - 替换

							?
s	A	B	C	B	D	A	B

t	B	D	C	A	B	A

→ ?

问题结构分析 → 递推关系建立 → 自底向上计算 → 最优方案追踪

递推关系建立：分析最优（子）结构

- 考察末尾元素：删除

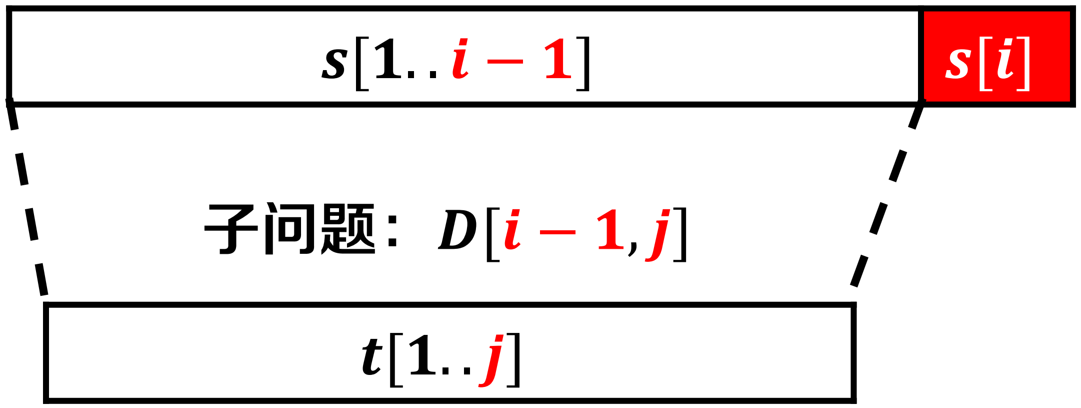
s	A	B	C	B	D	A	B

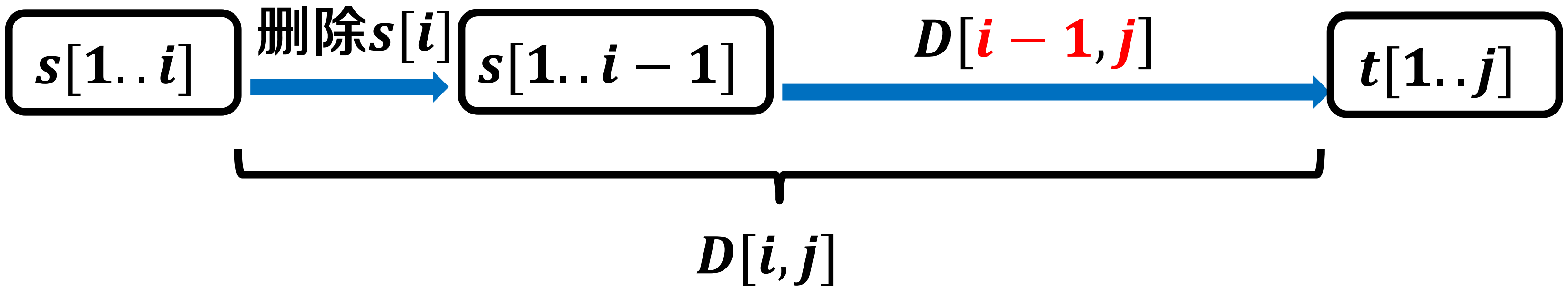
t	B	D	C	A	B	A

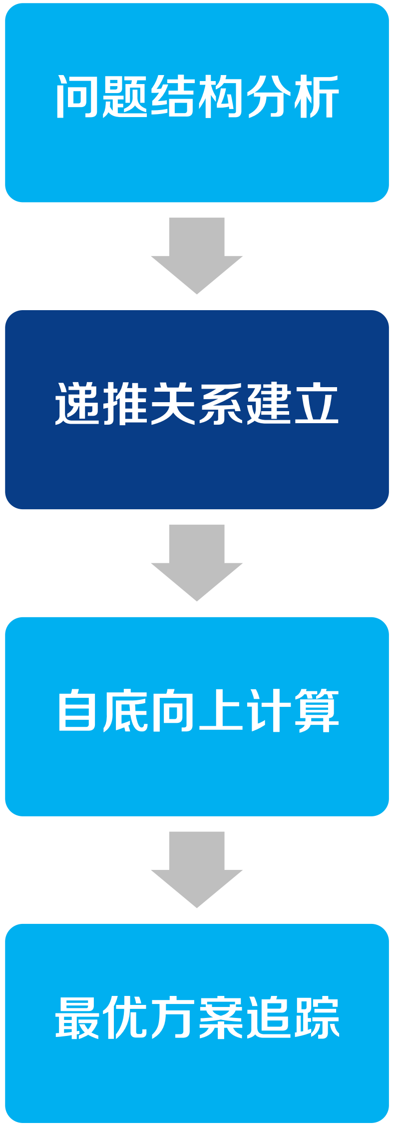
→

s	A	B	C	B	D	A	B

t	B	D	C	A	B	A







递推关系建立：分析最优（子）结构

- 考察末尾元素：删除

s	A	B	C	B	D	A	B

t	B	D	C	A	B	A

→

s	A	B	C	B	D	A	B

t	B	D	C	A	B	A

$s[1..i-1]$ $s[i]$

子问题：$D[i-1,j]$

$t[1..j]$

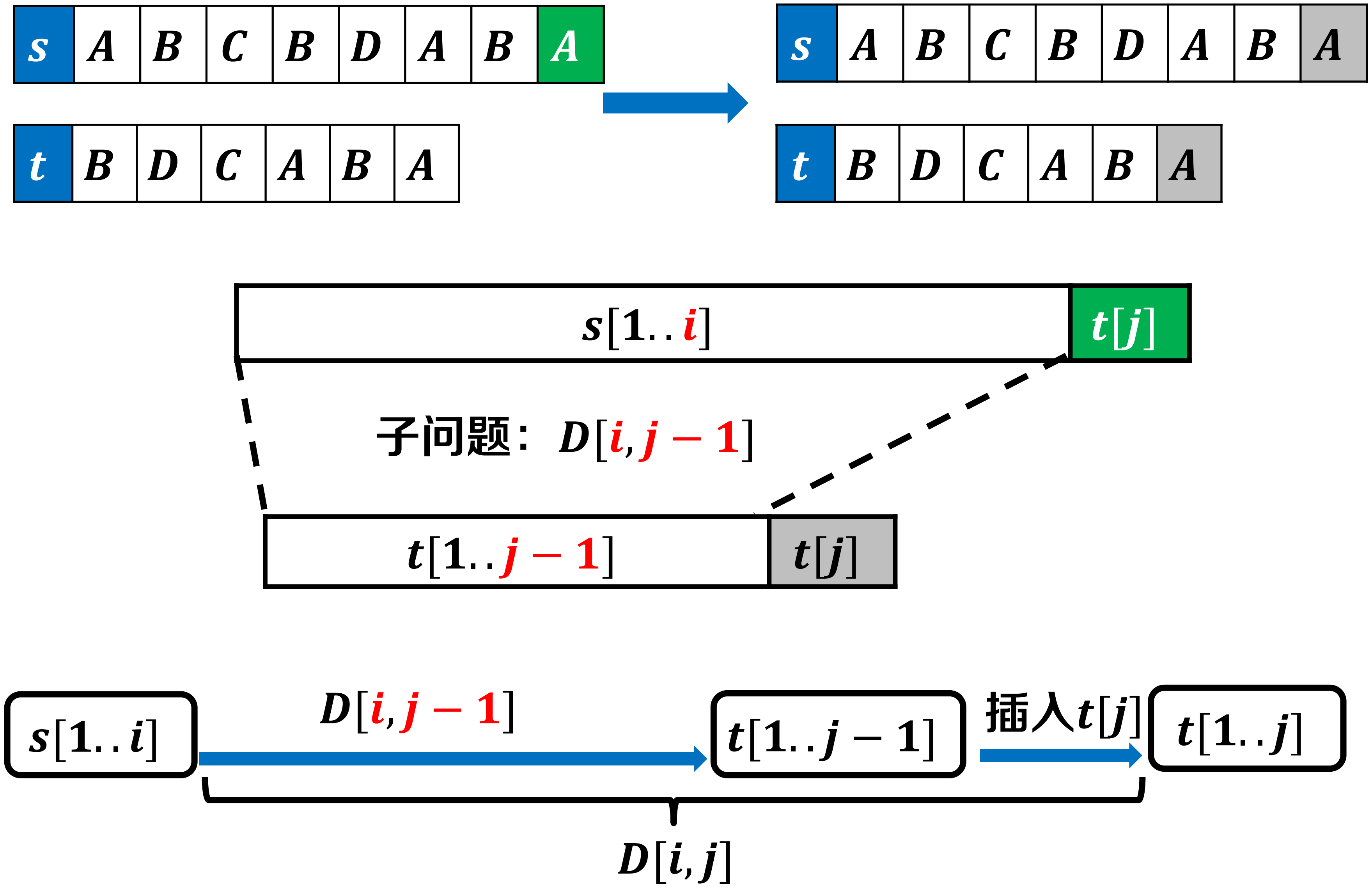
- $\boxed{D[i,j]} = \boxed{D[i-1,j]} + 1$

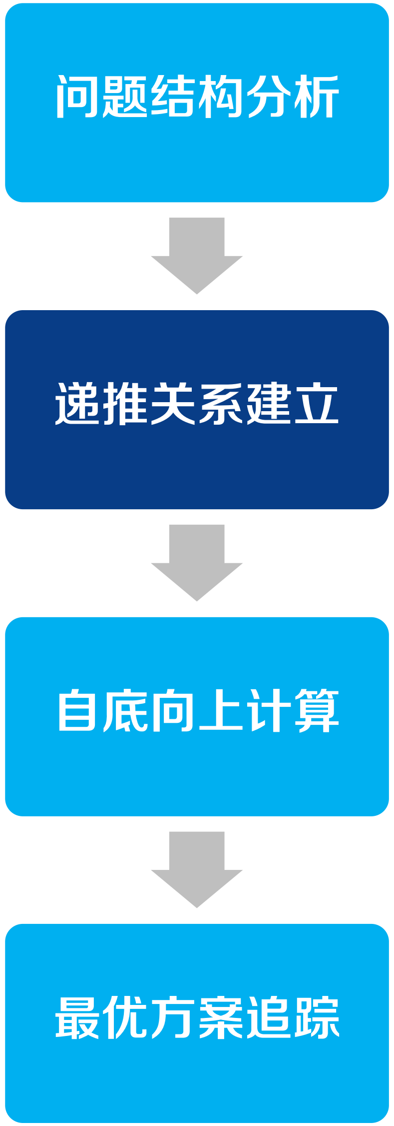
最优子结构

问题结构分析
↓
递推关系建立
↓
自底向上计算
↓
最优方案追踪

递推关系建立：分析最优（子）结构

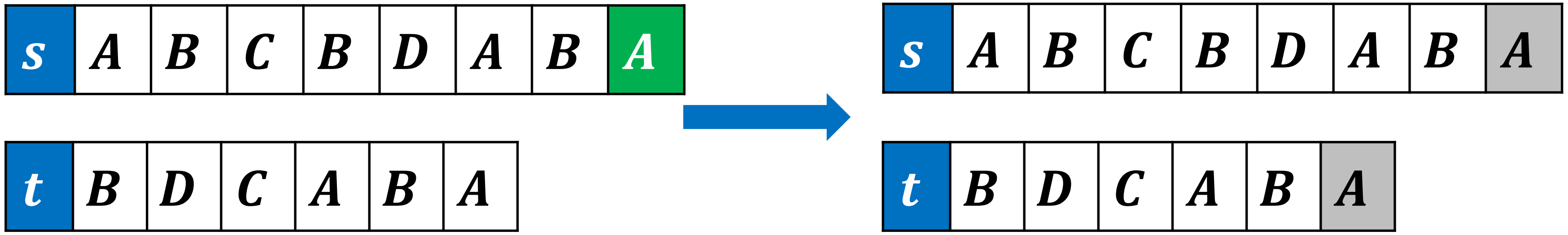
- 考察末尾元素：插入

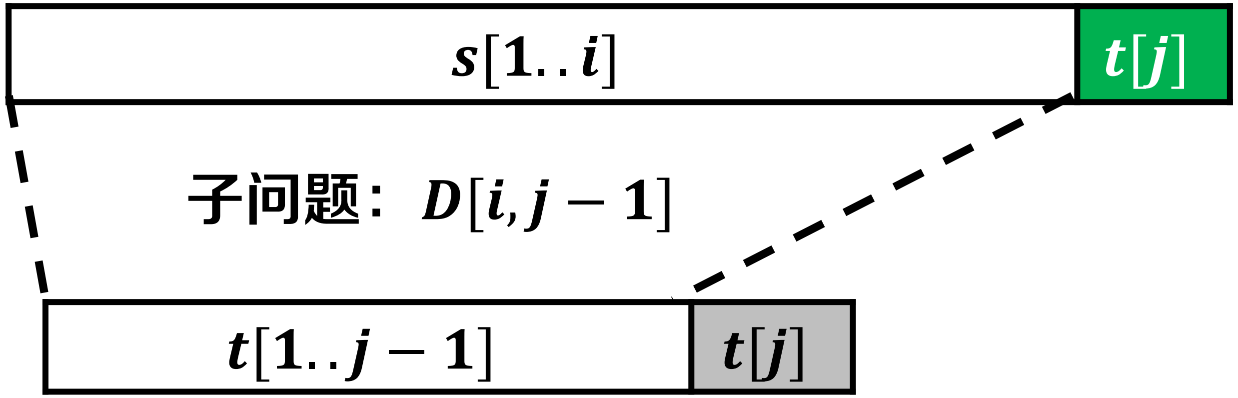




递推关系建立：分析最优（子）结构

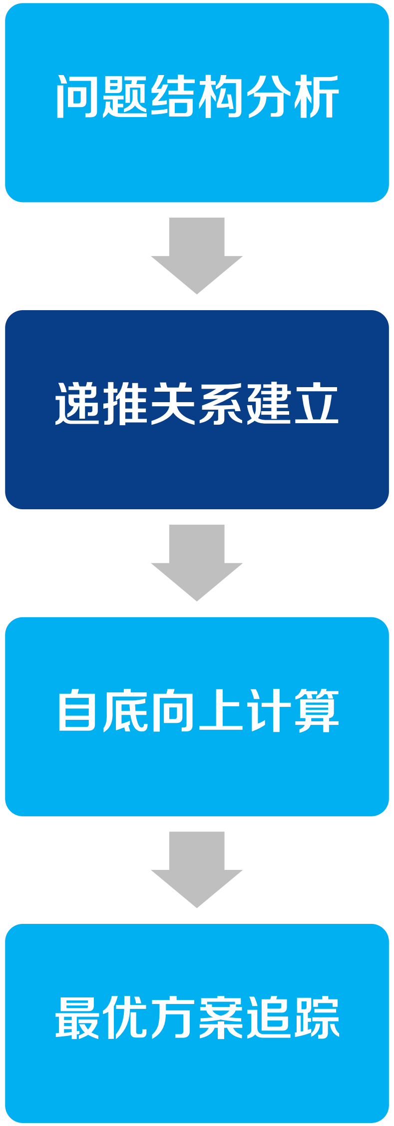
- 考察末尾元素：插入





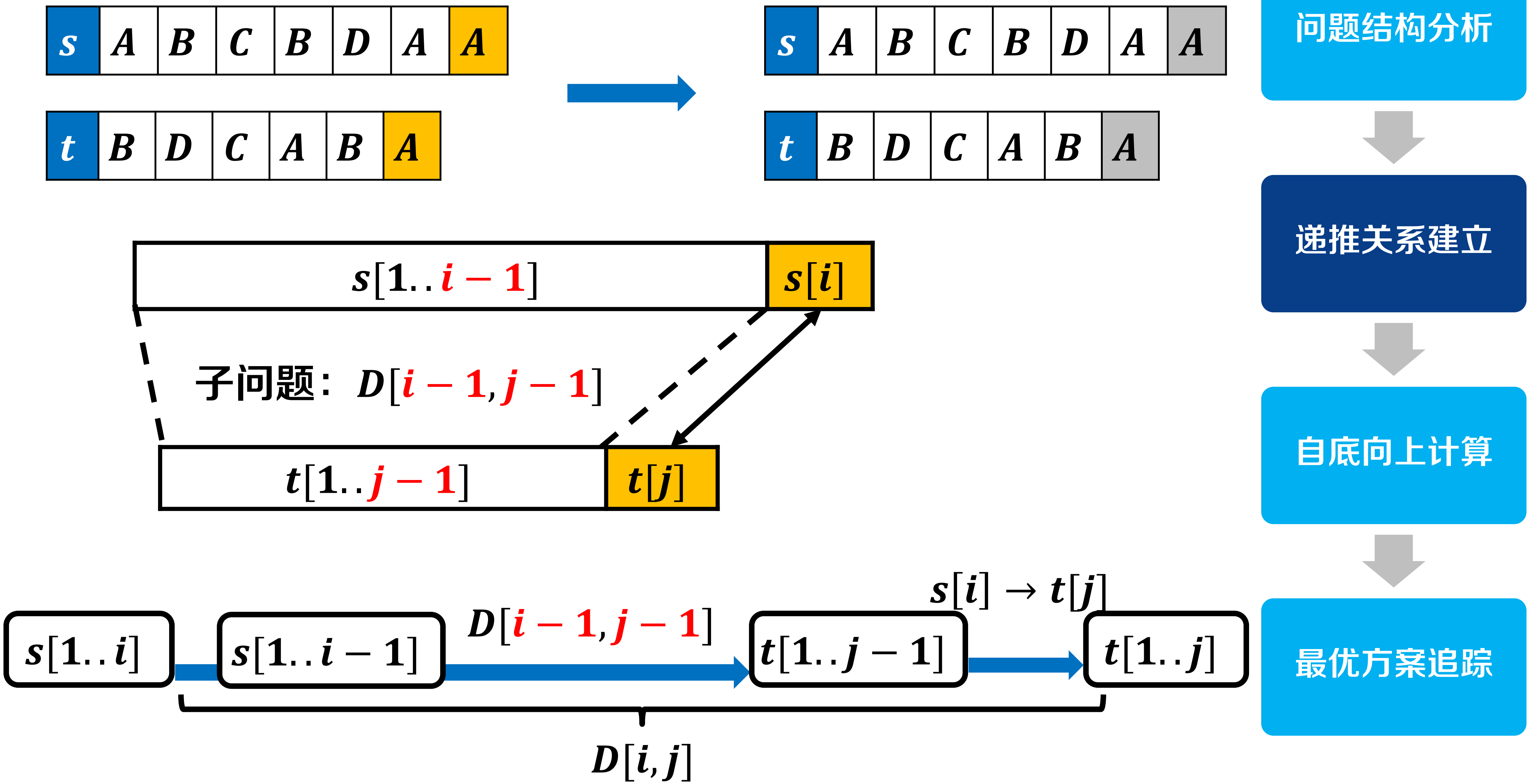
- $D[i,j] = D[i,j-1] + 1$

最优子结构



递推关系建立：分析最优（子）结构

- 考察末尾元素：替换



递推关系建立：分析最优（子）结构

- 考察末尾元素：替换

s	A	B	C	B	D	A	A

t	B	D	C	A	B	A

→

s	A	B	C	B	D	A	A

t	B	D	C	A	B	A

$s[1..i-1]$ $s[i]$

子问题：$D[i-1,j-1]$

如果$s[i]=t[j]$，不需要替换
否则需要替换

$t[1..j-1]$ $t[j]$

- $$\boxed{D[i,j]}=\boxed{D[i-1,j-1]}+\begin{cases}0, & if\ s[i]=t[j]\\ 1, & if\ s[i]\neq t[j]\end{cases}$$

最优子结构

问题结构分析

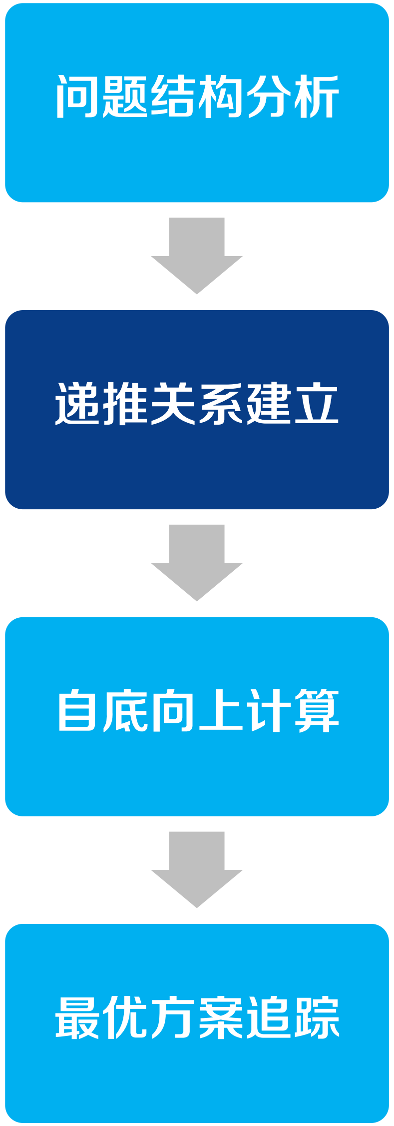
递推关系建立

自底向上计算

最优方案追踪

递推关系建立：构造递推公式

- 综合上面三种方式
  - $$D[i,j]=\min\begin{cases} D[i-1,j]+1 & \text{删除} \\ D[i,j-1]+1 & \text{插入} \\ D[i-1,j-1]+\begin{cases}0, & if\ s[i]=t[j] \\ 1, & if\ s[i]\neq t[j]\end{cases} & \text{替换}\end{cases}$$



递推关系建立：构造递推公式

- **最小编辑距离 vs. 最长公共子序列**

  - $$D[i,j] = \min \begin{cases} D[i-1,j]+1 \\ D[i,j-1]+1 \\ D[i-1,j-1]+\begin{cases} 0, & if\ s[i]=t[j] \\ 1, & if\ s[i]\neq t[j] \end{cases} \end{cases}$$

    删除

    插入

    替换

  - $$C[i,j] = \begin{cases} \max\{C[i-1,j], C[i,j-1]\}, & x_i \neq y_j \\ C[i-1,j-1]+1 & ,x_i = y_j \end{cases}$$

问题结构分析 → **递推关系建立** → 自底向上计算 → 最优方案追踪

自底向上计算：确定计算顺序

- 初始化
  - $D[i,0]=i$
    - 把长度为i的串变为空串至少需要i次操作（删除）
  - $D[0,j]=j$
    - 把空串变为长度为j的串至少需要j次操作（插入）

$i \backslash j$	0	1	2	…	$m-1$	m
0	0	1	2	…	$m-1$	m
1	1					
2	2					
…	…					
$n-1$	$n-1$					
n	n					

问题结构分析

递推关系建立

自底向上计算

最优方案追踪

自底向上计算：确定计算顺序

- 递推公式

  - $D[i,j] = \min \begin{cases} D[i-1,j]+1 \\ D[i,j-1]+1 \\ D[i-1,j-1] + \begin{cases} 0, & if\ s[i]=t[j] \\ 1, & if\ s[i]\neq t[j] \end{cases} \end{cases}$

删除

插入

替换

$i \backslash j$	0	1	2	$\dots$	$m-1$	m
0	0	1	2	$\dots$	$m-1$	m
1	1	$D[i-1,j-1]+\begin{cases}0\\1\end{cases}$				
2	2				$D[i-1,j]+1$	
$\dots$	$\dots$			$D[i,j]$		
$n-1$	$n-1$		$D[i,j-1]+1$			
n	n					

问题结构分析

递推关系建立

自底向上计算

最优方案追踪

自底向上计算：依次计算问题

- **递推公式**

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

$i \backslash j$	0	1	2	$\ldots$	$m-1$	m
0	0	1	2	$\ldots$	$m-1$	m
1	1					
2	2					
$\ldots$	$\ldots$					
$n-1$	$n-1$					
n	n					

问题结构分析

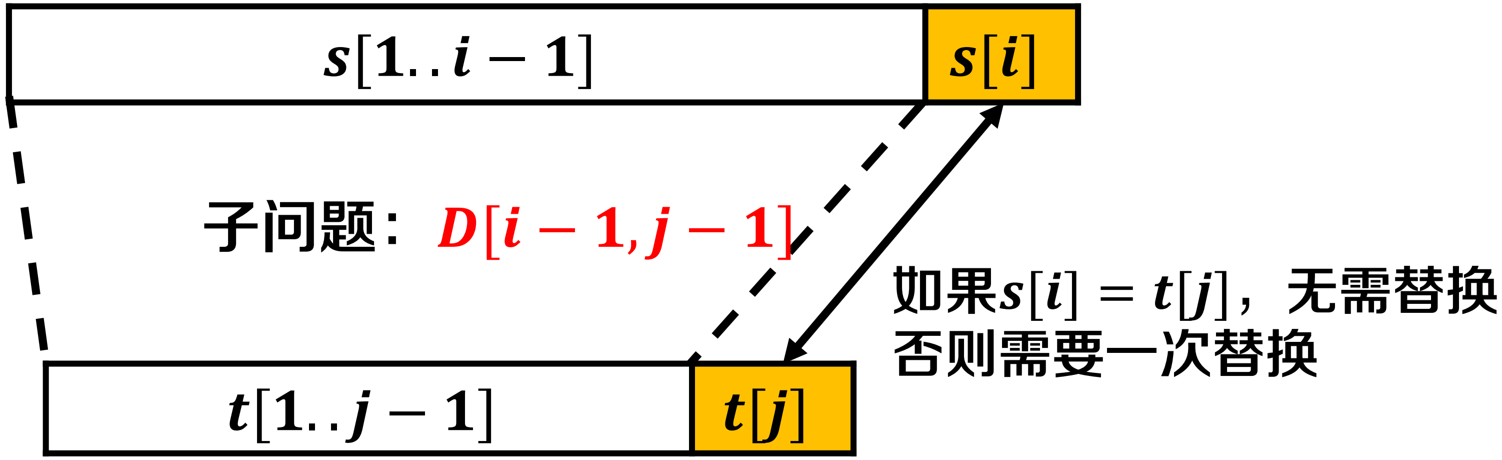
递推关系建立

自底向上计算

最优方案追踪

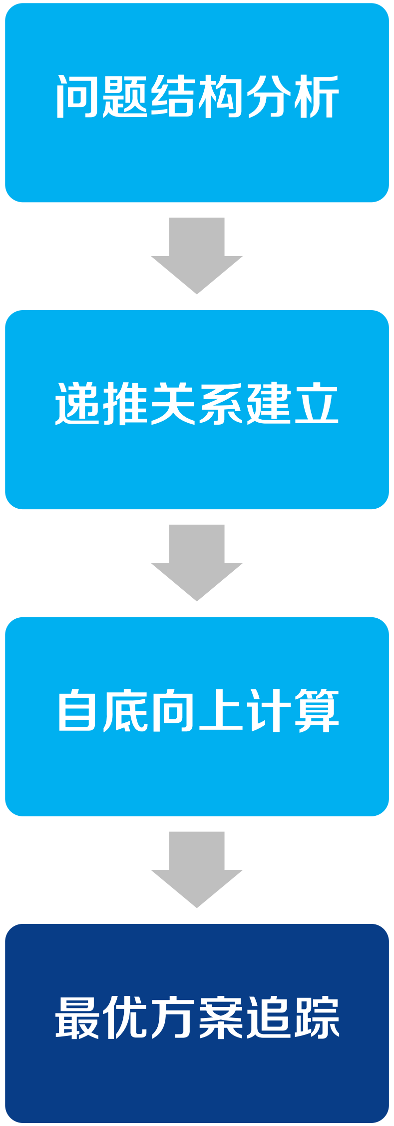
最优方案追踪：记录决策过程

- 追踪数组Rec，记录子问题来源



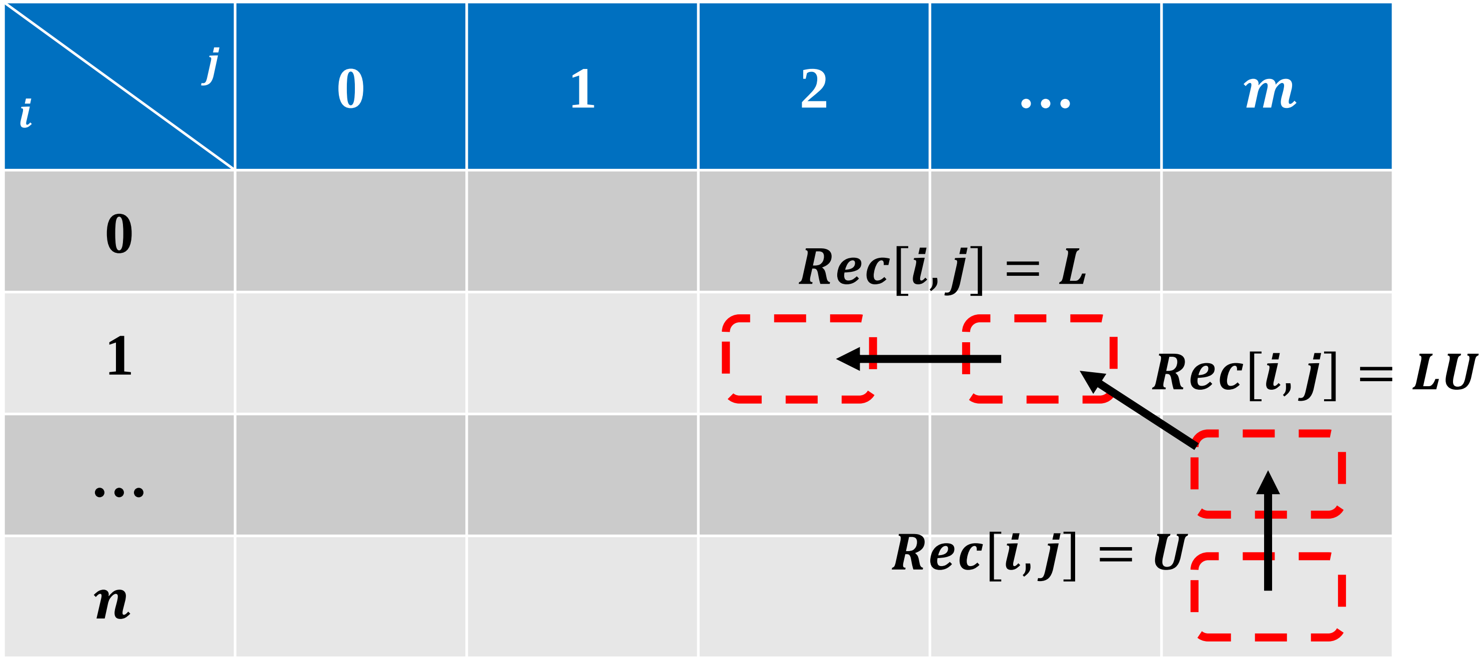
$D[i-1,j-1]$	$D[i-1,j]$
$D[i,j-1]$	$D[i,j]$

$Rec[i,j]$	子问题来源	操作
U	上侧，即$D[i-1,j]$	删除$s[i]$
L	左侧，即$D[i,j-1]$	插入$t[j]$
LU	左上，即$D[i-1,j-1]$	用$t[j]$替换$s[i]$/无操作

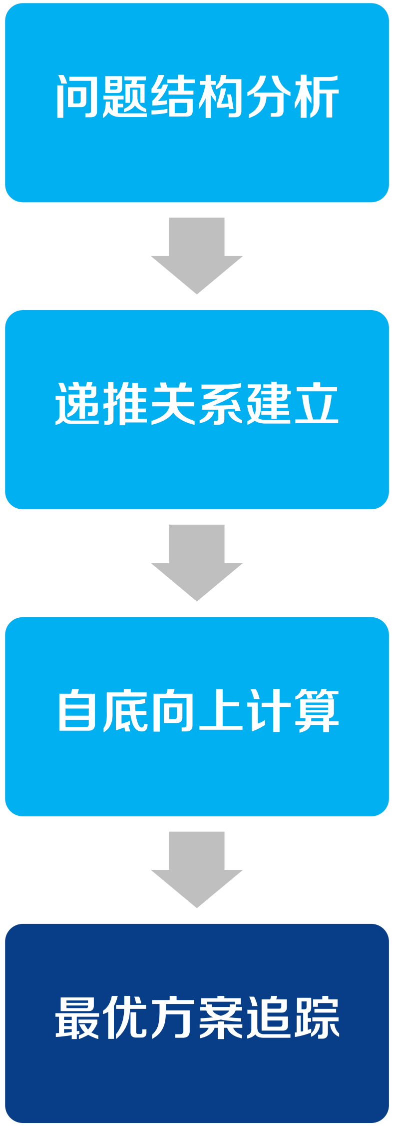


最优方案追踪：输出最优方案

- 根据数组Rec，输出最少编辑操作



$Rec[i,j]$	子问题来源	操作
U	上侧，即$D[i-1,j]$	删除$s[i]$
L	左侧，即$D[i,j-1]$	插入$t[j]$
LU	左上，即$D[i-1,j-1]$	用$t[j]$替换$s[i]$/无操作



算法实例

	1	2	3	4	5	6	7
s	A	B	C	B	D	A	B
t	B	D	C	A	B	A	

D

i \ *j*	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1						
2	2						
3	3						
4	4						
5	5						
6	6						
7	7						

初始化

Rec

i \ *j*	0	1	2	3	4	5	6
0		L	L	L	L	L	L
1	U						
2	U						
3	U						
4	U						
5	U						
6	U						
7	U						

算法实例

	1	2	3	4	5	6	7
s	A	B	C	B	D	A	B
t	B	D	C	A	B	A	

$s[i] \neq t[j]$

$$D[i,j] = \min \begin{cases} D[i-1,j]+1 \\ D[i,j-1]+1 \\ D[i-1,j-1] + \begin{cases} 0, if\ s[i] = t[j] \\ 1, if\ s[i] \neq t[j] \end{cases} \end{cases}$$

$D[i-1,j]$

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

$D[i,j-1]$

D

i \ j	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	1					
2	2						
3	3						
4	4						
5	5						
6	6						
7	7						

Rec

i \ j	0	1	2	3	4	5	6
0		L	L	L	L	L	L
1	U	LU					
2	U						
3	U						
4	U						
5	U						
6	U						
7	U						

算法实例

	1	2	3	4	5	6	7
s	A	B	C	B	D	A	B
t	B	D	C	A	D	A	

$$D[i,j] = \min \begin{cases} D[i-1,j]+1 \\ D[i,j-1]+1 \\ D[i-1,j-1] + \begin{cases} 0, if\ s[i]=t[j] \\ 1, if\ s[i]\neq t[j] \end{cases} \end{cases}$$

$s[i] \neq t[j]$

$D[i-1,j]$

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

$D[i,j-1]$

D

i \ j	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	1	2				
2	2						
3	3						
4	4						
5	5						
6	6						
7	7						

Rec

i \ j	0	1	2	3	4	5	6
0		L	L	L	L	L	L
1	U	LU	LU				
2	U						
3	U						
4	U						
5	U						
6	U						
7	U						

算法实例

	1	2	3	4	5	6	7
s	A	B	C	B	D	A	B
t	B	D	C	A	B	A	

$$D[i,j] = \min \begin{cases} D[i-1,j]+1 \\ D[i,j-1]+1 \\ D[i-1,j-1] + \begin{cases} 0, if\ s[i] = t[j] \\ 1, if\ s[i] \neq t[j] \end{cases} \end{cases}$$

D

$i \backslash j$	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	1	2	3	3	4	5
2	2	1	2	3	4	3	4
3	3	2	2	2	3	4	4
4	4	3	3	3	3	3	4
5	5	4	3	最优解			4
6	6	5	4	4	4	5	4
7	7	6	5	5	5	4	$\boxed{5}$

Rec

$i \backslash j$	0	1	2	3	4	5	6
0		L	L	L	L	L	L
1	U	LU	LU	LU	LU	L	LU
2	U	LU	LU	LU	LU	LU	L
3	U	U	LU	LU	L	L	LU
4	U	LU	LU	LU	LU	LU	L
5	U	U	LU	LU	LU	LU	LU
6	U	U	U	LU	LU	LU	LU
7	U	LU	U	LU	LU	LU	L

算法实例

	1	2	3	4	5	6	7
s	A	B	C	B	D	A	B
t	B	D	C	A	B	A	

$$D[i,j] = \min \begin{cases} D[i-1,j]+1 \\ D[i,j-1]+1 \\ D[i-1,j-1] + \begin{cases} 0, if\ s[i] = t[j] \\ 1, if\ s[i] \neq t[j] \end{cases} \end{cases}$$

操作：
删除A
无需操作
删除C
用D替换B
用C替换D
无需操作
无需操作
插入A

Rec

i \ j	0	1	2	3	4	5	6
0		L	L	L	L	L	L
1	U	LU	LU	LU	LU	L	LU
2	U	LU	LU	LU	LU	LU	L
3	U	U	LU	LU	L	L	LU
4	U	LU	LU	LU	LU	LU	L
5	U	U	LU	LU	LU	LU	LU
6	U	U	U	LU	LU	LU	LU
7	U	LU	U	LU	LU	LU	L

动态规划算法：伪代码

- **Minimum-Edit-Distance(s, t)**

输入: 字符串s和t
输出: s和t的最小编辑距离

$n \leftarrow \text{length}(s)$
$m \leftarrow \text{length}(t)$
新建$D[0..n, 0..m]$, $Rec[0..n, 0..m]$**两个二维数组**
//初始化
for $i \leftarrow 0$ *to* n **do**
  $D[i, 0] \leftarrow i$
  $Rec[i, 0] \leftarrow$ "U"
end
for $j \leftarrow 0$ *to* m **do**
  $D[0, j] \leftarrow j$
  $Rec[0, j] \leftarrow$ "L"
end

动态规划算法：伪代码

- **Minimum-Edit-Distance(s, t)**

```
//动态规划
for i ← 1 to n do
    for j ← 1 to m do
        c ← 0
        if s_i ≠ t_j then
            c ← 1
        end
        replace ← D[i − 1, j − 1] + c
        delete ← D[i − 1, j] + 1
        insert ← D[i, j − 1] + 1
        if replace = min{delete, insert, replace} then
            D[i, j] ← D[i − 1, j − 1] + c
            Rec[i, j] ← "LU"
        end
        else if insert = min{delete, insert, replace} then
            D[i, j] ← D[i, j − 1] + 1
            Rec[i, j] ← "L"
        end
        else
            D[i, j] ← D[i − 1, j] + 1
            Rec[i, j] ← "U"
        end
    end
end
```

最优方案追踪：伪代码

- **Print-MED(Rec, s, t, i, j)**

```
输入: 矩阵Rec, 字符串s, t, 位置索引i和j
输出: 操作序列
if i = 0 and j = 0 then
  | return NULL
end
if Rec[i, j] = "LU" then
  | Print-MED(Rec, s, t, i − 1, j − 1)
  | if s_i = t_j then
  |   | print "无需操作"
  | end
  | else
  |   | print "用t_j替换s_i"
  | end
end
else if Rec[i, j] = "U" then
  | Print-MED(Rec, s, t, i − 1, j)
  | print "删除s_i"
end
else
  | Print-MED(Rec, s, t, i, j − 1)
  | print "插入t_j"
end
```

时间复杂度分析

```
//动态规划
for i ← 1 to n do
    for j ← 1 to m do
        c ← 0
        if s_i ≠ t_j then
            c ← 1
        end
        replace ← D[i-1, j-1] + c
        delete ← D[i-1, j] + 1
        insert ← D[i, j-1] + 1
        if replace = min{delete, insert, replace} then
            D[i, j] ← D[i-1, j-1] + c
            Rec[i, j] ← "LU"
        end
        else if insert = min{delete, insert, replace} then
            D[i, j] ← D[i, j-1] + 1
            Rec[i, j] ← "L"
        end
        else
            D[i, j] ← D[i-1, j] + 1
            Rec[i, j] ← "U"
        end
    end
end
```

$O(m)$ $O(mn)$

时间复杂度：$O(mn)$

小结

最长公共子序列

如果$s_i \neq t_j$

s	A	B	C	B	D	A	B

t	B	D	C	A	B	A

如果$s_i = t_j$

s	A	B	C	B	D	A	B

t	B	D	C	A	B

$$C[i,j] = \begin{cases} \max\{C[i-1,j], C[i,j-1]\}, & x_i \neq y_j \\ C[i-1,j-1]+1 & , x_i = y_j \end{cases}$$

最小编辑距离

删除

s	A	B	C	B	D	A	B

t	B	D	C	A	B	A

插入

s	A	B	C	B	D	A	B	?

t	B	D	C	A	B	A

替换

s	A	B	C	B	D	A	B	?

t	B	D	C	A	B	A

$$D[i,j] = \min \begin{cases} D[i-1,j]+1 \\ D[i,j-1]+1 \\ D[i-1,j-1] + \begin{cases} 0, if\ s[i] = t[j] \\ 1, if\ s[i] \neq t[j] \end{cases} \end{cases}$$

谢谢

BDA

Beihang University