

本节内容

KMP算法

进一步优化

KMP算法

根据模式串T，求出 next 数组

利用next数组进行匹配
(主串指针不回溯)

T = 'abaabc'

next数组:

next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
	0	1	1	2	2	3

if (S[i] != T[j]) j=next[j];

if (j==0) { i++; j++ }

KMP算法, 最坏时间复杂度 $O(m+n)$

其中, 求 next 数组时间复杂度 $O(m)$
模式匹配过程最坏时间复杂度 $O(n)$

```
int Index_KMP(SString S, SString T, int next[]){
    int i=1, j=1;
    while(i<=S.length&& j<=T.length){
        if(j==0 || S.ch[i]==T.ch[j]){
            ++i;
            ++j;           //继续比较后继字符
        }
        else
            j=next[j];     //模式串向右移动
    }
    if(j>T.length)
        return i-T.length; //匹配成功
    else
        return 0;
}
```

手算求next数组的方法

根据模式串T，求出 next 数组

T = 'abaabc'

next数组:

next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
	0	1	1	2	2	3

if (S[i] != T[j]) j=next[j];

if (j==0) { i++; j++ }

next[1]都无脑写 0

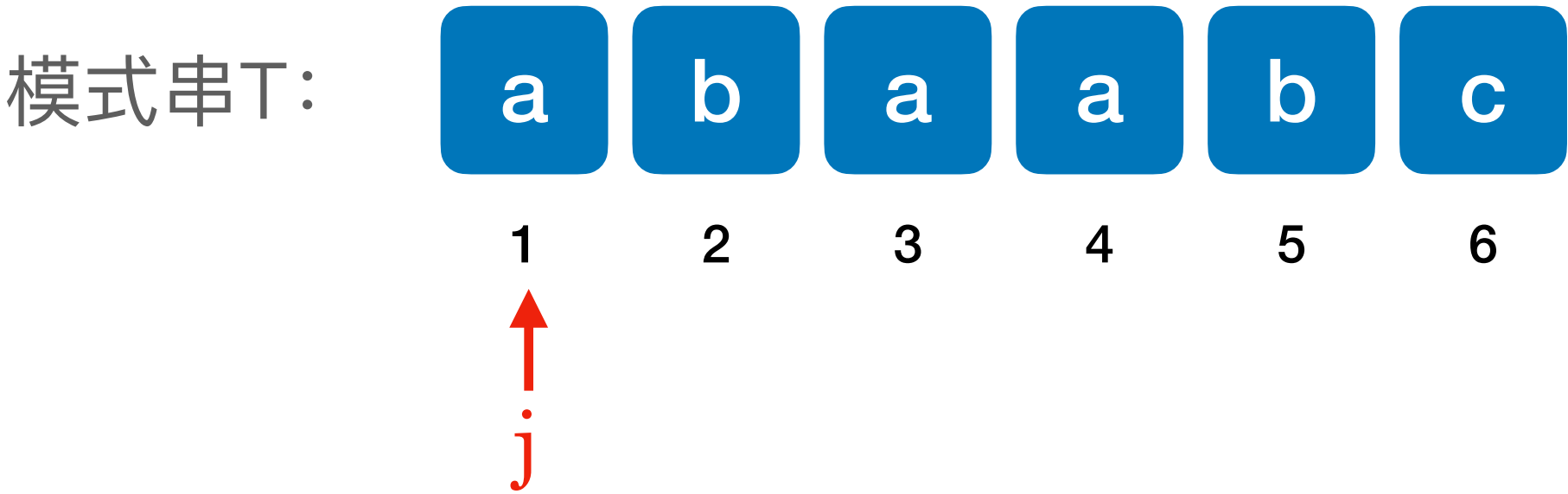
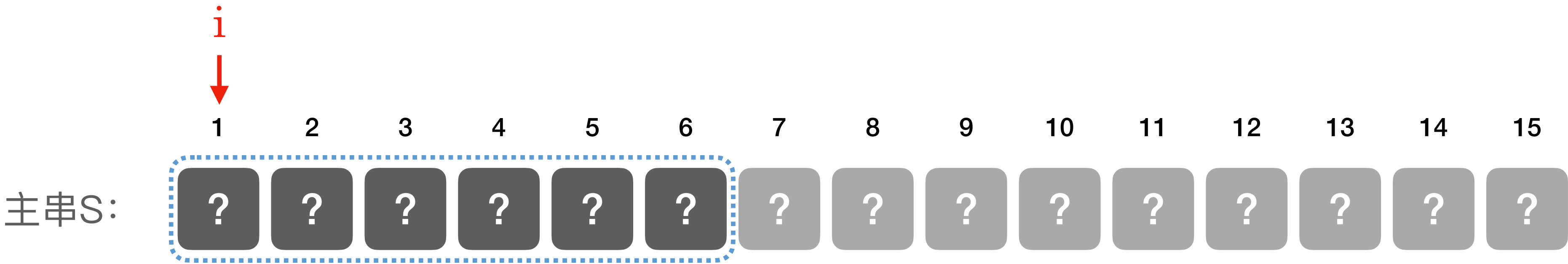
next[2]都无脑写 1

其他 next: 在不匹配的位置前，划一根美丽的分界线
模式串一步一步往后退，直到分界线之前“能对上”，或模式串完全跨过分界线为止。此时 j 指向哪儿，next数组值就是多少

KMP算法，最坏时间复杂度 $O(m+n)$

其中，求 next 数组时间复杂度 $O(m)$
模式匹配过程最坏时间复杂度 $O(n)$

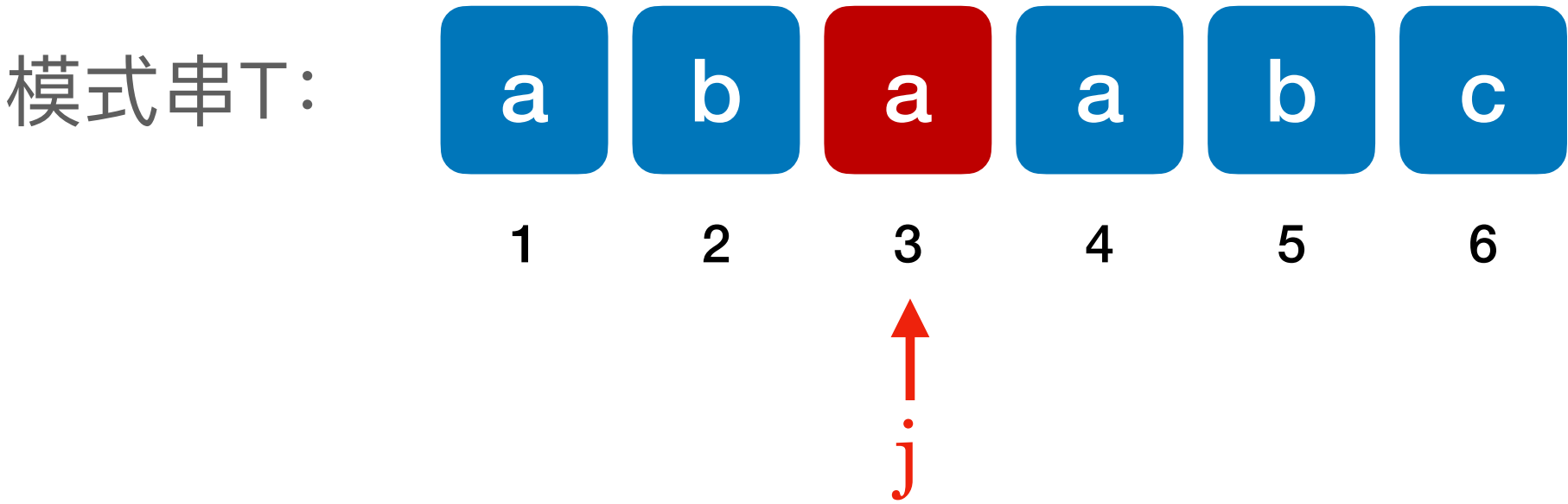
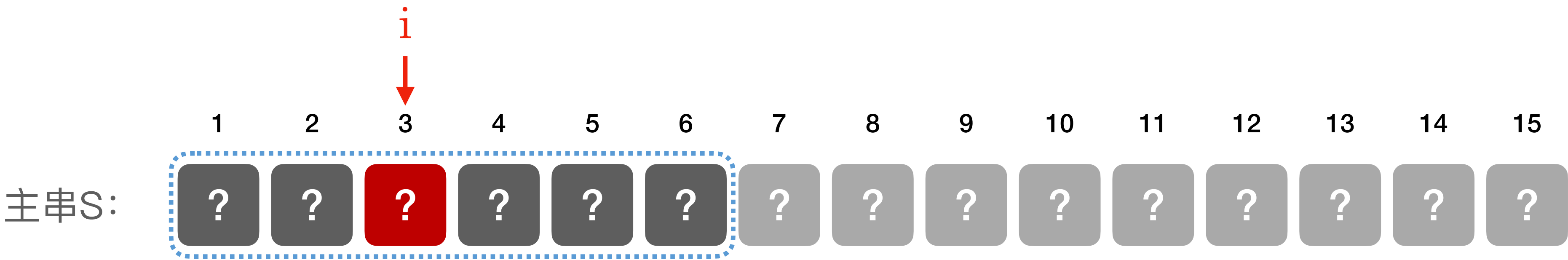
next数组的优化思路



next数组:

next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
	0	1	1	2	2	3

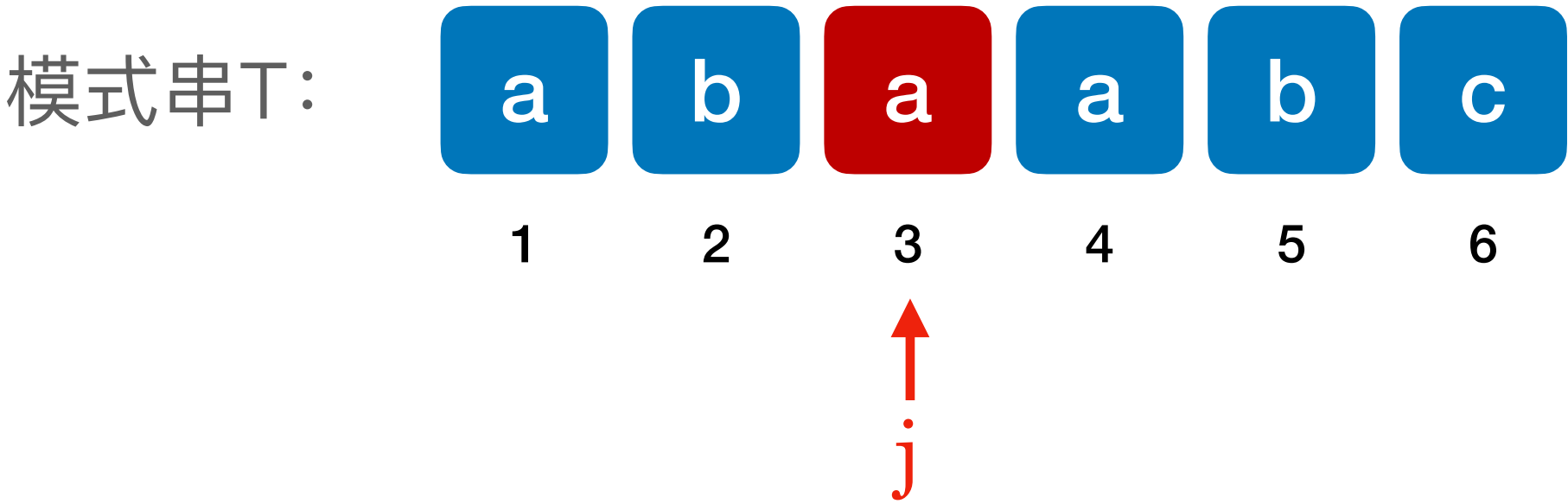
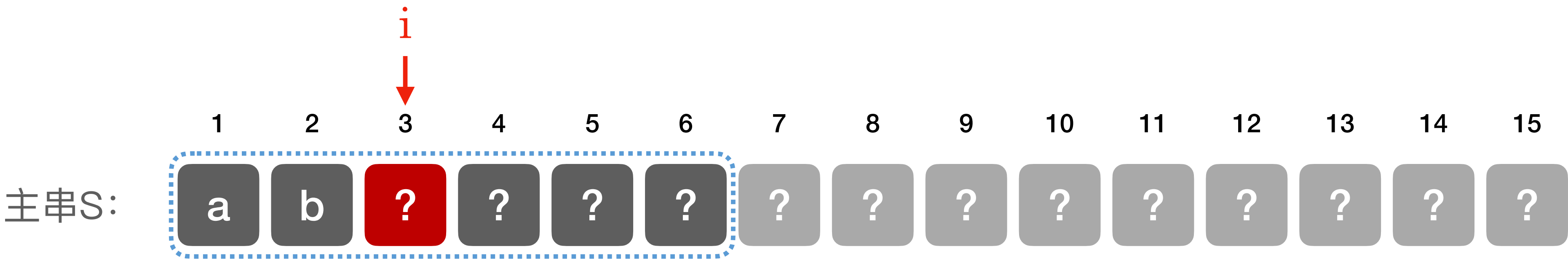
next数组的优化思路



next数组:

next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
	0	1	1	2	2	3

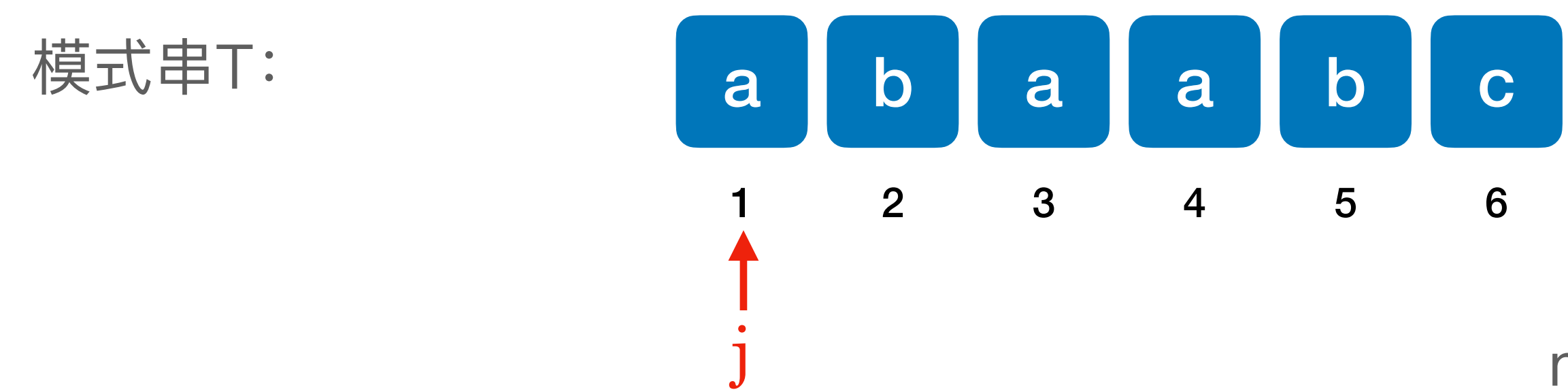
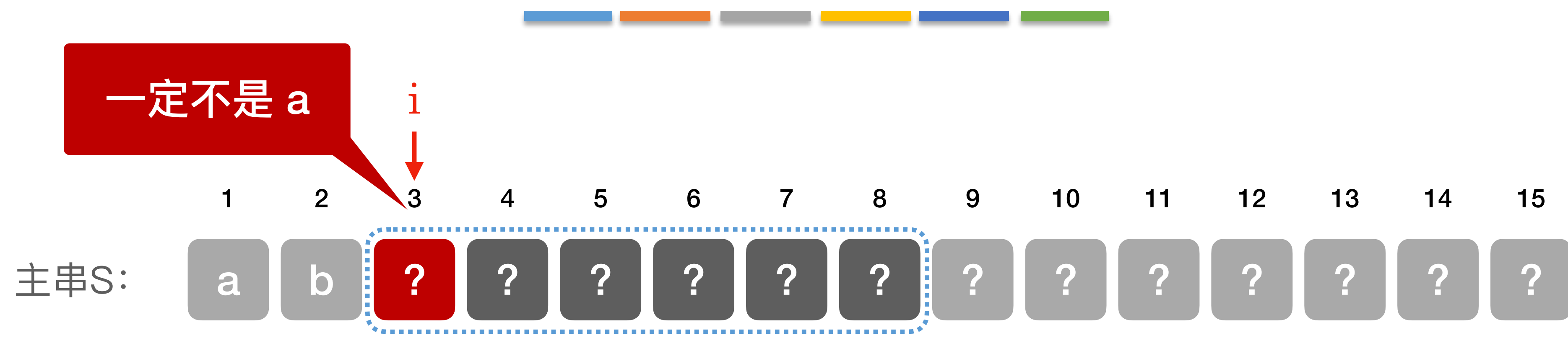
next数组的优化思路



next数组:

next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
	0	1	1	2	2	3

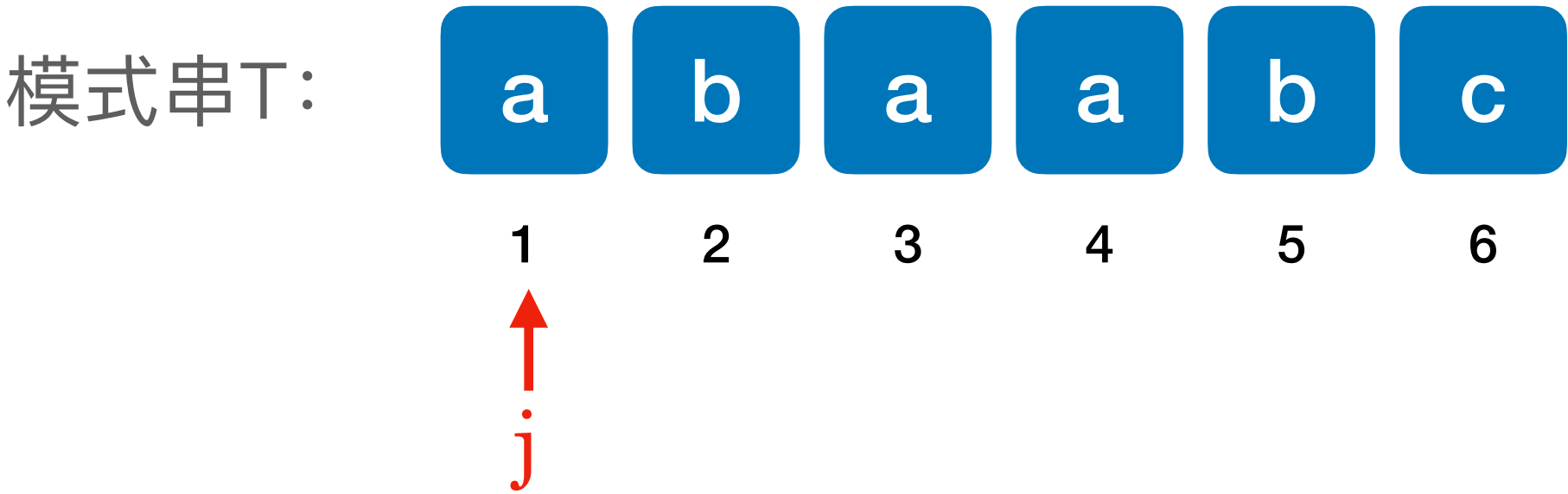
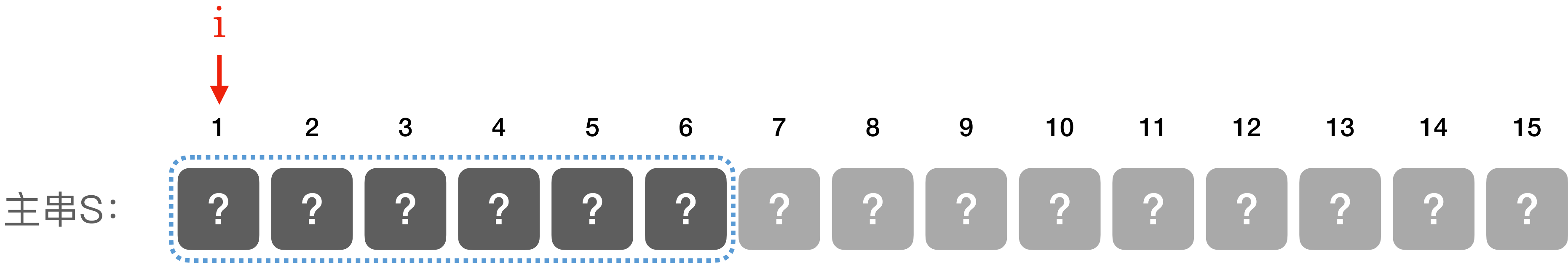
next数组的优化思路



next数组:

next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
	0	1	1	2	2	3

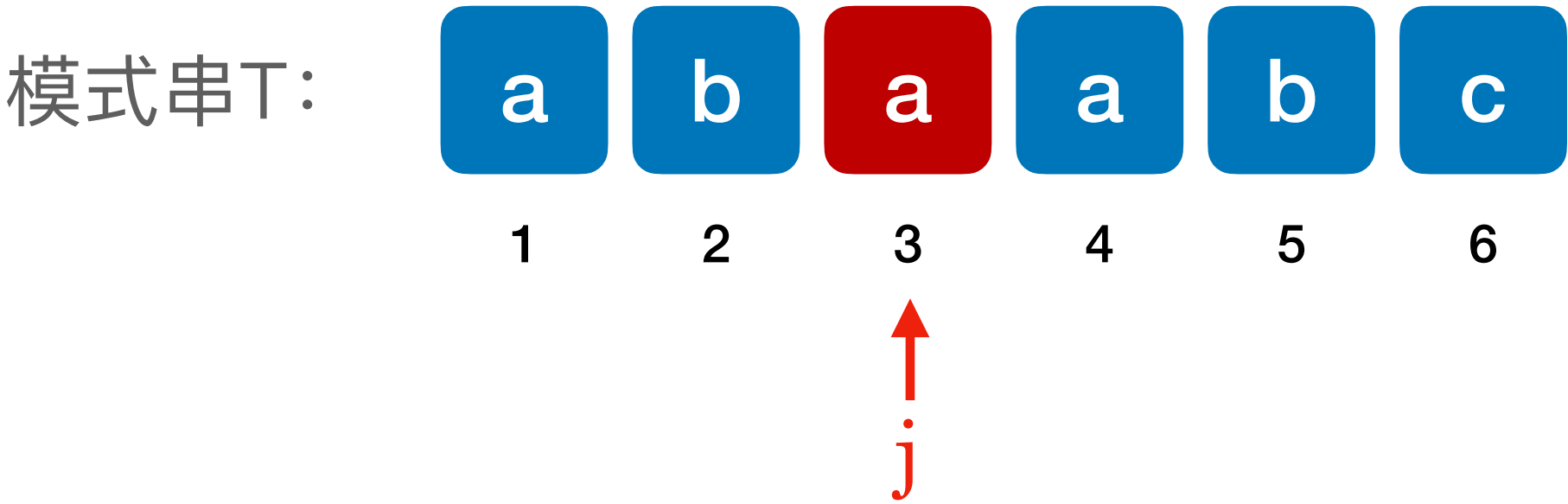
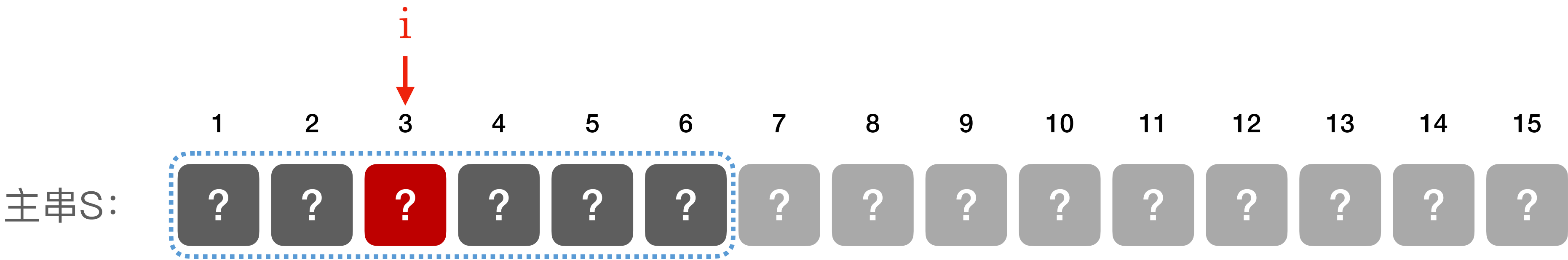
next数组的优化思路



next数组:

next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
	0	1	0	2	2	3

next数组的优化思路



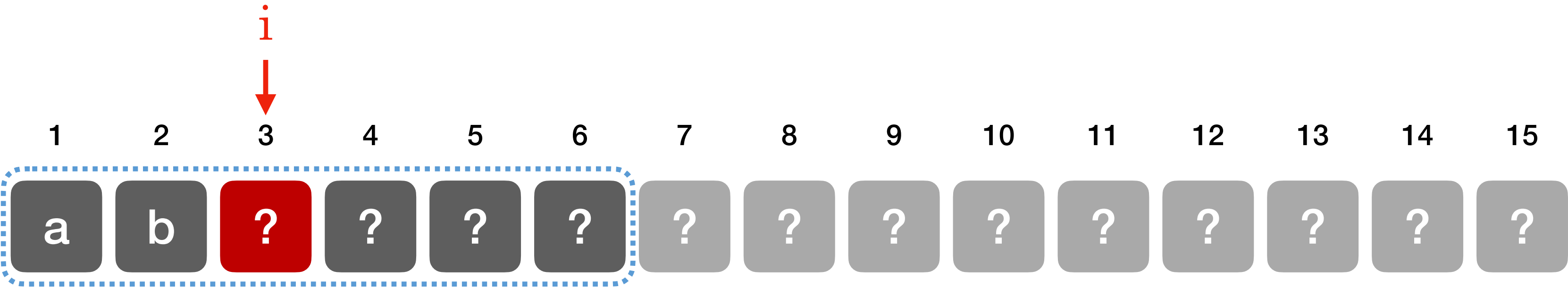
next数组:

next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
	0	1	0	2	2	3

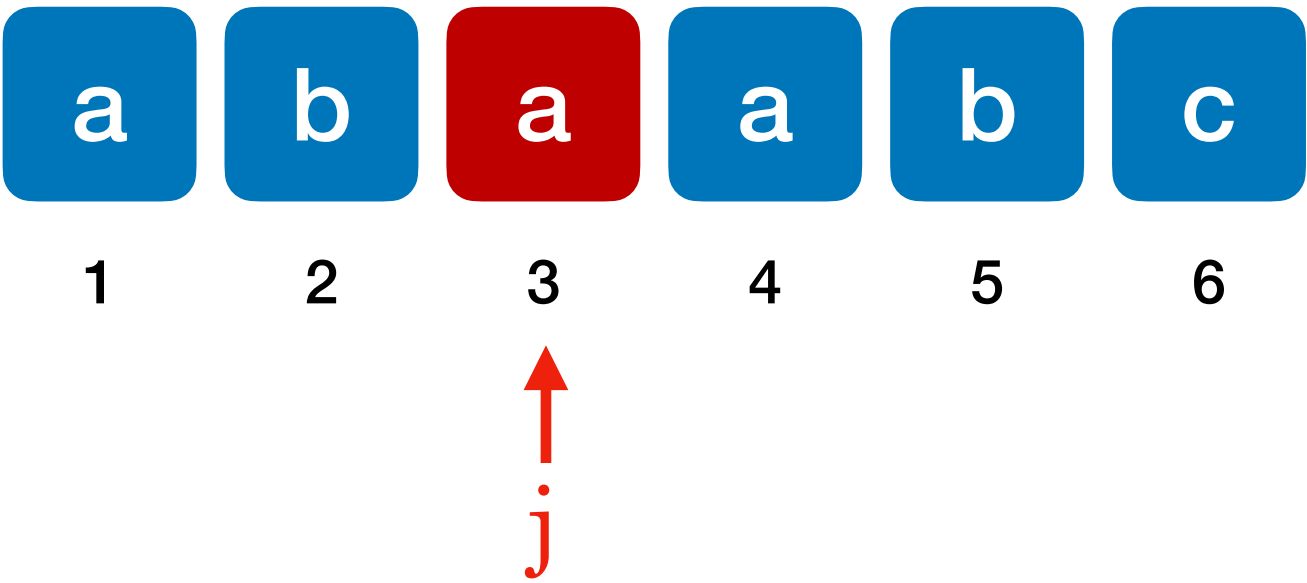
next数组的优化思路



主串S:



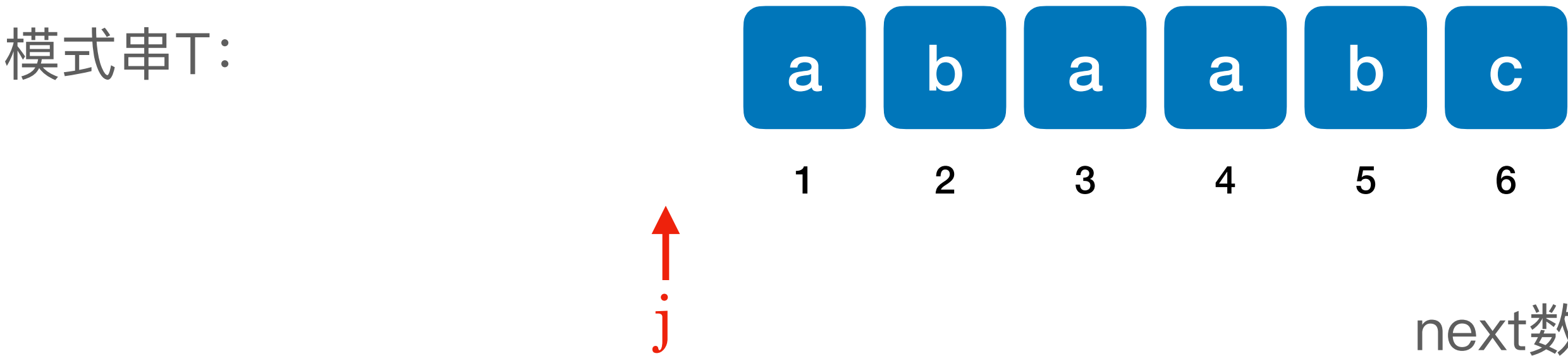
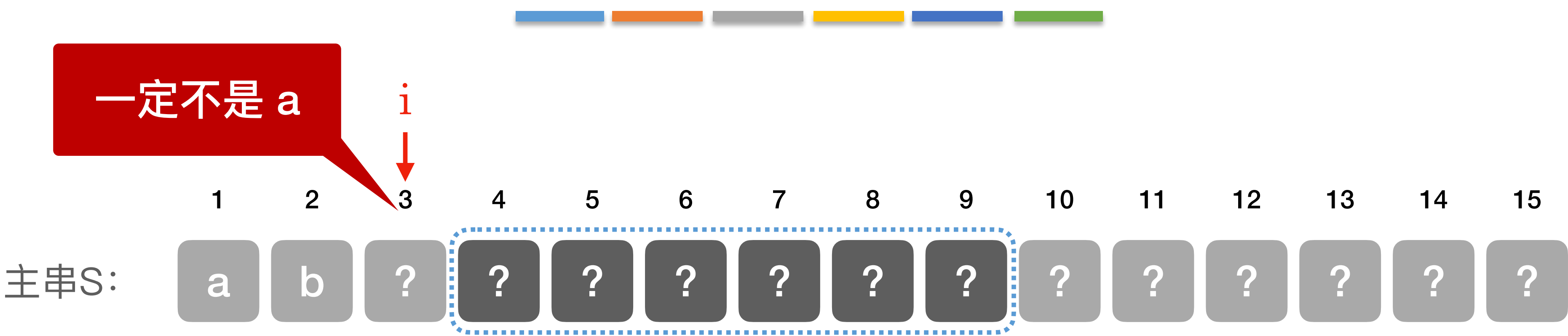
模式串T:



next数组:

next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
	0	1	0	2	2	3

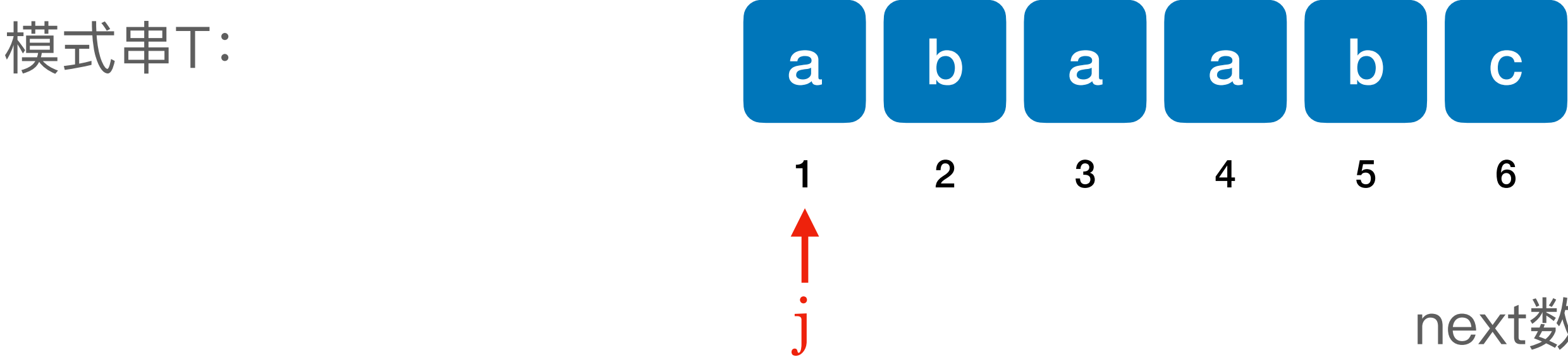
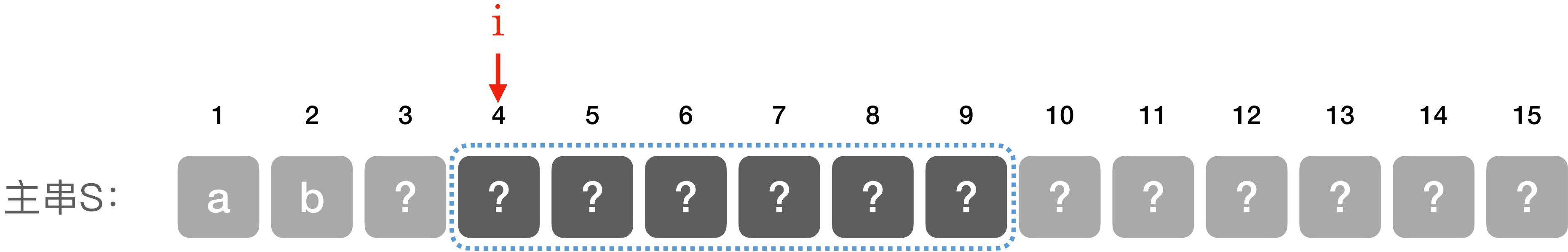
next数组的优化思路



next数组:

next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
	0	1	0	2	2	3

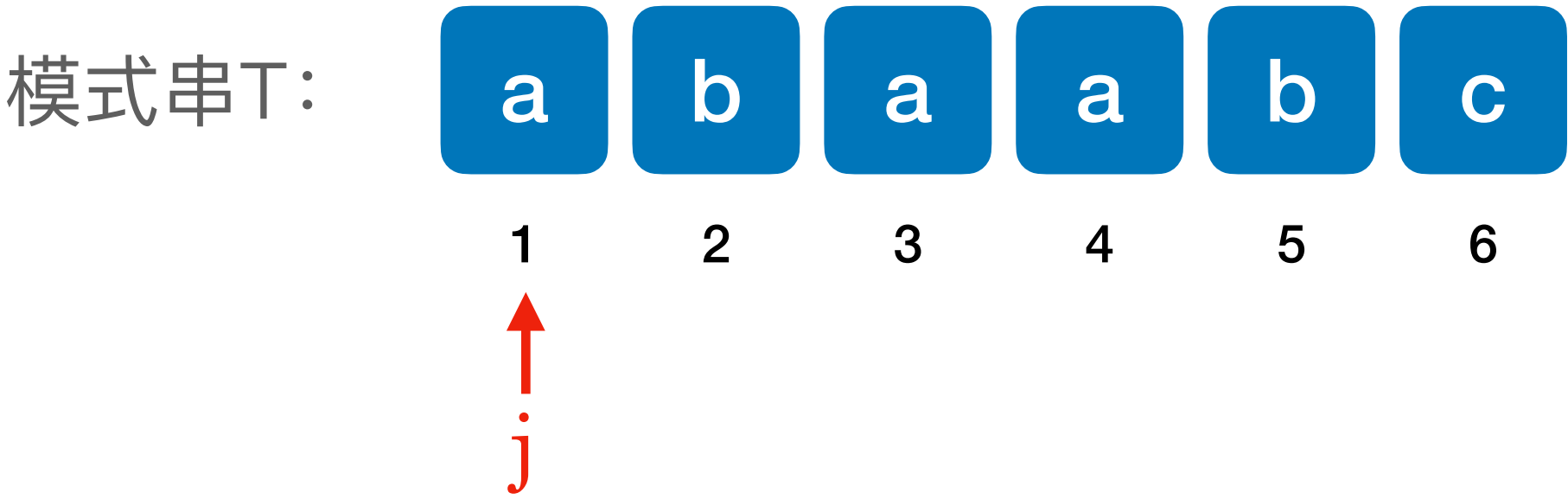
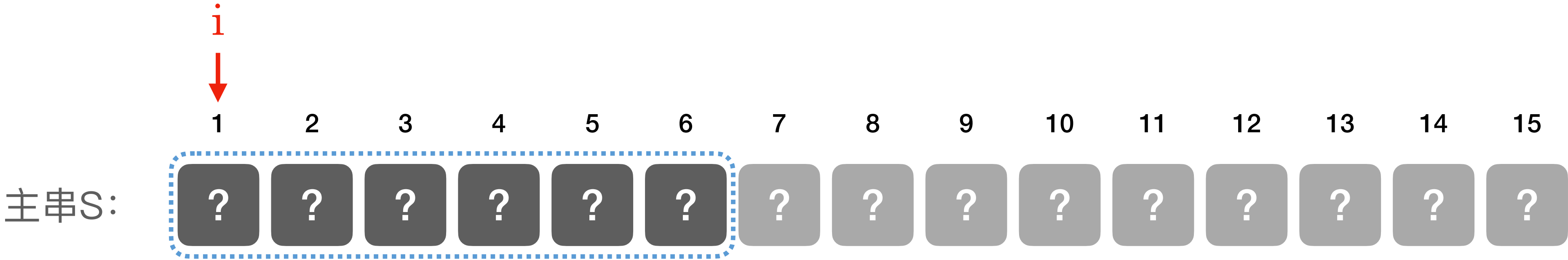
next数组的优化思路



next数组:

next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
	0	1	0	2	2	3

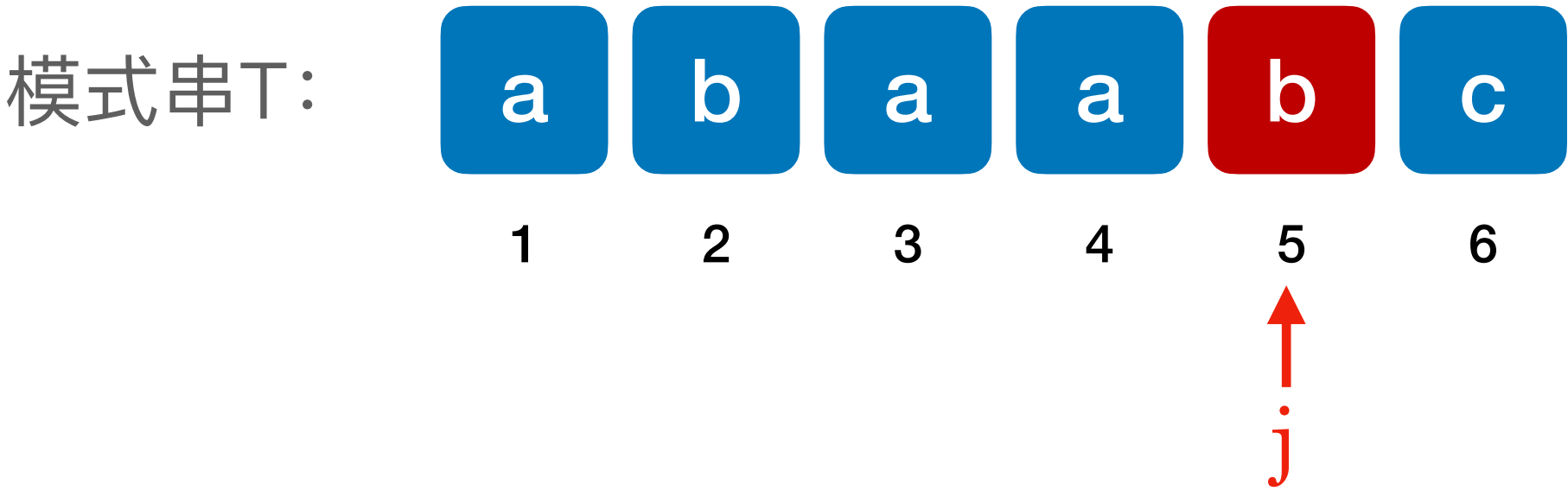
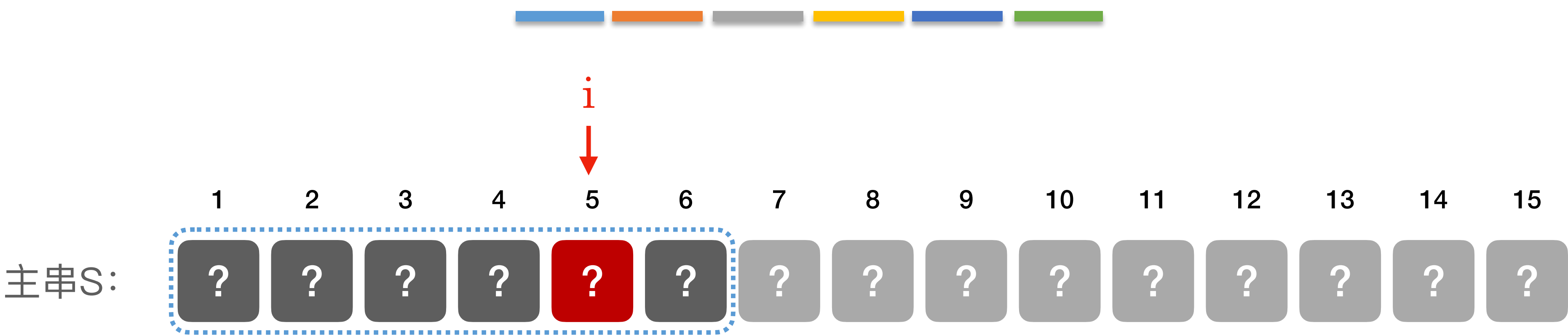
next数组的优化思路



next数组:

next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
	0	1	0	2	2	3

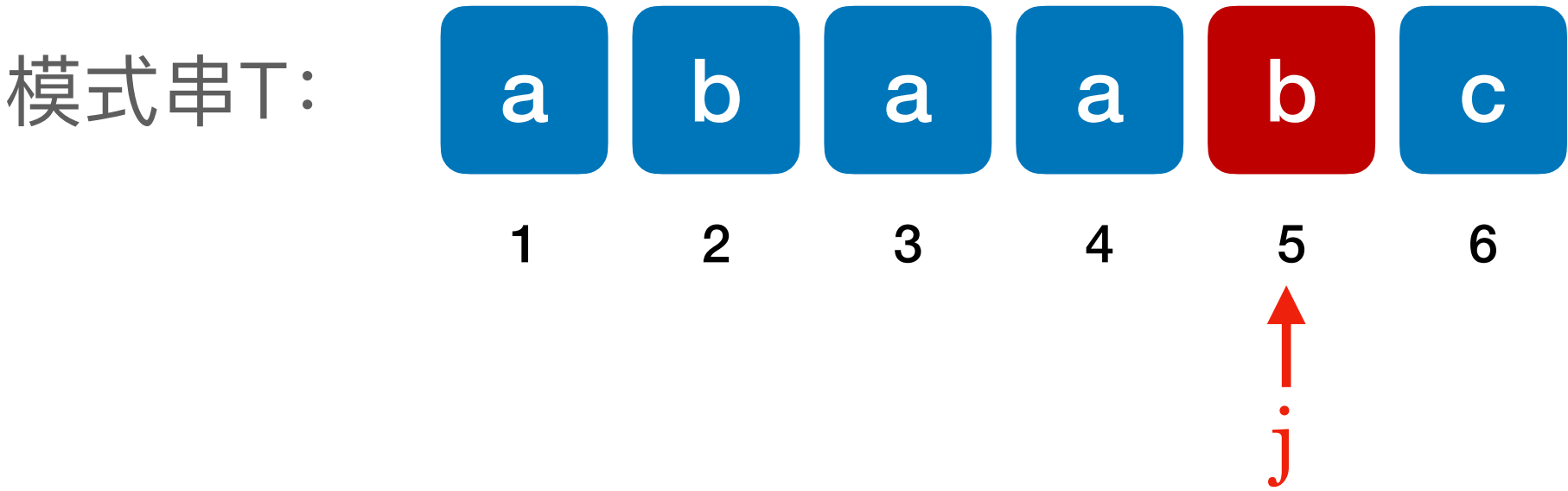
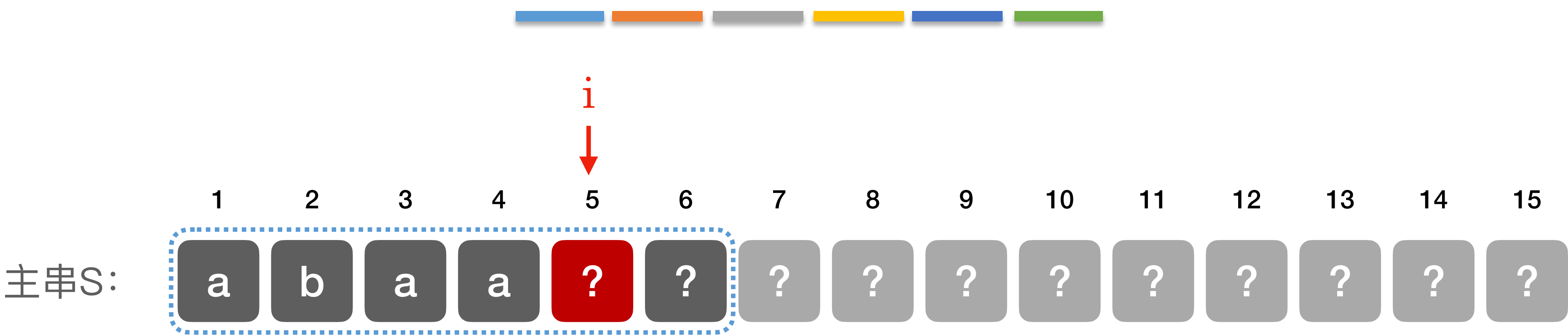
next数组的优化思路



next数组:

next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
	0	1	0	2	2	3

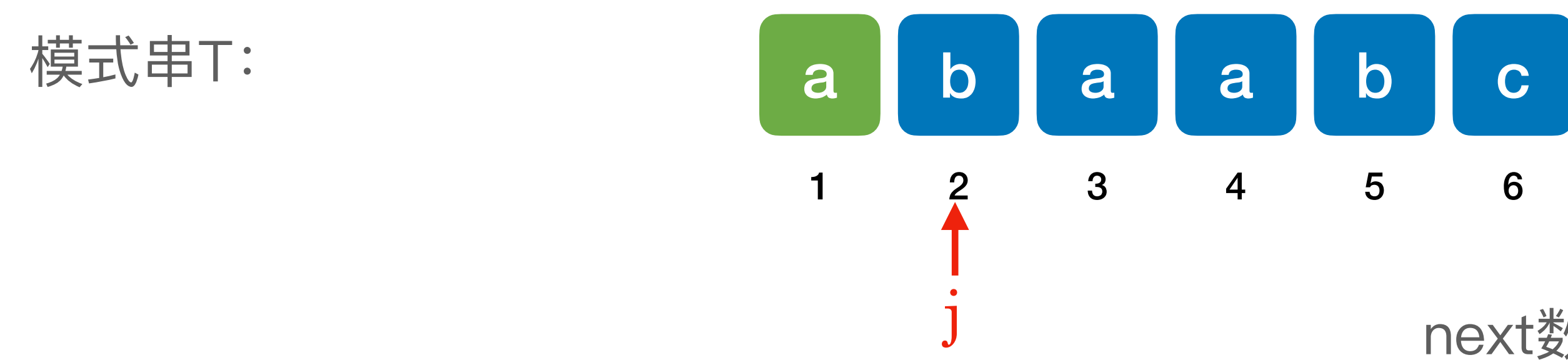
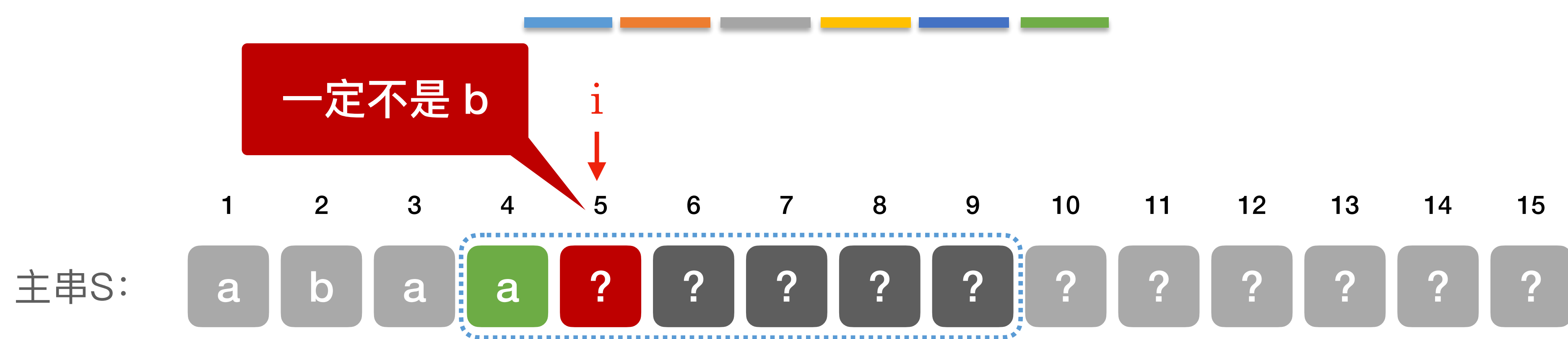
next数组的优化思路



next数组:

next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
	0	1	0	2	2	3

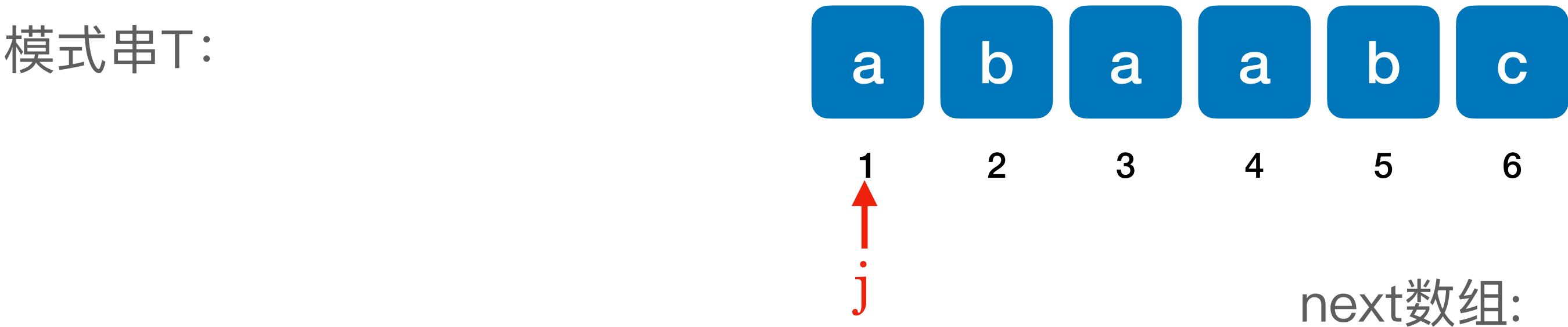
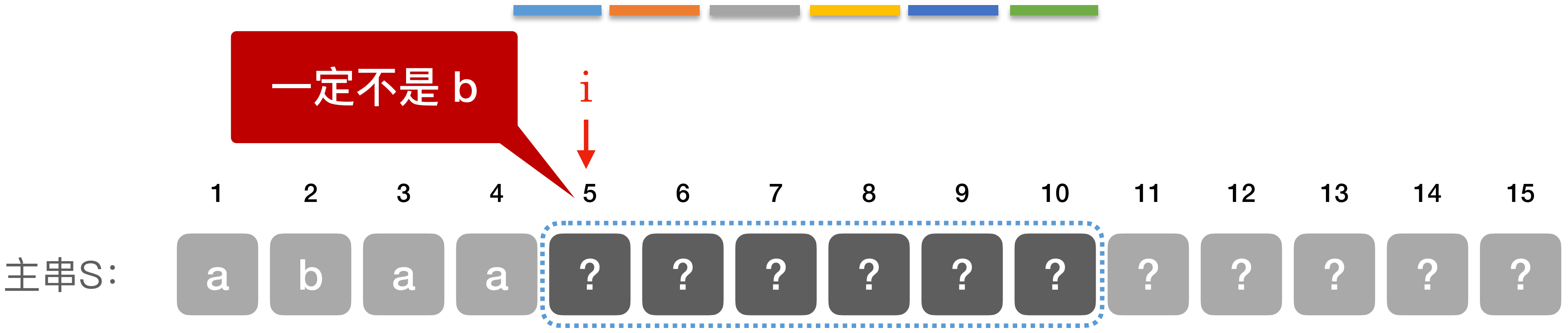
next数组的优化思路



next数组:

next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
	0	1	0	2	2	3

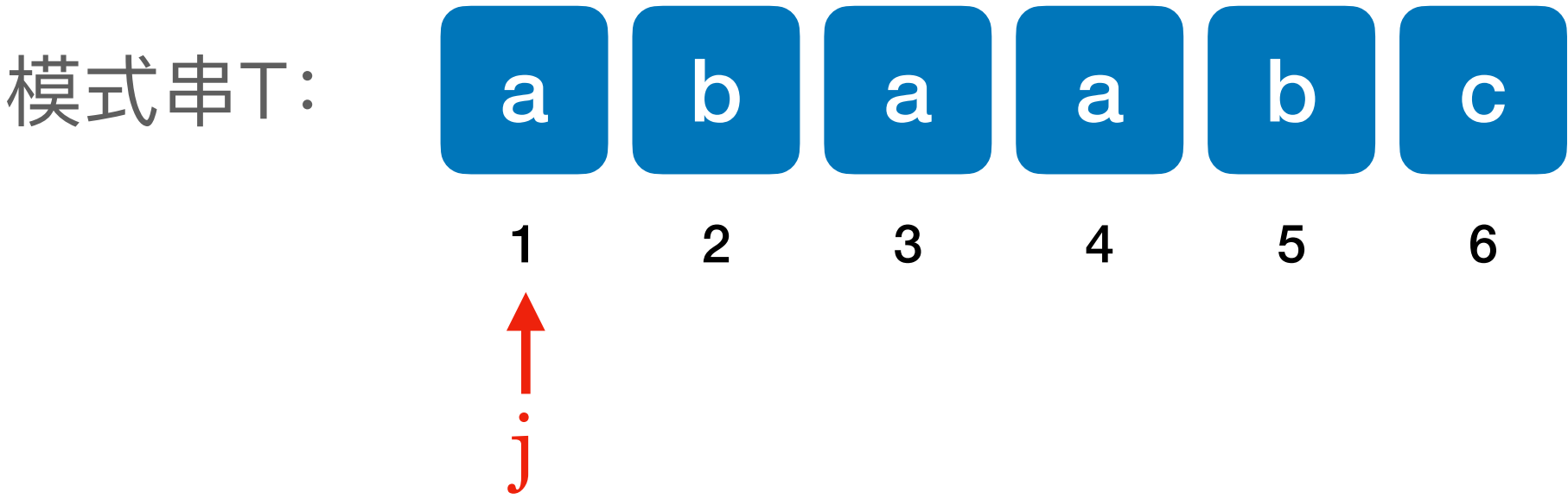
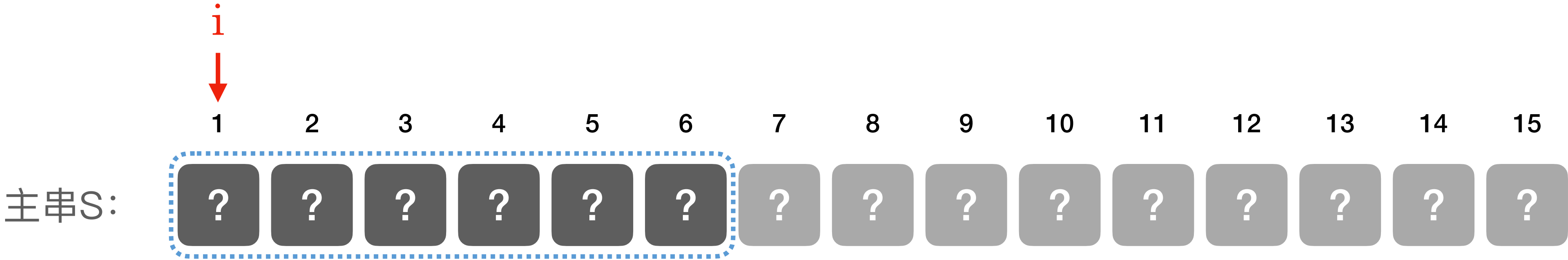
next数组的优化思路



next数组:

next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
	0	1	0	2	2	3

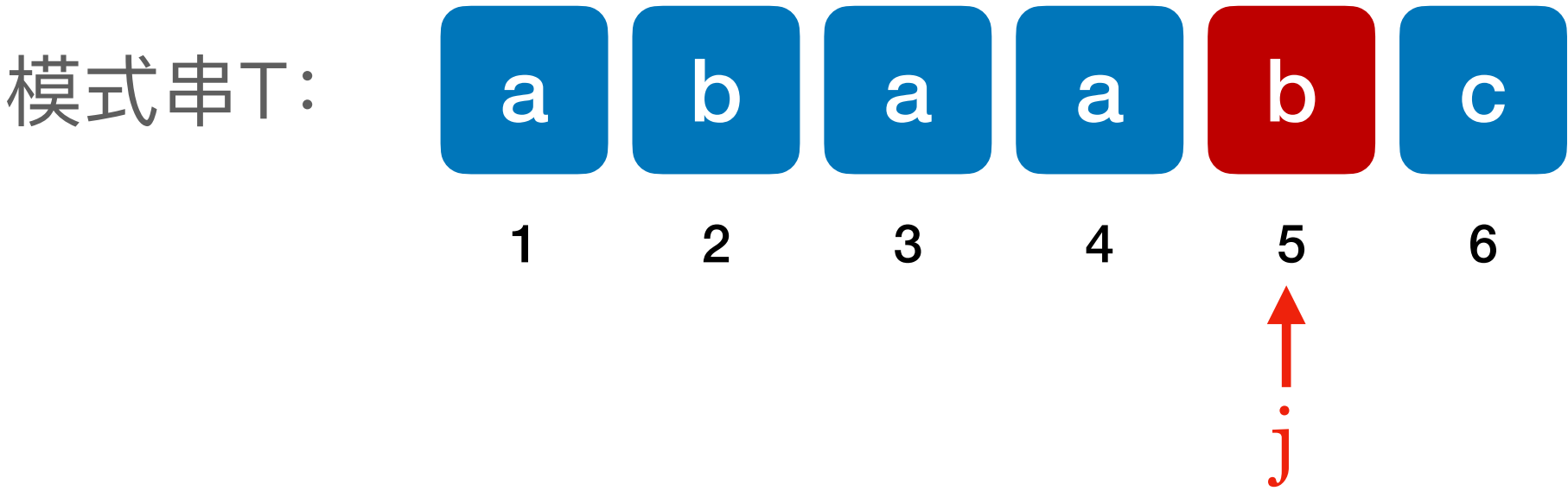
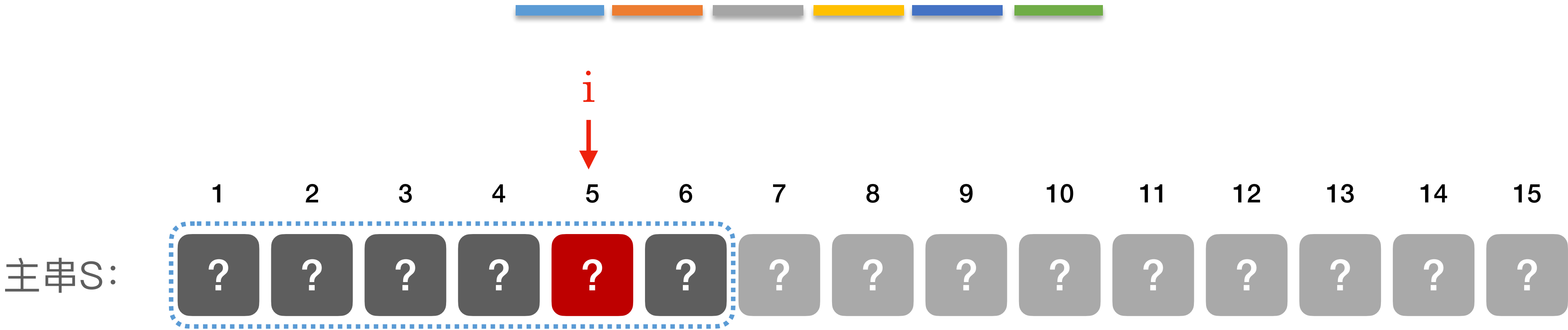
next数组的优化思路



next数组:

next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
	0	1	0	2	1	3

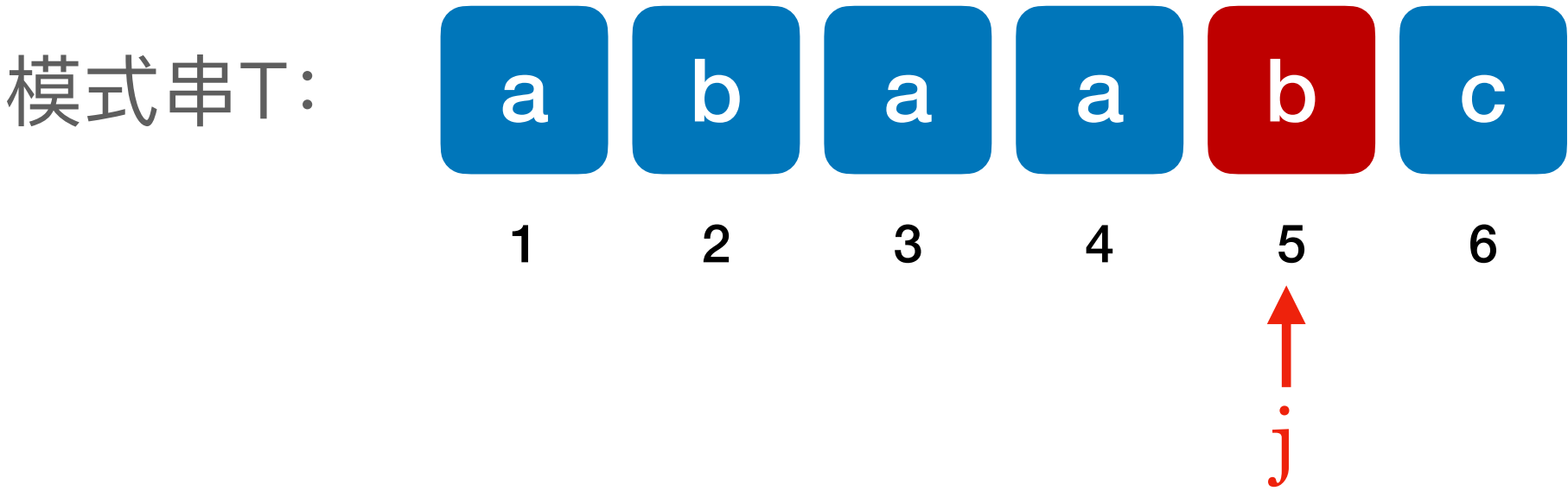
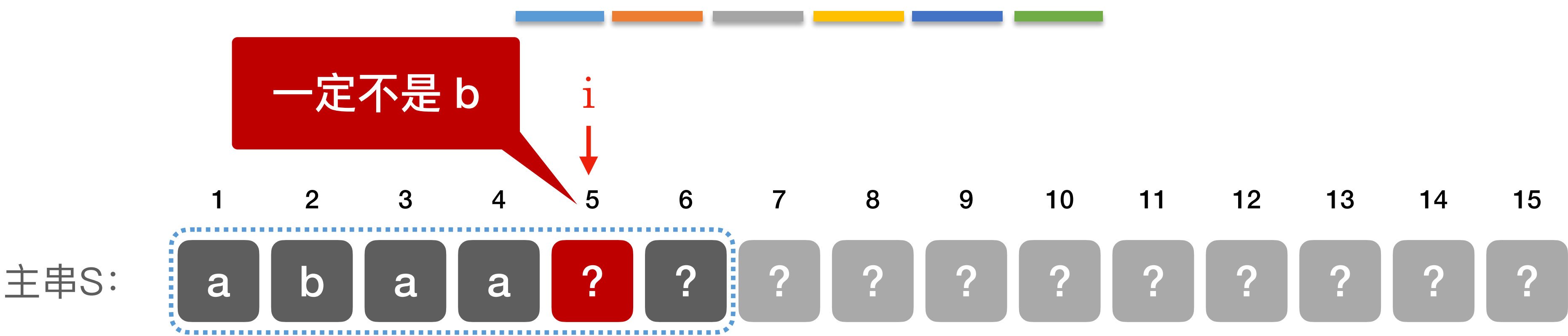
next数组的优化思路



next数组:

next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
	0	1	0	2	1	3

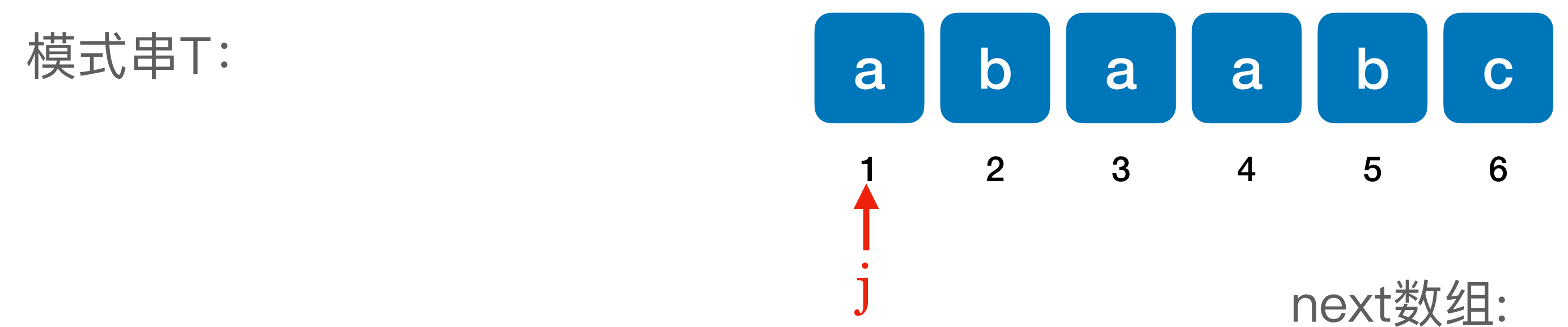
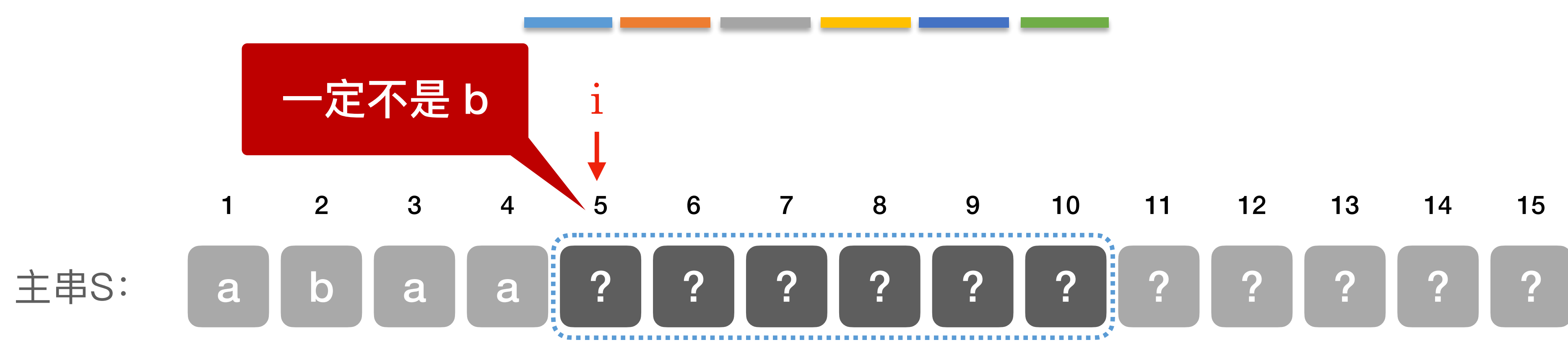
next数组的优化思路



next数组:

next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
	0	1	0	2	1	3

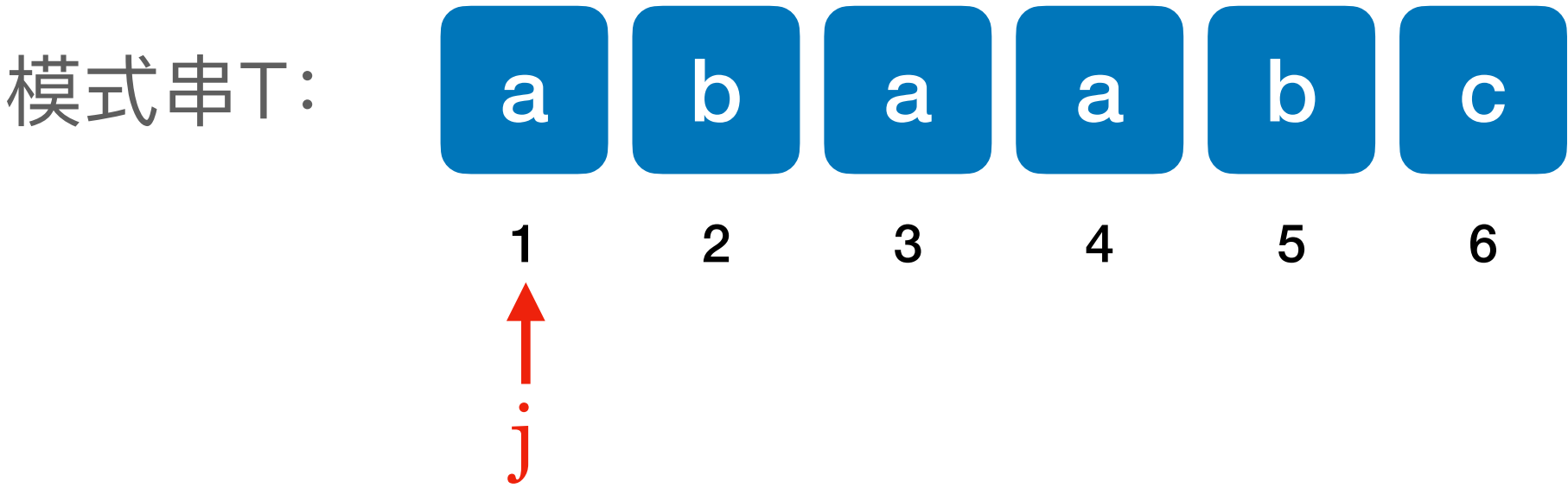
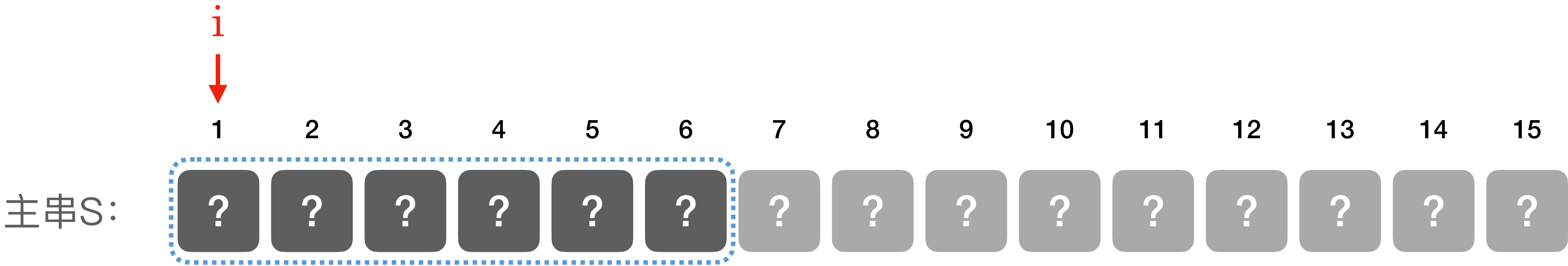
next数组的优化思路



next数组:

next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
	0	1	0	2	1	3

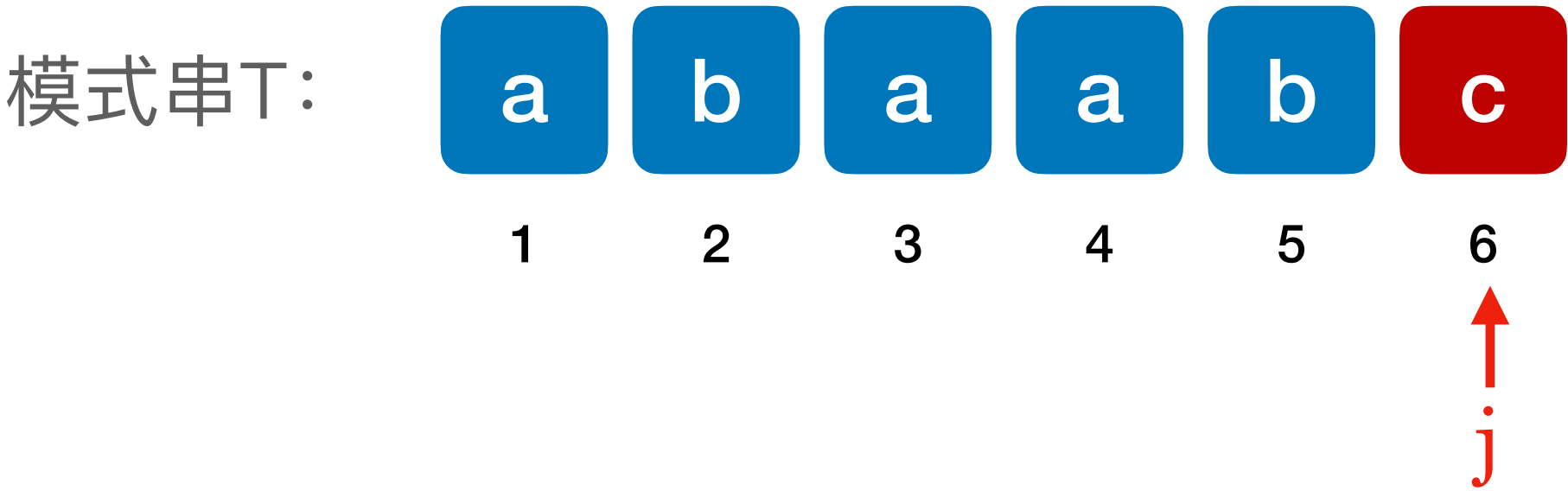
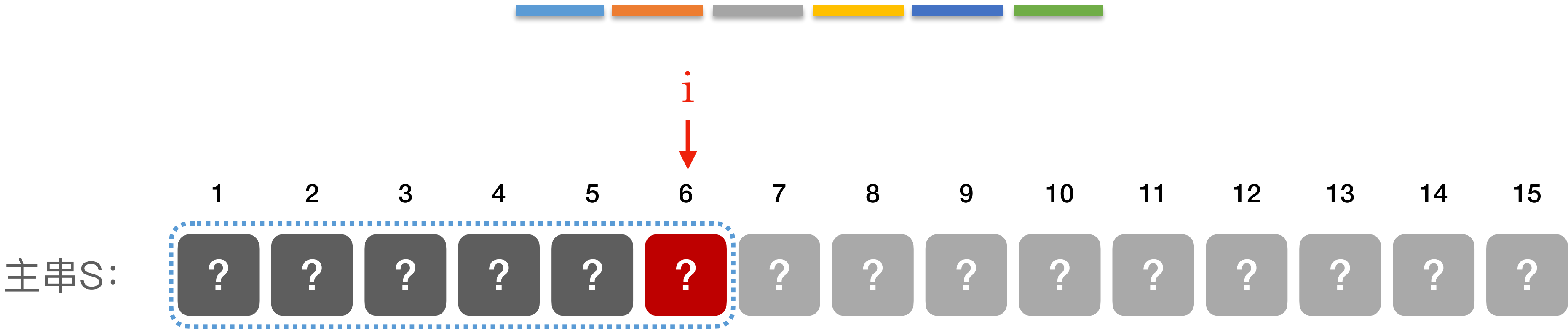
next数组的优化思路



next数组:

next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
	0	1	0	2	1	3

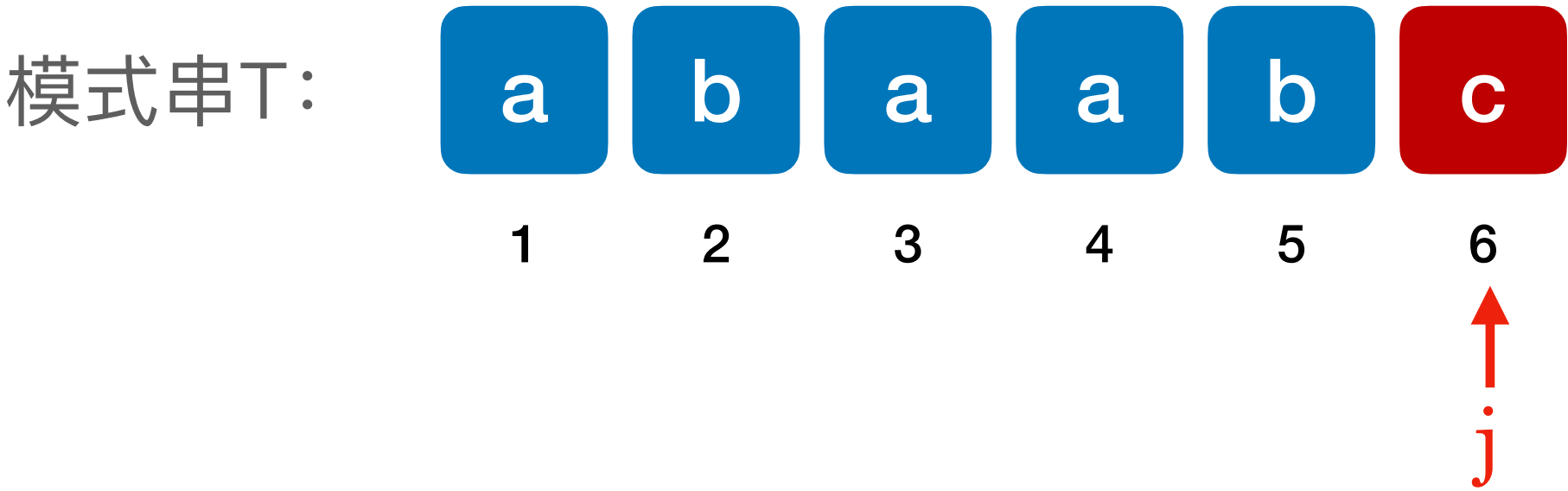
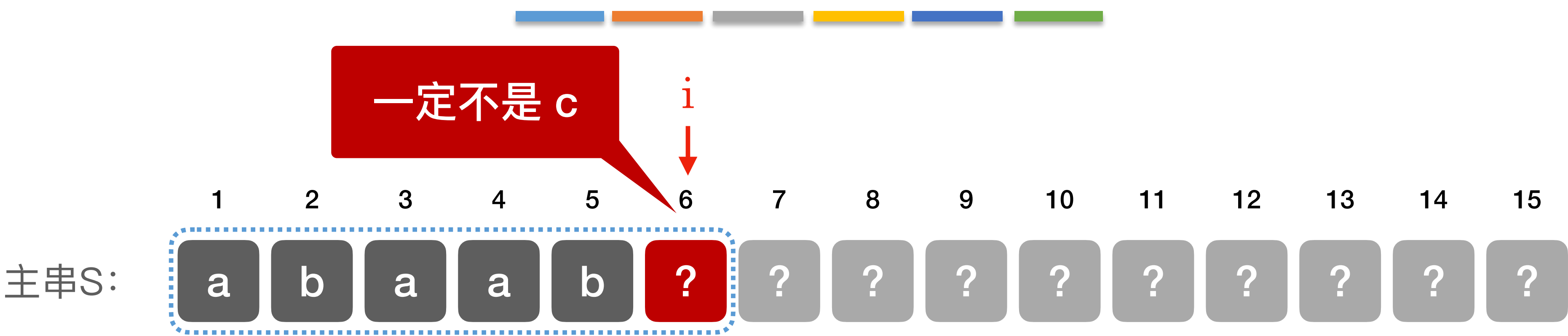
next数组的优化思路



next数组:

next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
	0	1	0	2	1	3

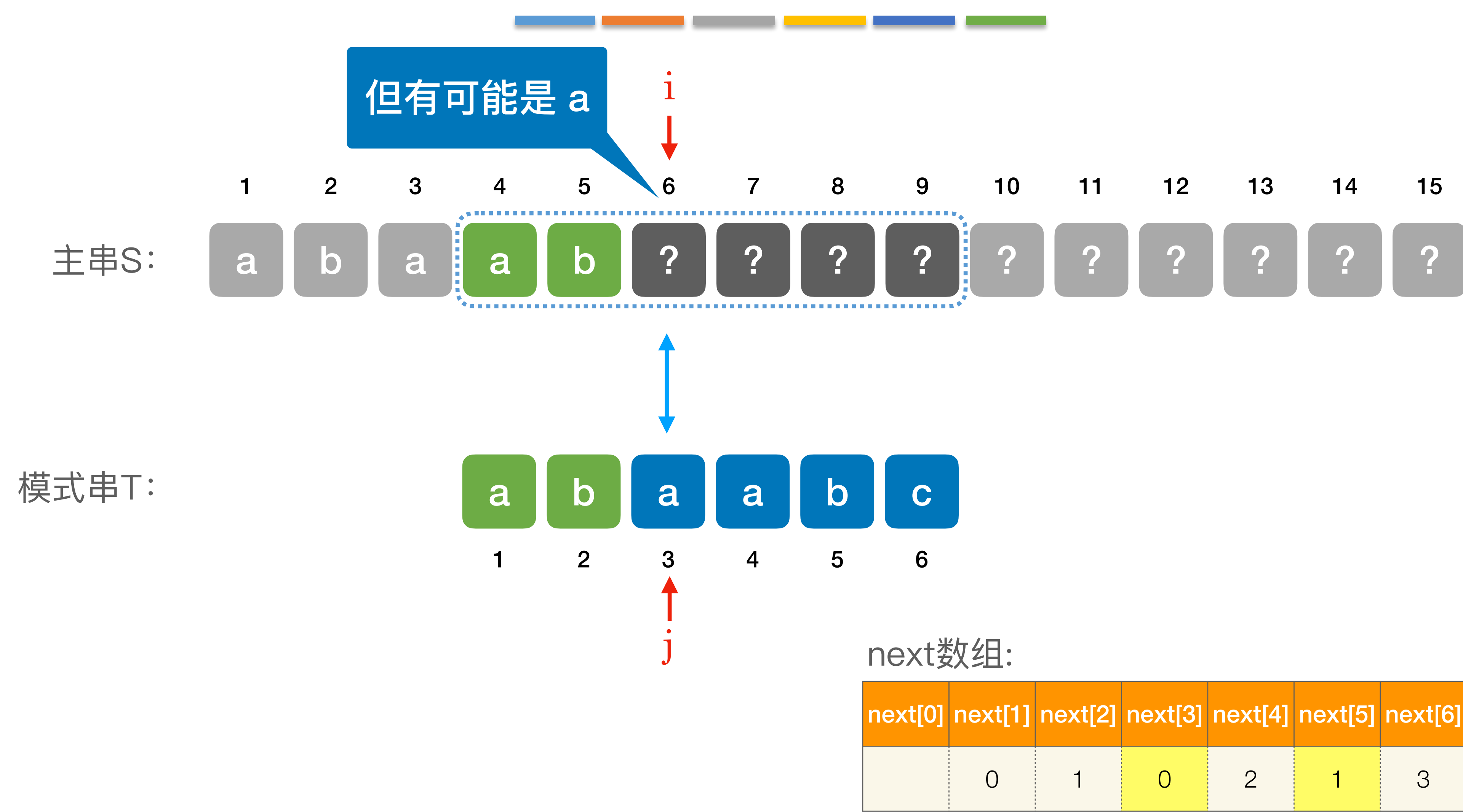
next数组的优化思路



next数组:

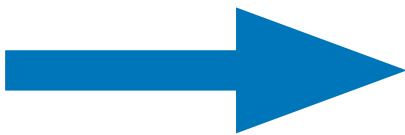
next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
	0	1	0	2	1	3

next数组的优化思路



KMP算法的进一步优化

根据模式串T，求出 next 数组



利用next数组进行匹配
(主串指针不回溯)

使用nextval数组

T = 'abaabc'

next数组:

next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
	0	1	1	2	2	3



优化

nextval数组:

nextval[0]	nextval[1]	nextval[2]	nextval[3]	nextval[4]	nextval[5]	nextval[6]
	0	1	0	2	1	3

```
int Index_KMP(SString S,SString T,int next[]){
    int i=1, j=1;
    while(i<=S.length&& j<=T.length){
        if(j==0 || S.ch[i]==T.ch[j]){
            ++i;
            ++j;           //继续比较后继字符
        }
        else
            j=next[j];     //模式串向右移动
    }
    if(j>T.length)
        return i-T.length; //匹配成功
    else
        return 0;
}
```

练习1：求nextval数组



ababaa

模式串 T = ababaa

序号j	1	2	3	4	5	6
模式串	a	b	a	b	a	a
next[j]	0	1	1	2	3	4

手算解题：先求next数组，再由next数组求nextval数组

```
nextval[1]=0;
for (int j=2; j<=T.length; j++) {
    if(T.ch[next[j]]==T.ch[j])
        nextval[j]=nextval[next[j]];
    else
        nextval[j]=next[j];
}
```

练习1：求nextval数组



ababaa

模式串 T = ababaa

序号j	1	2	3	4	5	6
模式串	a	b	a	b	a	a
next[j]	0	1	1	2	3	4

手算解题：先求next数组，再由next数组求nextval数组

```
nextval[1]=0;
for (int j=2; j<=T.length; j++) {
    if(T.ch[next[j]]==T.ch[j])
        nextval[j]=nextval[next[j]];
    else
        nextval[j]=next[j];
}
```

序号j	1	2	3	4	5	6
模式串	a	b	a	b	a	a
nextval[j]	0	1	0	1	0	4

练习2：求nextval数组



aaaab

模式串 T = aaaab

序号j	1	2	3	4	5
模式串	a	a	a	a	b
next[j]	0	1	2	3	4

手算解题：先求next数组，再由next数组求nextval数组

```
nextval[1]=0;
for (int j=2; j<=T.length; j++) {
    if(T.ch[next[j]]==T.ch[j])
        nextval[j]=nextval[next[j]];
    else
        nextval[j]=next[j];
}
```

练习2：求nextval数组



aaaab

模式串 T = aaaab

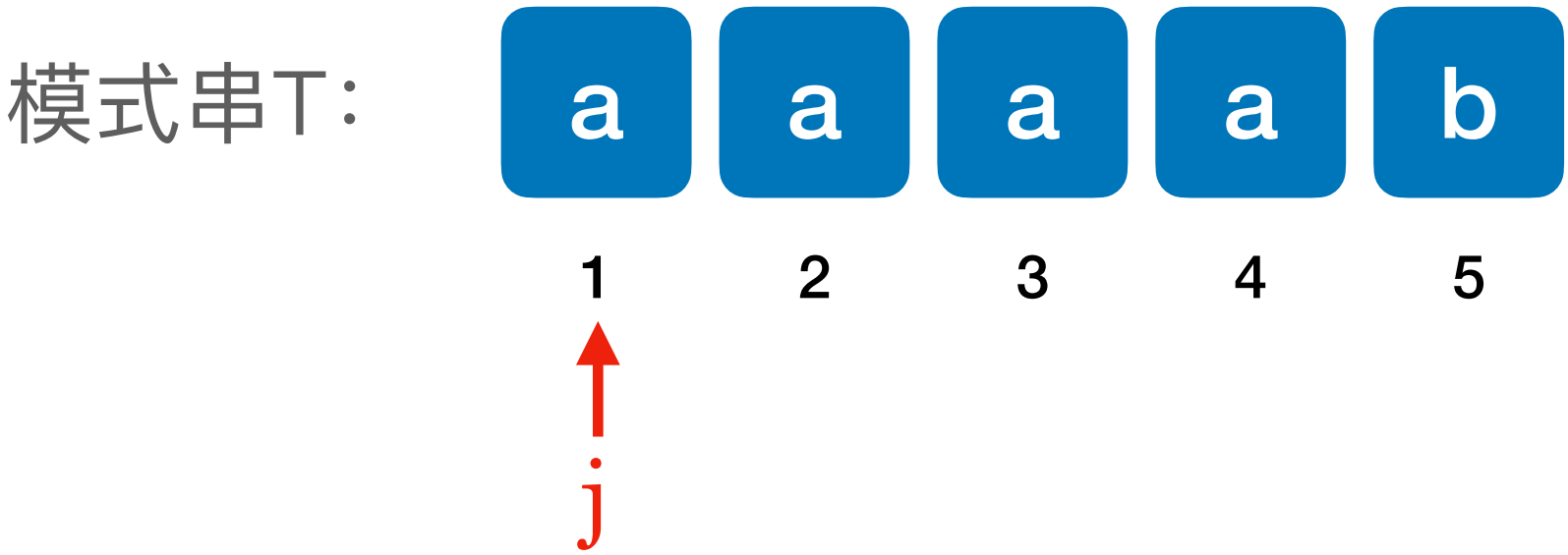
序号j	1	2	3	4	5
模式串	a	a	a	a	b
next[j]	0	1	2	3	4

手算解题：先求next数组，再由next数组求nextval数组

```
nextval[1]=0;
for (int j=2; j<=T.length; j++) {
    if(T.ch[next[j]]==T.ch[j])
        nextval[j]=nextval[next[j]];
    else
        nextval[j]=next[j];
}
```

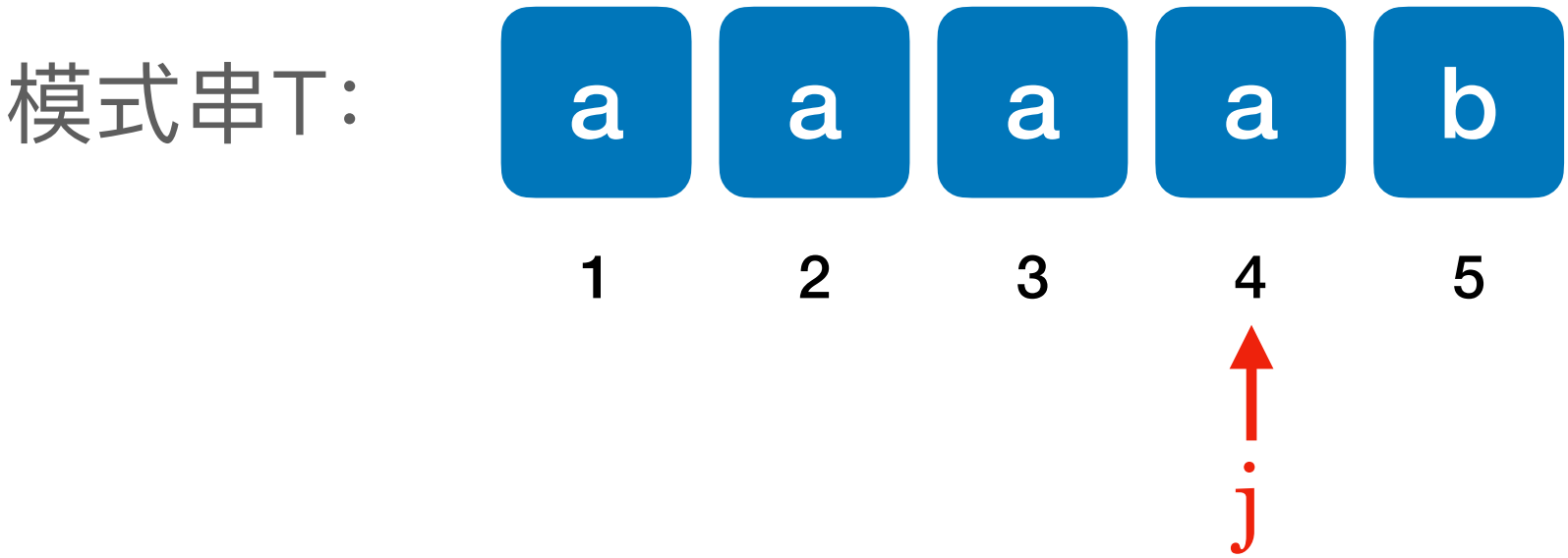
序号j	1	2	3	4	5
模式串	a	a	a	a	b
nextval[j]	0	0	0	0	4

例：KMP算法（优化前）



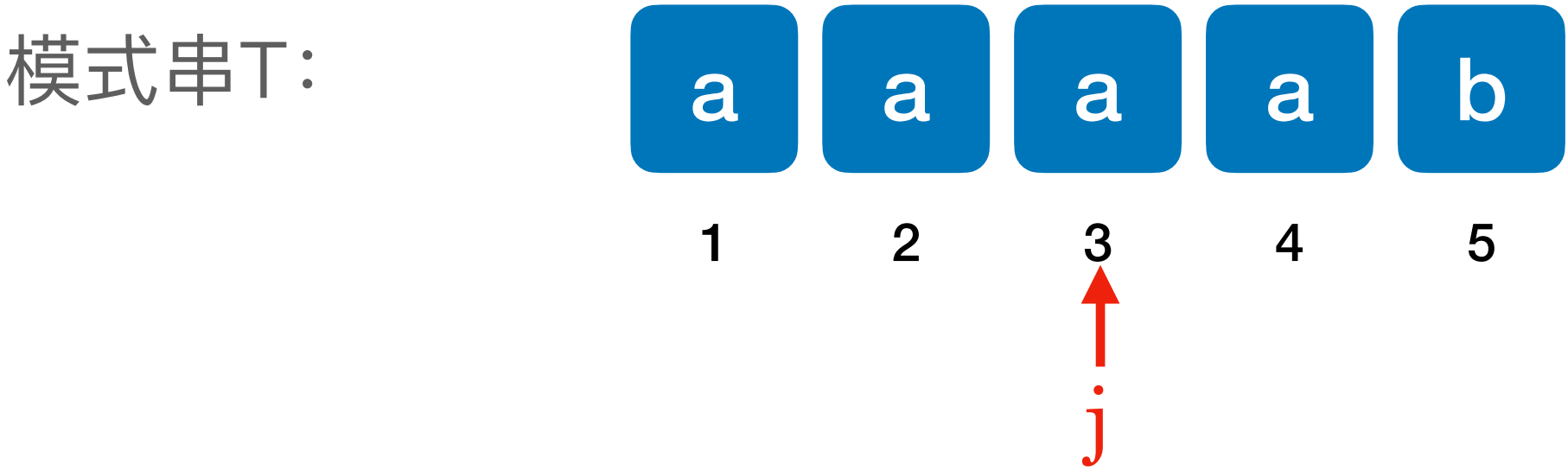
序号j	1	2	3	4	5
模式串	a	a	a	a	b
next[j]	0	1	2	3	4

例：KMP算法（优化前）



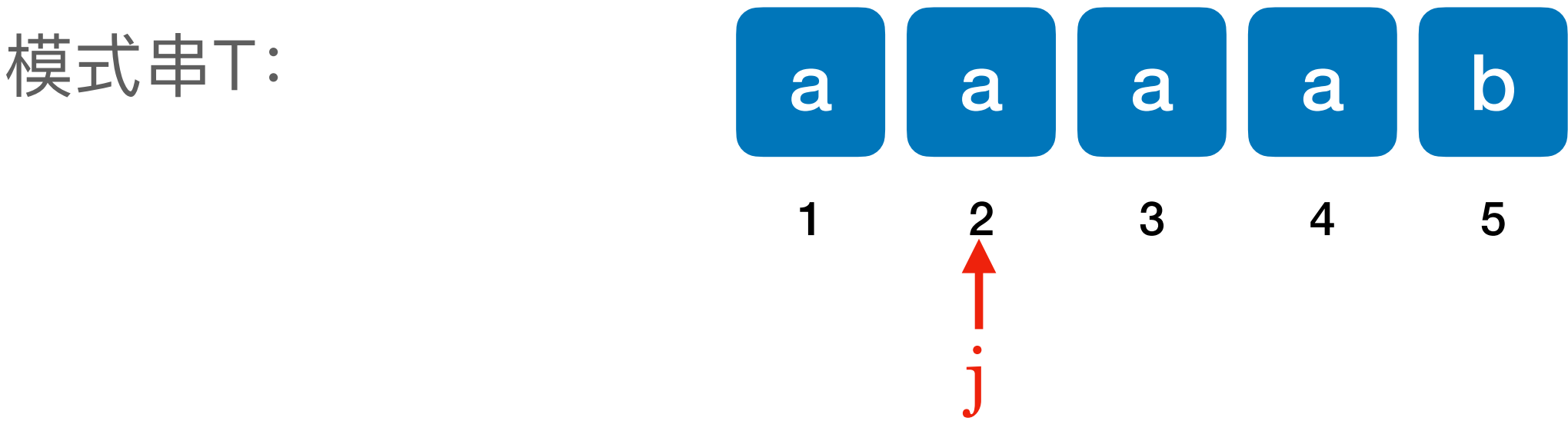
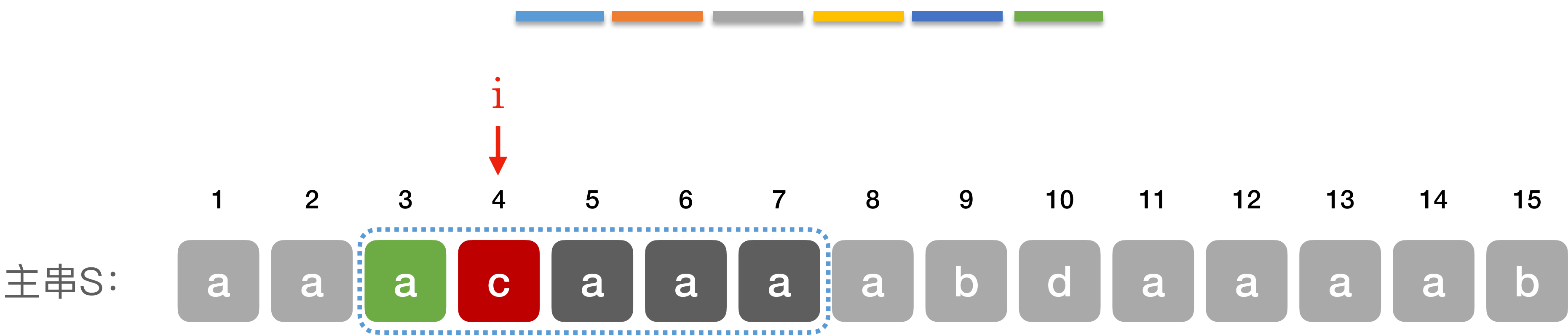
序号j	1	2	3	4	5
模式串	a	a	a	a	b
next[j]	0	1	2	3	4

例：KMP算法（优化前）



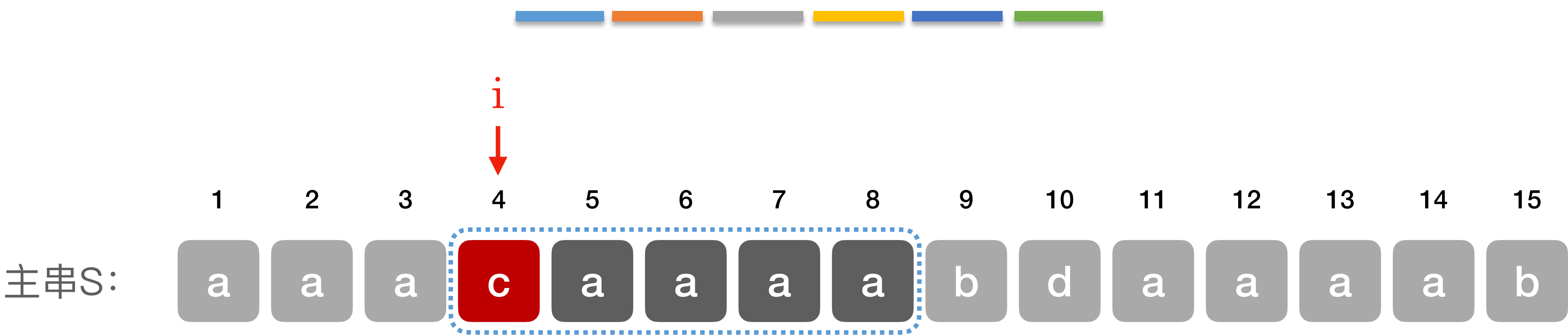
序号j	1	2	3	4	5
模式串	a	a	a	a	b
next[j]	0	1	2	3	4

例：KMP算法（优化前）

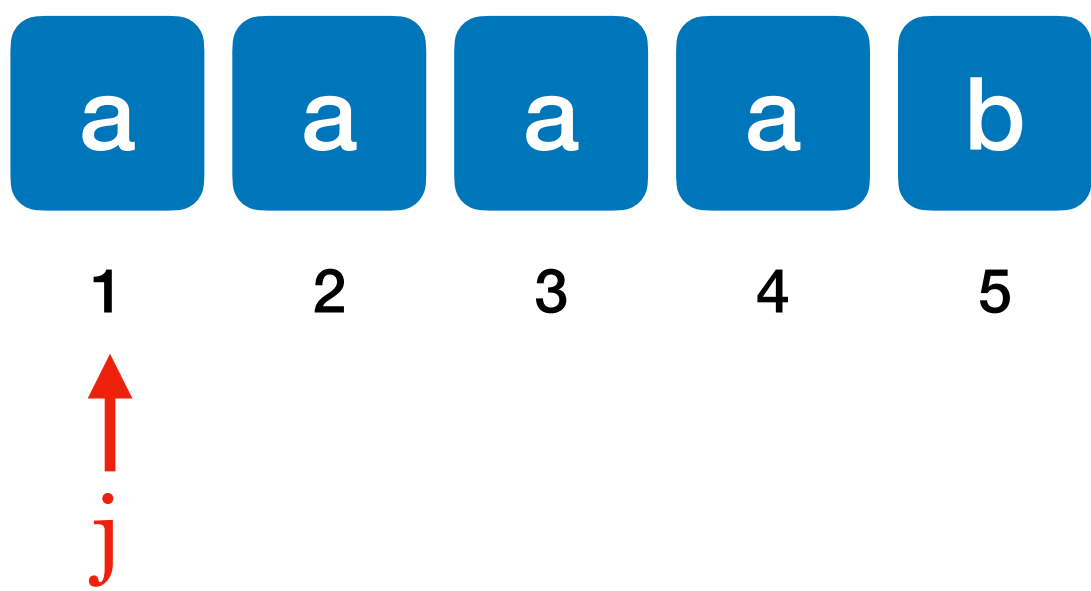


序号j	1	2	3	4	5
模式串	a	a	a	a	b
next[j]	0	1	2	3	4

例：KMP算法（优化前）

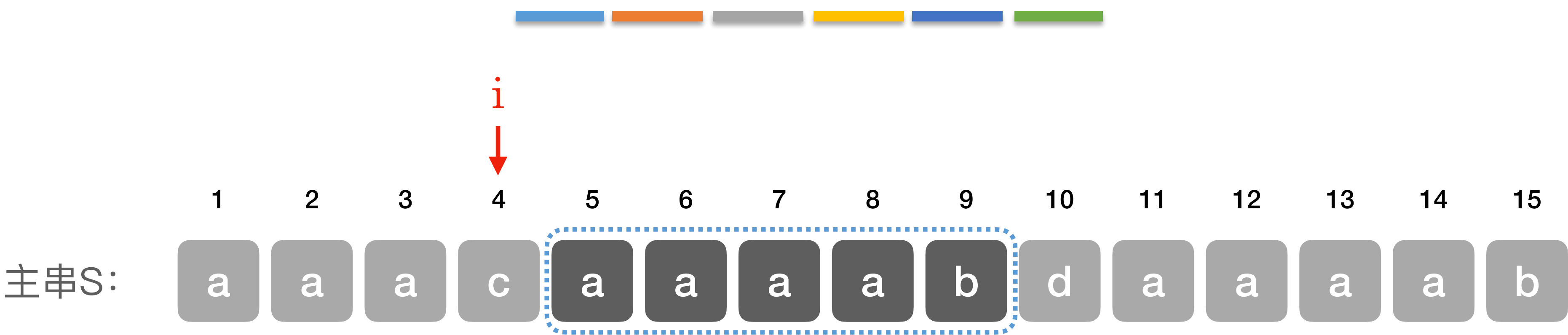


模式串T:

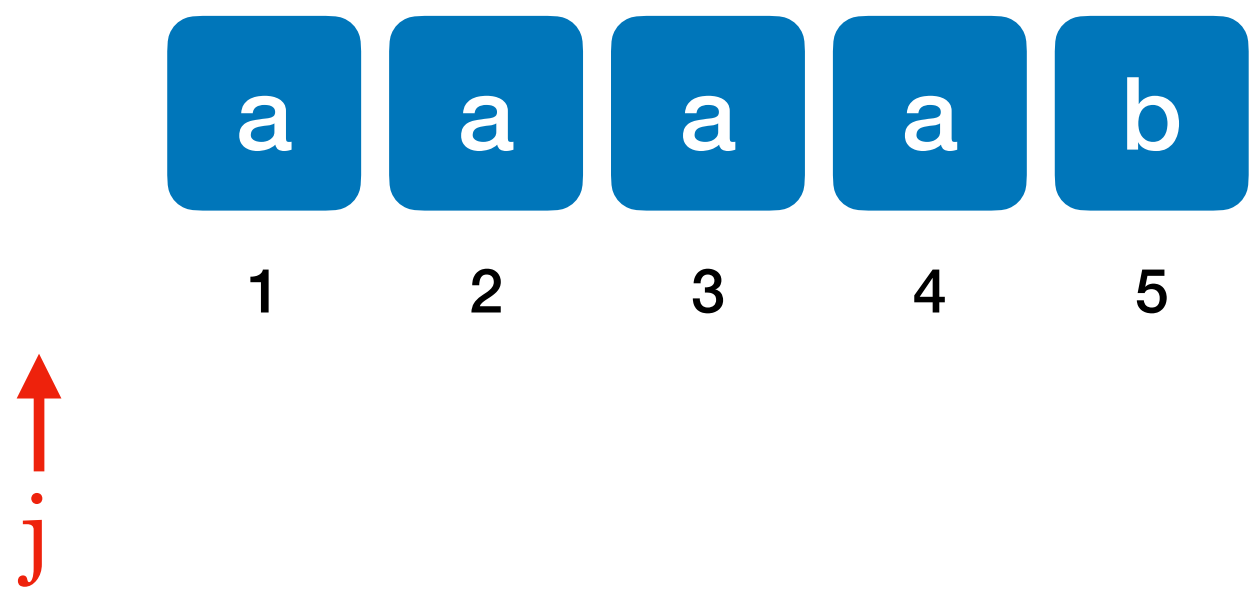


序号j	1	2	3	4	5
模式串	a	a	a	a	b
next[j]	0	1	2	3	4

例：KMP算法（优化前）



模式串T:

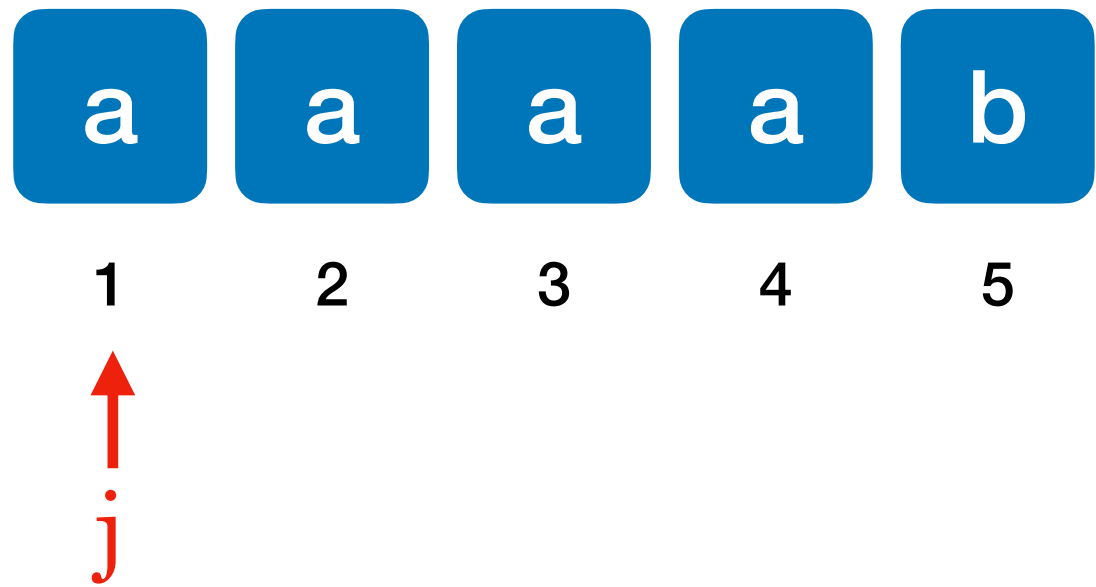


序号j	1	2	3	4	5
模式串	a	a	a	a	b
next[j]	0	1	2	3	4

例：KMP算法（优化前）

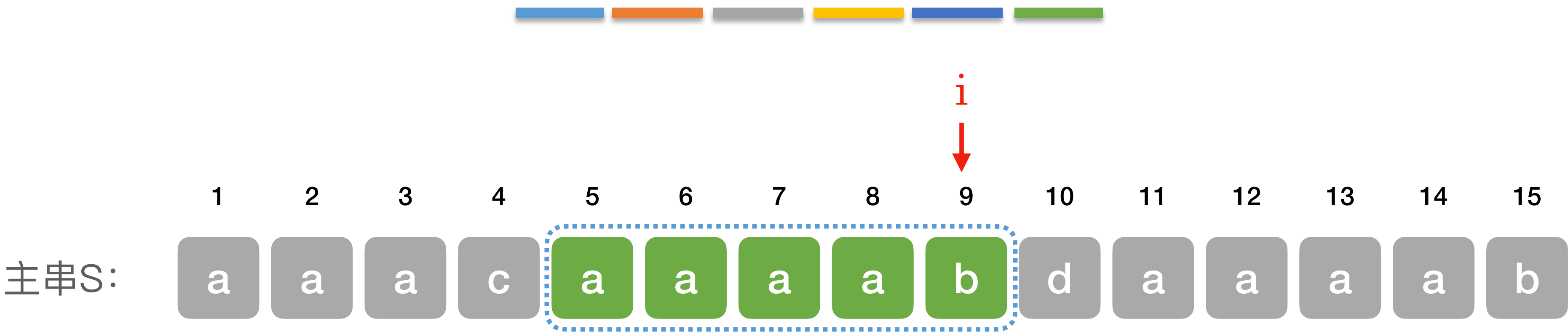


模式串T:

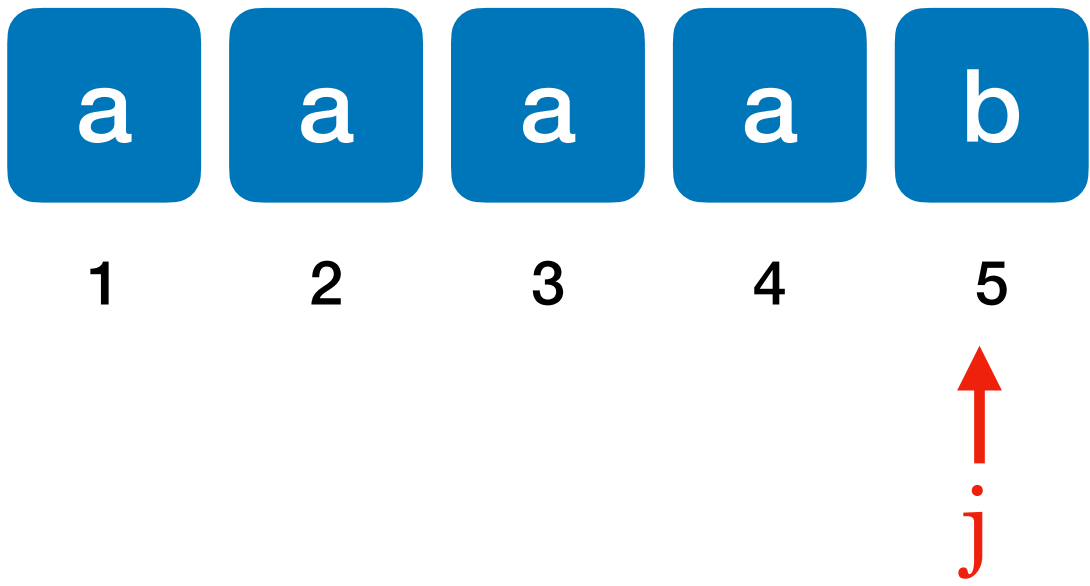


序号j	1	2	3	4	5
模式串	a	a	a	a	b
next[j]	0	1	2	3	4

例：KMP算法（优化前）

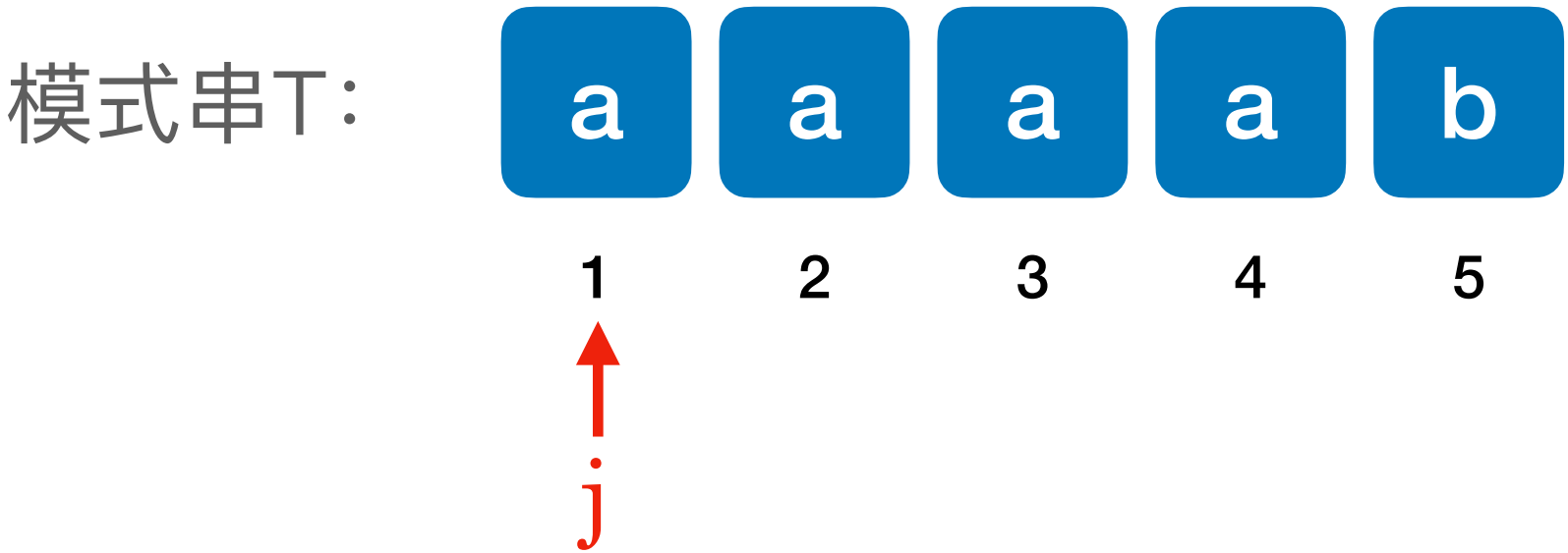
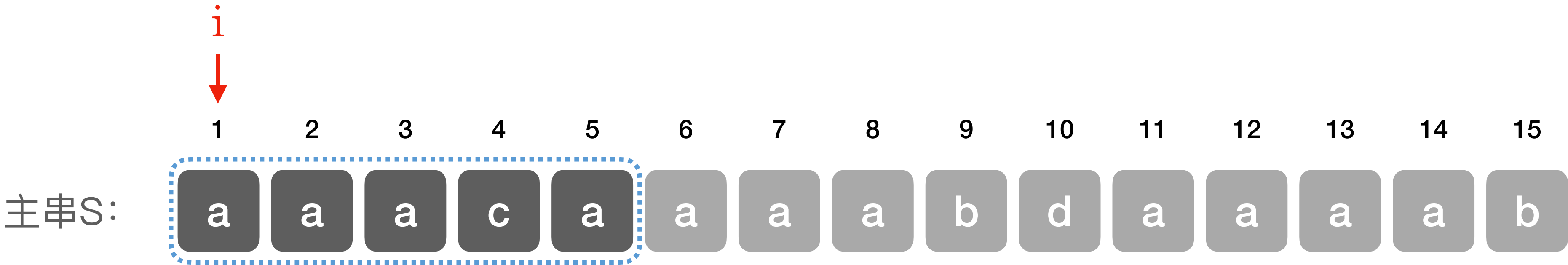


模式串T:



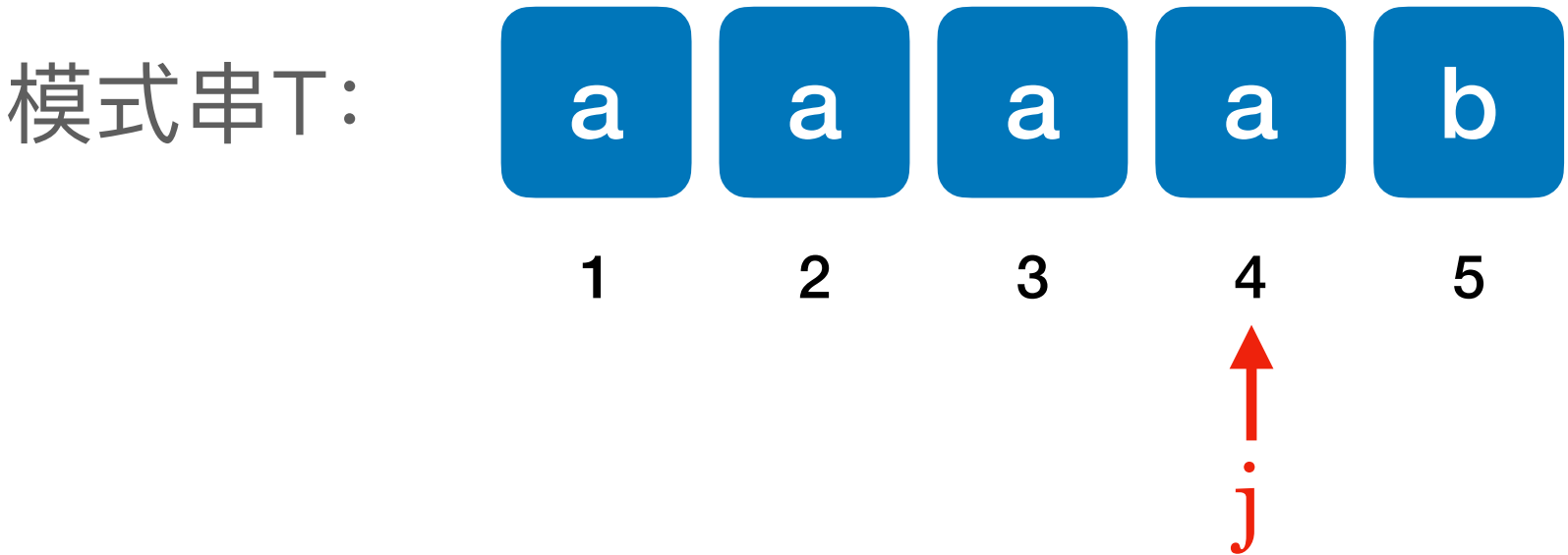
序号j	1	2	3	4	5
模式串	a	a	a	a	b
next[j]	0	1	2	3	4

例：KMP算法（优化后）



序号j	1	2	3	4	5
模式串	a	a	a	a	b
nextval[j]	0	0	0	0	4

例：KMP算法（优化后）

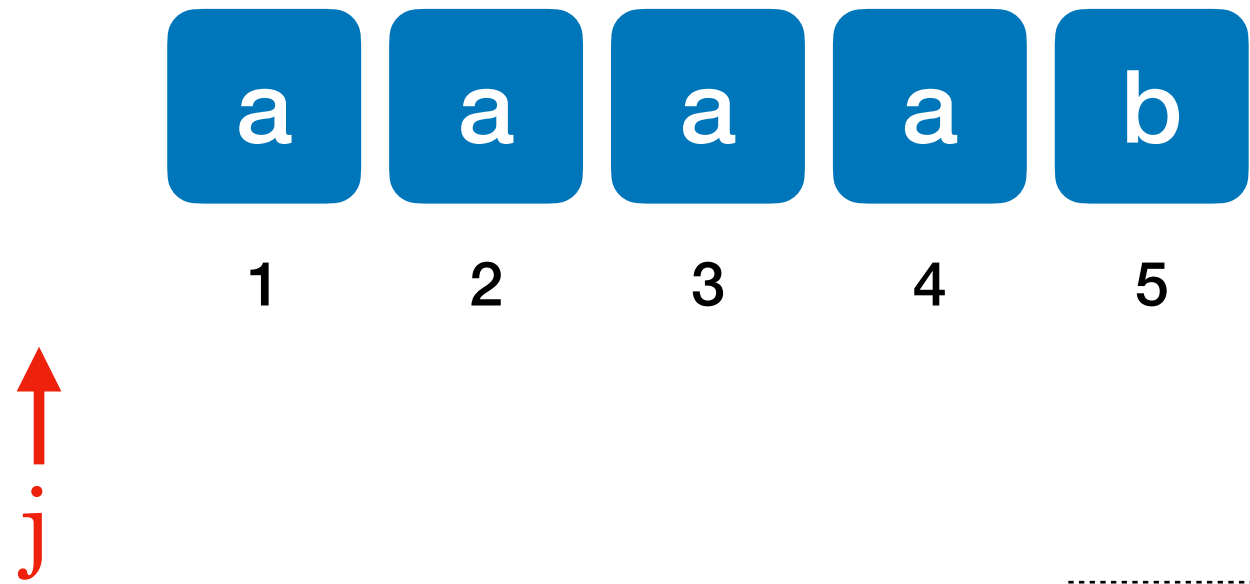


序号j	1	2	3	4	5
模式串	a	a	a	a	b
nextval[j]	0	0	0	0	4

例：KMP算法（优化后）

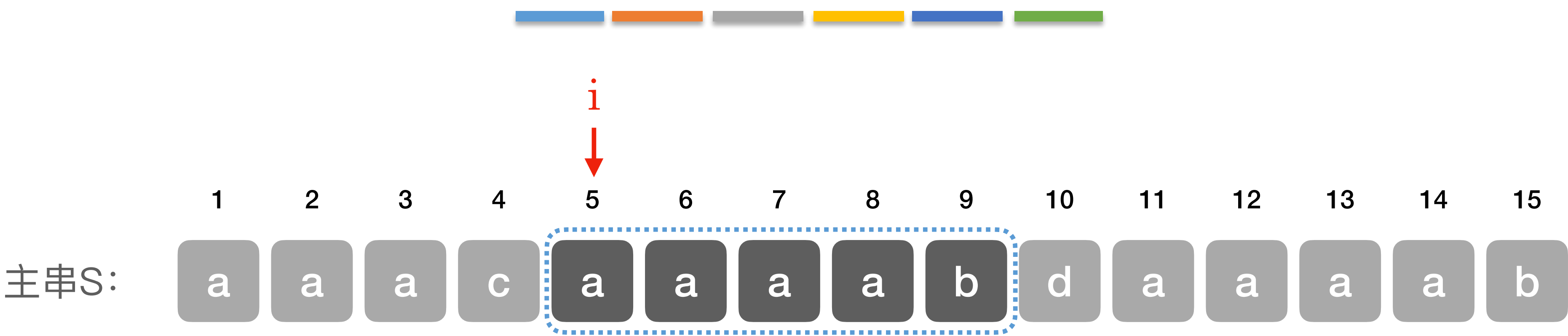


模式串T:

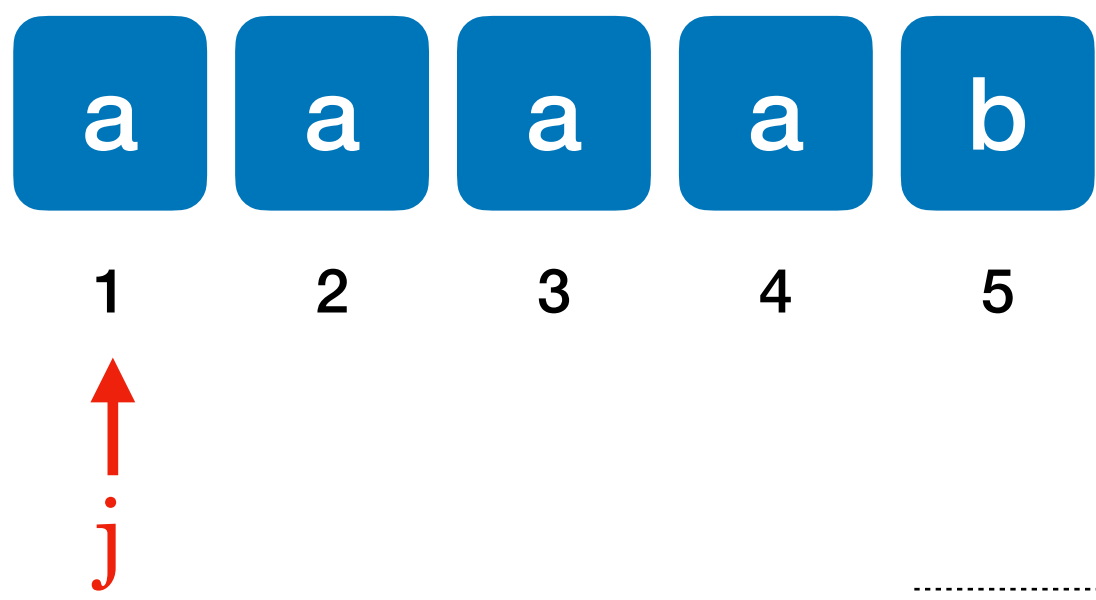


序号j	1	2	3	4	5
模式串	a	a	a	a	b
nextval[j]	0	0	0	0	4

例：KMP算法（优化后）

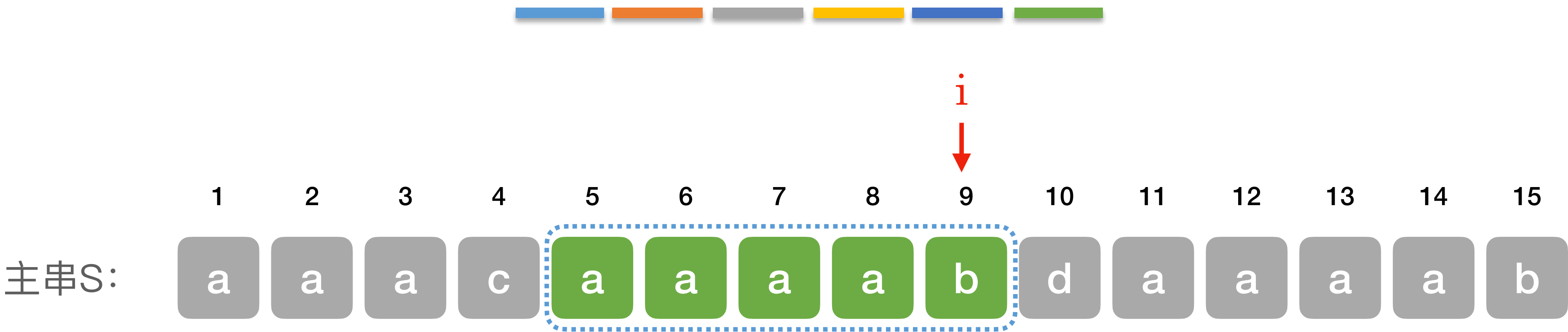


模式串T:

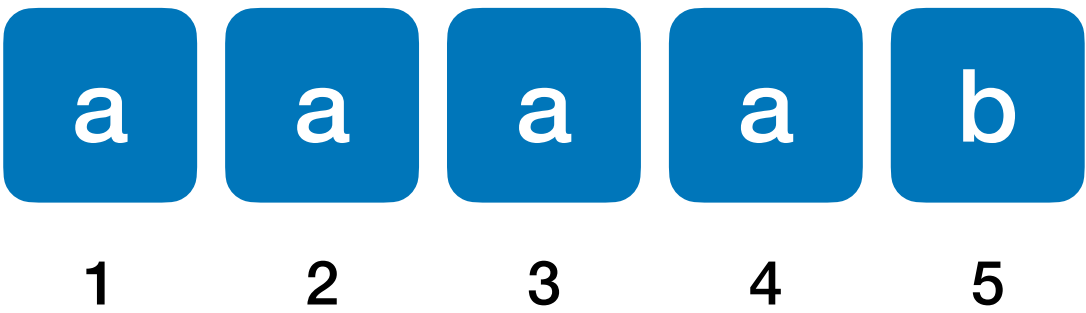


序号j	1	2	3	4	5
模式串	a	a	a	a	b
nextval[j]	0	0	0	0	4

例：KMP算法（优化后）



模式串T:



序号j	1	2	3	4	5
模式串	a	a	a	a	b
nextval[j]	0	0	0	0	4

补充学习：B站搜索“王道数据结构”–旧版 KMP

王道计算机考研 数据结构

441.0万播放 · 总弹幕数6.2万 2020-02-27 22:02:46

giao

奥利给

别优了哥

yyds

从22年视频过来

随着算法优化，我也在优化了

大哥们该睡了，别卷了，，，

空降到5:32

孩子都

32

王道论坛

bilibili

例1: KMP算法存在的问题

32

优化。。

	g	o	o	g	l	e
1	2	3	4	5	6	

序号j	1	2	3	4	5	6
模式串	g	o	o	g	l	e
next[j]	0	1	1	1	2	1

王道考研/CSKAOYAN.COM

8 人正在看, 已装填 367 条弹幕

弹

发个友善的弹幕见证当下

弹幕礼仪 >

发送



王道论坛

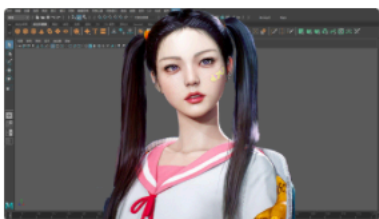
发消息

王道论坛www.cskaoyan.com

已关注 25.6万

弹幕列表

展开



每天建模1小时，挑战接单赚钱

广告 3D游戏兼职赚钱

视频选集 (37/90)

自动连播

- P34 【旧版】4.2_1_串的朴素模式匹配算法 11:33
- P35 【旧版】4.2_2_KMP算法(上) 16:31
- P36 【旧版】4.2_3_KMP算法(下) 17:31
- P37 【旧版】4.2_4_KMP算法(下)**
- P38 【旧版】5.1.1 树的定义和基本术语 15:17
- P39 【旧版】5.1.2 树的性质 05:51
- P40 【旧版】5.2_1_二叉树的定义和基本... 12:46
- P41 【旧版】5.2_2_二叉树的性质 07:30
- P42 【旧版】5.2_3_二叉树的存储结构 10:45
- P43 【旧版】5.3_1_二叉树的先中后序遍... 23:08



2023徐涛考研政治精讲【最新合集】| B站独家
考研政治徐涛

本章在408中的地位

IV

考查内容

数据结构

[考查目标]

1. 掌握数据结构的基本概念、基本原理和基本方法。
2. 掌握数据的逻辑结构、存储结构及基本操作的实现,能够对算法进行基本的时间复杂度与空间复杂度的分析。
3. 能够运用数据结构的基本原理和方法进行问题的分析与求解,具备采用 C 或 C++ 语言设计与实现算法的能力。

一、线性表

- (一) 线性表的基本概念
- (二) 线性表的实现
 1. 顺序存储
 2. 链式存储
- (三) 线性表的应用

二、栈、队列和数组

- (一) 栈和队列的基本概念
- (二) 栈和队列的顺序存储结构

- (三) 栈和队列的链式存储结构
- (四) 多维数组的存储
- (五) 特殊矩阵的压缩存储
- (六) 栈、队列和数组的应用

三、树与二叉树

- (一) 树的基本概念
- (二) 二叉树
 1. 二叉树的定义及其主要特性
 2. 二叉树的顺序存储结构和链式存储结构
 3. 二叉树的遍历
 4. 线索二叉树的基本概念和构造
- (三) 树、森林
 1. 树的存储结构
 2. 森林与二叉树的转换
 3. 树和森林的遍历
- (四) 树与二叉树的应用
 1. 二叉搜索树
 2. 平衡二叉树
 3. 哈夫曼(Huffman)树和哈夫曼编码

四、图

- (一) 图的基本概念
- (二) 图的存储及基本操作
 1. 邻接矩阵法
 2. 邻接表法
 3. 邻接多重表、十字链表
- (三) 图的遍历
 1. 深度优先搜索
 2. 广度优先搜索



本章在408中的地位



你好像没有任何牌面吧？



卑微

- (四) 图的基本应用
1. 最小(代价)生成树
 2. 最短路径
 3. 拓扑排序
 4. 关键路径

五、查找

- (一) 查找的基本概念
- (二) 顺序查找法
- (三) 分块查找法
- (四) 折半查找法
- (五) B 树及其基本操作、B⁺树的基本概念
- (六) 散列 (Hash) 表
- (七) 字符串模式匹配
- (八) 查找算法的分析及应用

六、排序

- (一) 排序的基本概念
- (二) 插入排序
1. 直接插入排序
 2. 折半插入排序
- (三) 起泡排序 (Bubble Sort)
- (四) 简单选择排序
- (五) 希尔排序 (Shell Sort)
- (六) 快速排序
- (七) 堆排序
- (八) 二路归并排序 (Merge Sort)
- (九) 基数排序
- (十) 外部排序
- (十一) 各种排序算法的比较

(十二) 排序算法的应用

计算机组成原理

[考查目标]

1. 理解单处理器计算机系统中各部件的内部工作原理、组成结构以及相互连接方式,具有完整的计算机系统的整机概念。
2. 理解计算机系统层次化结构概念,熟悉硬件与软件之间的界面,掌握指令集体系结构的基本知识和基本实现方法。
3. 能够综合运用计算机组成的基本原理和基本方法,对有关计算机硬件系统中的理论和实际问题进行计算、分析,对一些基本部件进行简单设计;并能对高级程序设计语言(如 C 语言)中的相关问题进行分析。

一、计算机系统概述

(一) 计算机系统层次结构

1. 计算机系统的基本组成
2. 计算机硬件的基本组成
3. 计算机软件和硬件的关系
4. 计算机系统的工作过程

(二) 计算机性能指标

吞吐量、响应时间, CPU 时钟周期、主频、CPI、CPU 执行时间, MIPS、MFLOPS、GFLOPS、TFLOPS、PFLOPS、EFLOPS、ZFLOPS。

二、数据的表示和运算

(一) 数制与编码

1. 进位计数制及其相互转换



欢迎大家对本节视频进行评价~



学员评分：4.2.3 KMP...

扫一扫二维码打开或分享给好友



– 腾讯文档 –

可多人实时在线编辑，权限安全可控



公众号：王道在线



b站：王道计算机教育



抖音：王道计算机考研