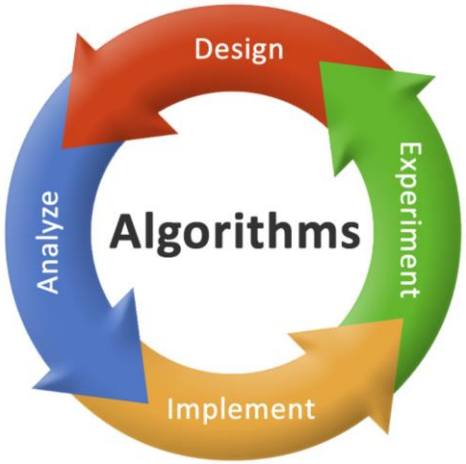


Algorithm Design and Analysis

算法设计与分析

Chapter 10: KMP

张乾坤

华中科技大学网络空间安全学院
School of Cyber Science and Engineering, HUST
华中科技大学
HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

字符串模式匹配

- 线性存储的一组数据（默认是字符）
- 特殊操作集
  - 求串的长度
  - 比较两串是否相等
  - 两串相接
  - 求子串
  - 插入子串
  - 匹配子串
  - 删除子串

字符串模式匹配

目 标

给定一段文本，从中找出某个指定的关键字。

例如从一本 **Thomas Love Peacock** 写于十九世纪的小说《 **Headlong Hall** 》中找到那个最长的单词
osseocarnisanguineoviscericartilaginonervomedullary

或者从古希腊喜剧《 **Assemblywomen** 》中找到一道菜的名字
Lopadotemachoselachogaleokranioleipsanodrimhypotrimmatosilphioparaomelitokatakechymenokichlepikossyphophattoperisteralektryonoptekephalliokigklopeleiolagoiosiraiobaphetraganopterygon

字符串模式匹配

目 标

给定一段文本：**string** = $s_0 s_1 \cdots s_{n-1}$

给定一个模式：**pattern** = $p_0 p_1 \cdots p_{m-1}$

求 **pattern** 在 **string** 中出现的位置

```
Position PatternMatch(char *string, char *pattern)
```

字符串模式匹配-Naive

■ 方法1：C的库函数 strstr

```
char *strstr(char *string, char *pattern)
```

```
#include <stdio.h>
#include <string.h>

typedef char* Position;

int main()
{   char string[] = "This is a simple example.";
    char pattern[] = "simple";
    Position p = strstr(string, pattern);
    printf("%s\n", p);
    return 0;
}
```

```
simple example.

--------------------------------
Process exited after 0.665 seconds with return value 0
请按任意键继续. . .
```

字符串模式匹配-Naive

■ 方法1：C的库函数 strstr

```
char *strstr(char *string, char *pattern)
```

```
#include <stdio.h>
#include <string.h>

typedef char* Position;
#define NotFound NULL
int main()
{   char string[] = "This is a simple example.";
    char pattern[] = "sample";
    Position p = strstr(string, pattern);
    if ( p == NotFound ) printf("Not Found.\n");
    else printf("%s\n", p);
    return 0;
}
```

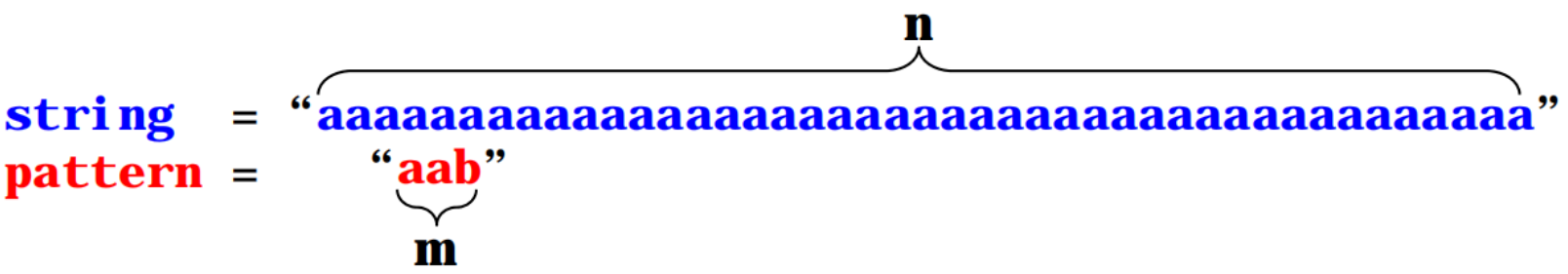
```
Not Found.

--------------------------------
Process exited after 0.1277 seconds with return value 0
请按任意键继续. . .
```

字符串模式匹配-Naive

■ 方法1：C的库函数 strstr

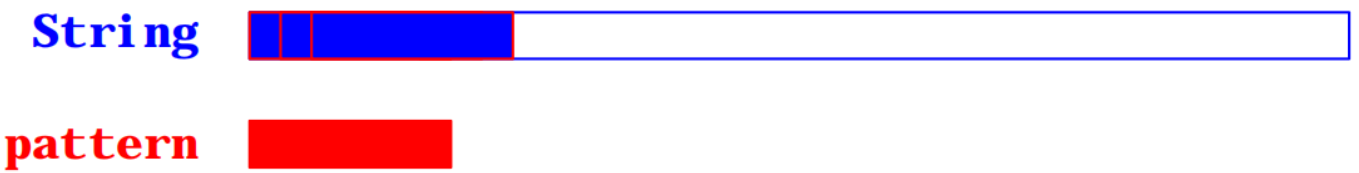
```
char *strstr(char *string, char *pattern)
```



$$T = O(n \cdot m)$$

字符串模式匹配-Naive

■ 方法2：从末尾开始比



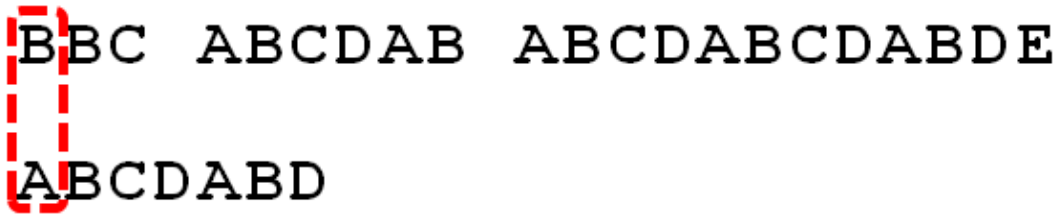
string = “aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa”
pattern = “aab”

$$T = O(n)$$

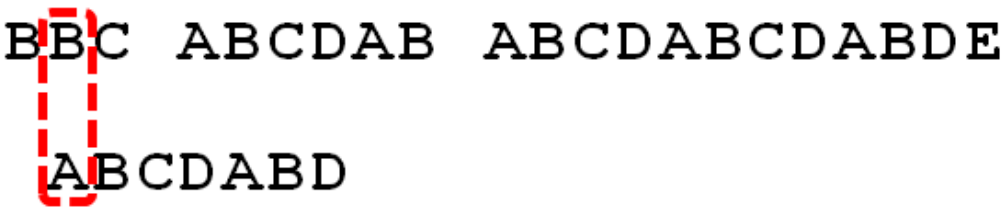
string = “aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa”
pattern = “baa”

Knuth-Morris-Pratt算法

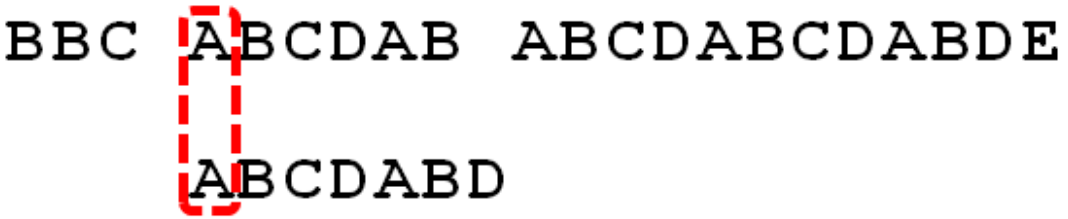
1



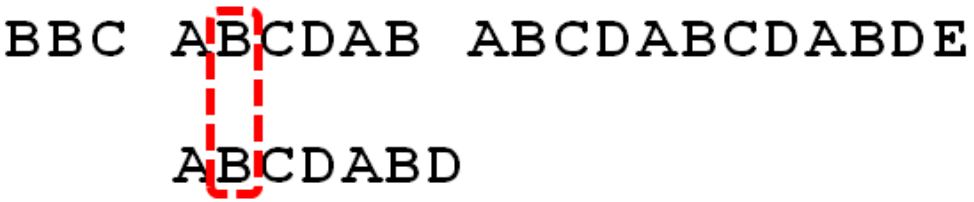
2



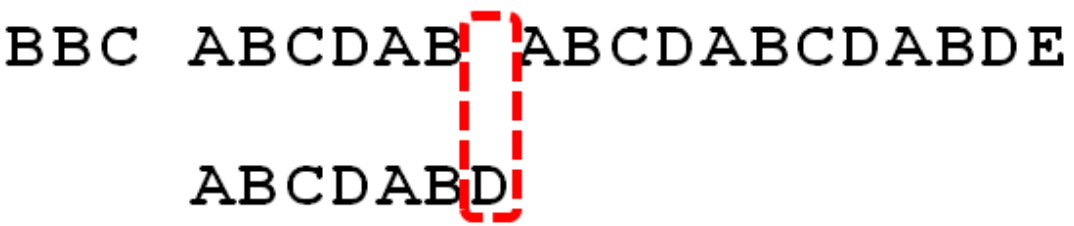
3



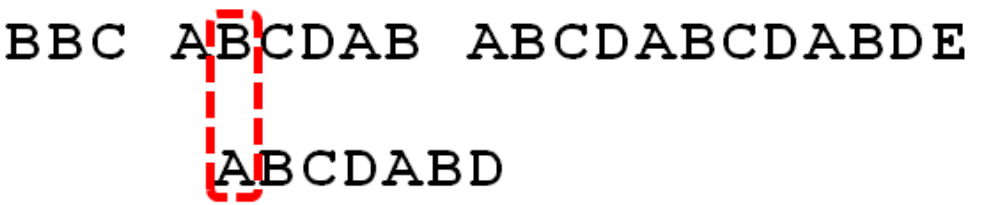
4



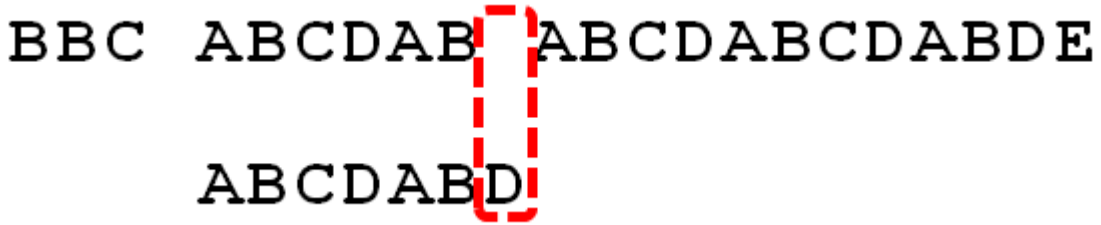
5



6



Knuth-Morris-Pratt算法

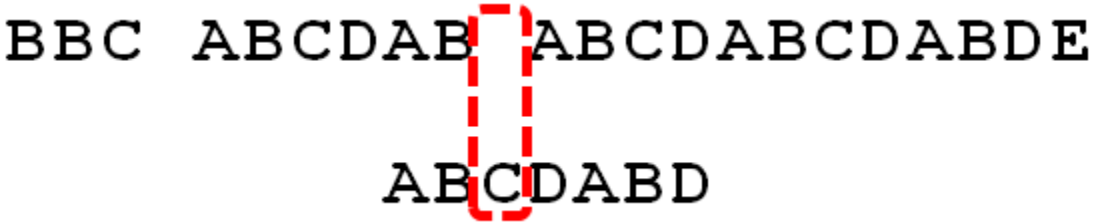


已经看过前面的ABCDAB
不应该回到B，应回到AB

假设有一张这样的表
部分匹配表

搜索词	A	B	C	D	A	B	D
部分匹配值	0	0	0	0	1	2	0

移动位数 = 已匹配的字符数 - 对应的部分匹配值
4 = 6 - 2

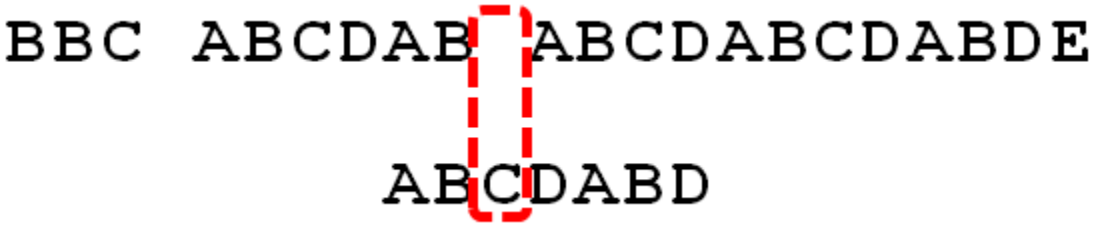


Knuth-Morris-Pratt算法

假设有一张这样的表

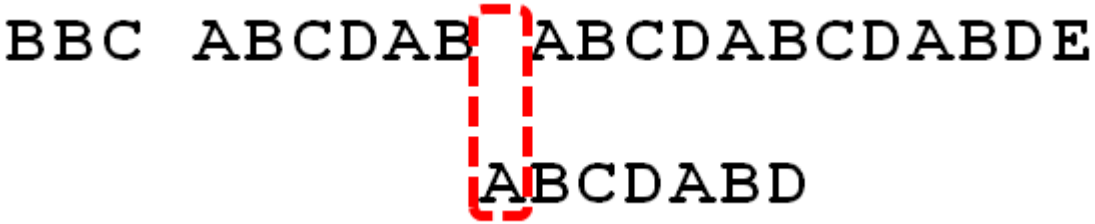
部分匹配表

搜索词	A	B	C	D	A	B	D
部分匹配值	0	0	0	0	1	2	0



移动位数 = 已匹配的字符数 - 对应的部分匹配值

2 = 2 - 0



Knuth-Morris-Pratt算法

假设有一张这样的表
部分匹配表

搜索词	A	B	C	D	A	B	D
部分匹配值	0	0	0	0	1	2	0

```
BBC ABCDAB ABCDABCDABDE
           ABCDABD
```

移动位数 = 已匹配的字符数 - 对应的部分匹配值
4 = 6 - 2

```
BBC ABCDAB ABCDABCDABDE
               ABCDABD
```

时间复杂度?
O(m+n)

Knuth-Morris-Pratt算法

- 如何产生“部分匹配表”?

两个概念:

- “前缀”：除了最后一个字符以外，一个字符串的全部头部组合
- “后缀”：除了第一个字符以外，一个字符串的全部尾部组合

字符串：　“bread”

前缀：　b，br，bre，brea

后缀：　read，ead，ad，d

Knuth-Morris-Pratt算法

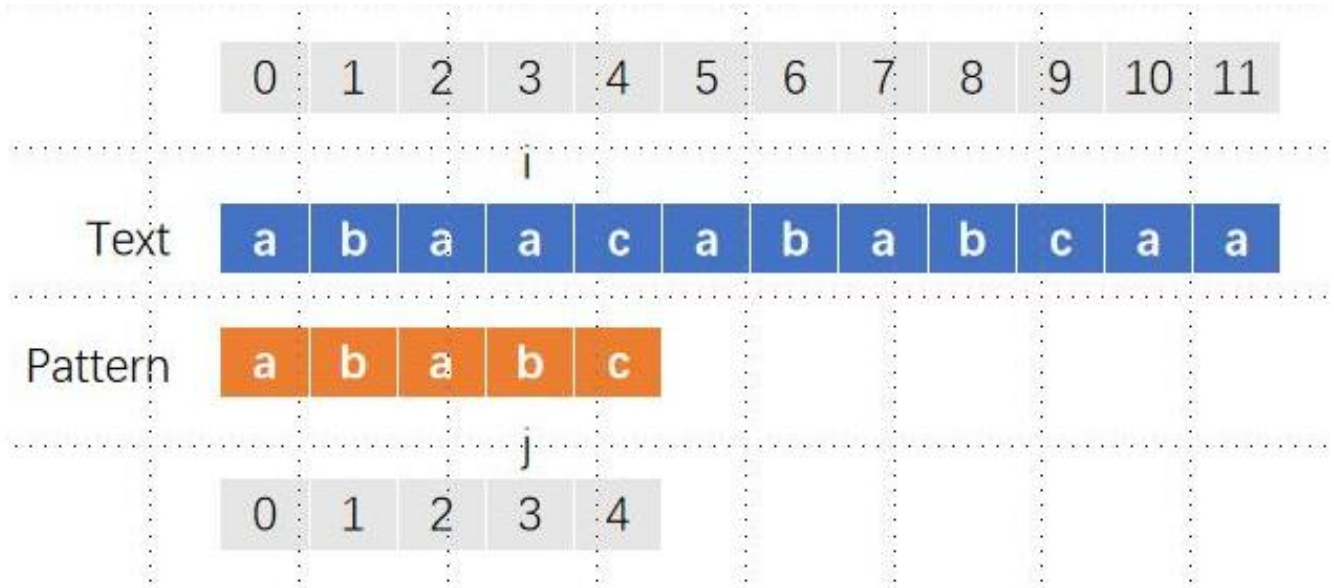
- 如何产生“部分匹配表”?
- "部分匹配值"就是"前缀"和"后缀"的最长的共有元素的长度
- 以“ABCDABD”为例:
  - "A"的前缀和后缀都为空集，共有元素的长度为0;
  - "AB"的前缀为[A]，后缀为[B]，共有元素的长度为0;
  - "ABC"的前缀为[A, AB]，后缀为[BC, C]，共有元素的长度0;
  - "ABCD"的前缀为[A, AB, ABC]，后缀为[BCD, CD, D]，共有元素的长度为0;

Knuth-Morris-Pratt算法

- 如何产生“部分匹配表”？ Next数组
- "部分匹配值"就是"前缀"和"后缀"的最长的共有元素的长度
- 以“ABCDABD”为例：
  - "ABCDA"的前缀为[A, AB, ABC, ABCD]， 后缀为[BCDA, CDA, DA, A]， 共有元素为"A"， 长度为1；
  - "ABCDAB"的前缀为[A, AB, ABC, ABCD, ABCDA]， 后缀为[BCDAB, CDAB, DAB, AB, B]， 共有元素为"AB"， 长度为2；
  - "ABCDABD"的前缀为[A, AB, ABC, ABCD, ABCDA, ABCDAB]， 后缀为[BCDABD, CDABD, DABD, ABD, BD, D]， 共有元素的长度为0。

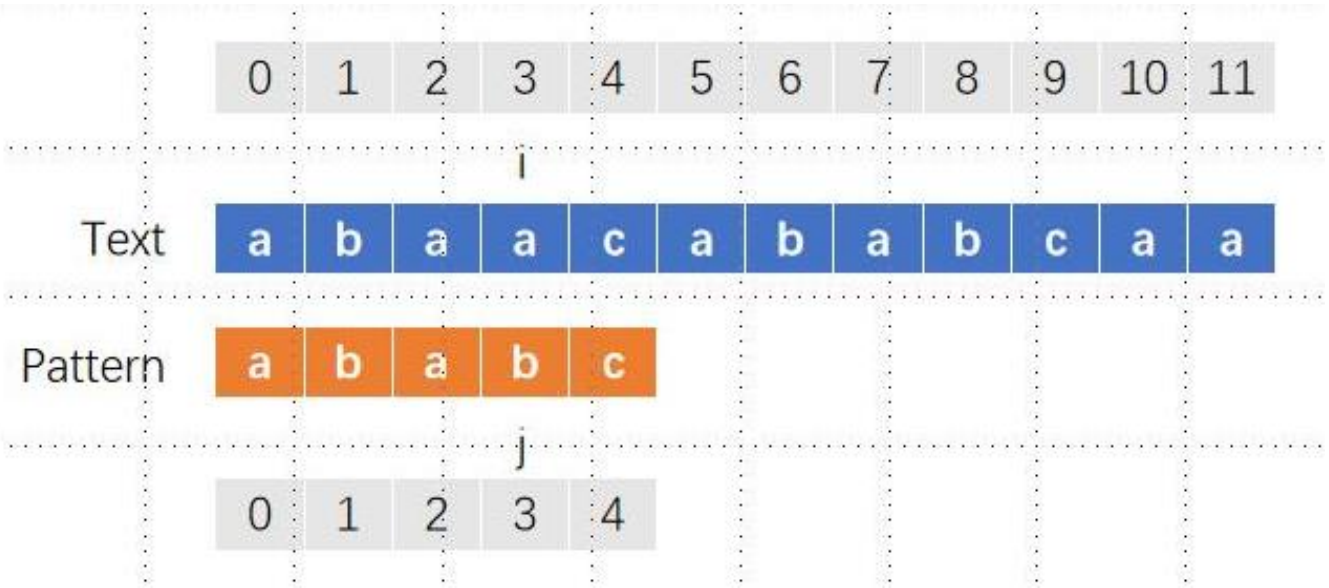
练习

- 下列模式串的next数组是什么?
- 如何匹配?



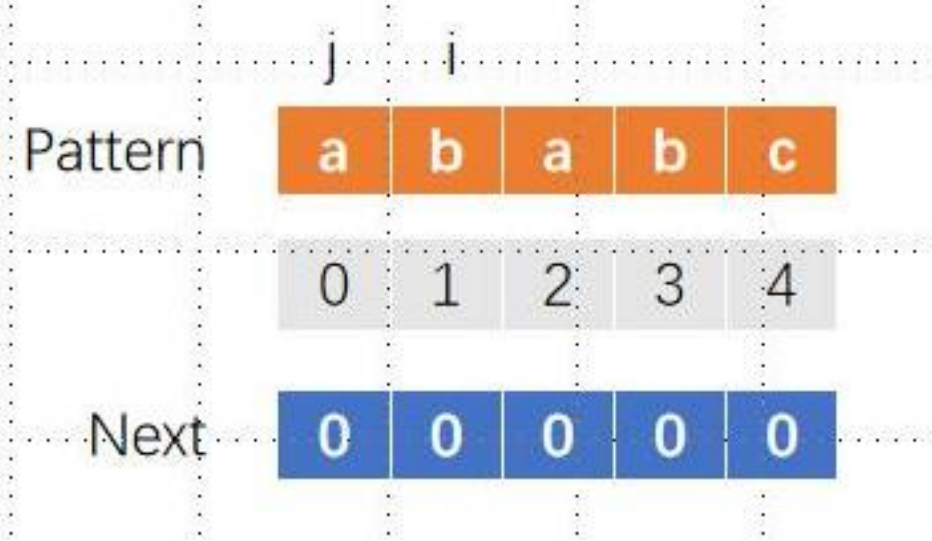
练习

- 下列模式串的next数组是什么？[0, 0, 1, 2, 0]
- 如何匹配？移动位数 = 已匹配的字符数 - 对应的next值



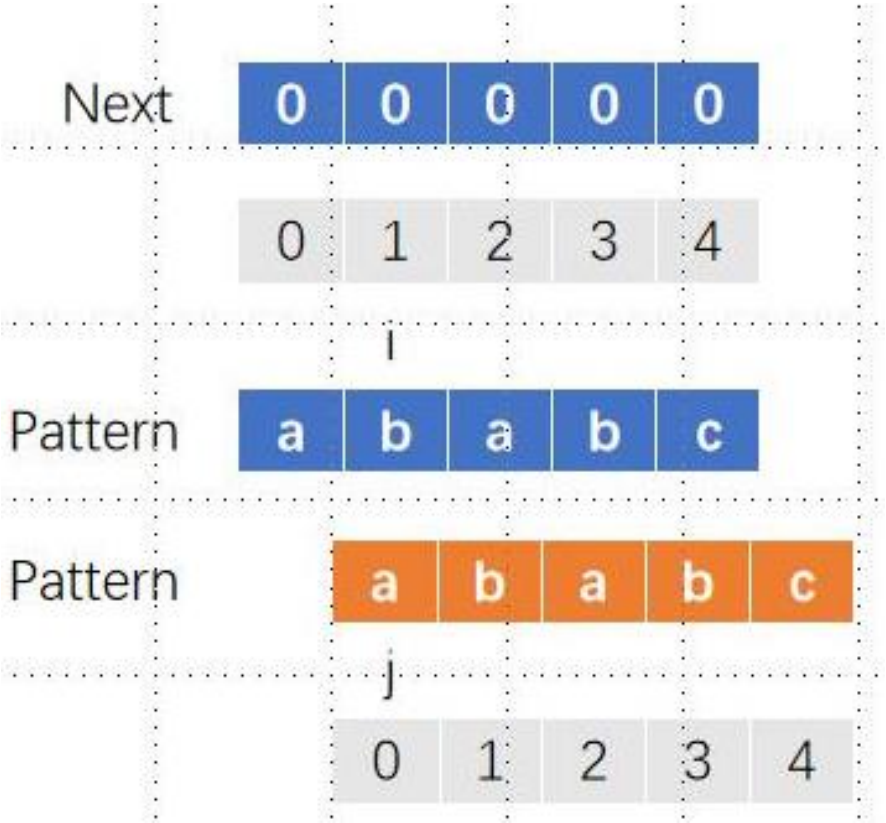
如何计算模式串的next数组？

- 初始化：长度与模式串相等的全0的next数组
- 设定 2 个指针 i 和 j，j 指向位置 0，i 从位置 1



如何计算模式串的next数组？

- 依然看作是 2 个字符串的比较，j 指向的是模式字符串的前缀，而 i 指向的是模式字符串的后缀

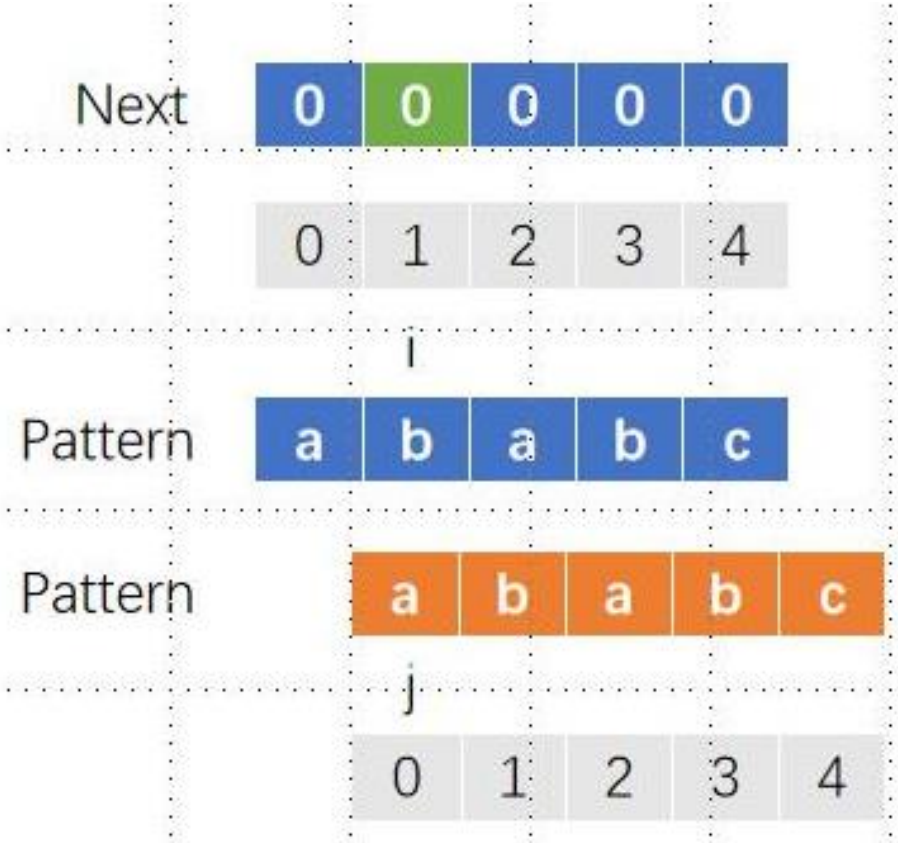


如何计算模式串的next数组？

- 和字符串匹配一样，执行同样的处理：
  1. 当 i 和 j 匹配时，i 和 j 同时右移一格
  2. 当 i 和 j 不匹配时，如果 j 不在字符串开头（位置 0），就回退到上一个能匹配到的位置
  3. 当 i 和 j 不匹配时，如果 j 在字符串开头（位置 0），那么 i 就右移一格

如何计算模式串的next数组？

- 对 next [1] 赋值：i 和 j 不匹配，同时 j 是字符串开头，赋值 0



如何计算模式串的next数组？

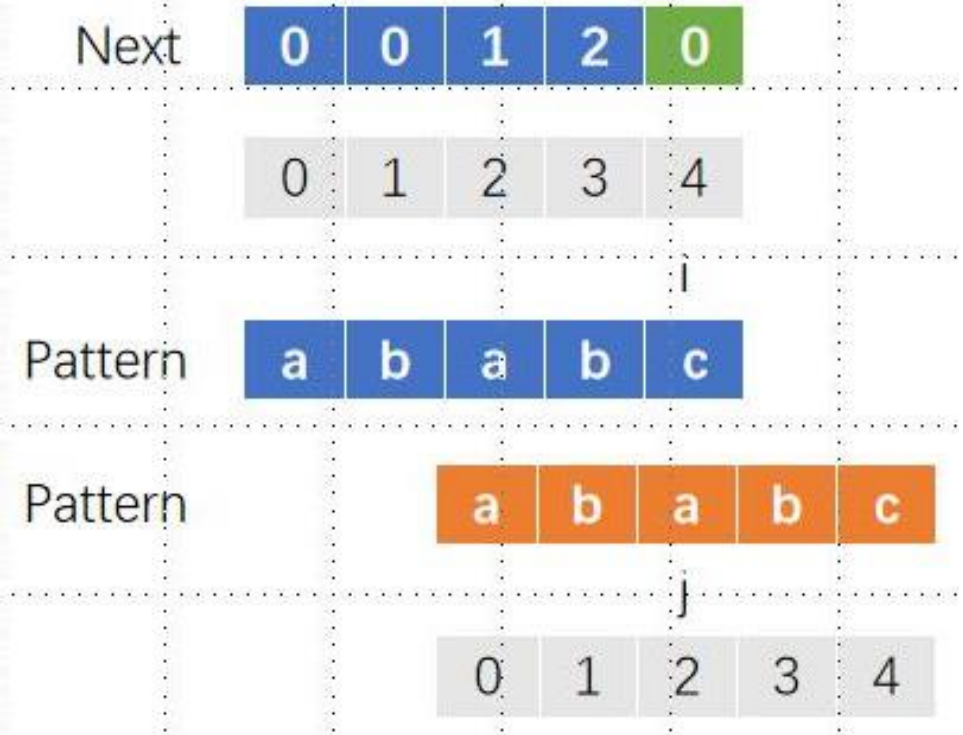
- 对 next [2] 赋值，i 和 j 匹配，此时 j 为 0，代表只有 1 个字符匹配 (j+1)，赋值 1

如何计算模式串的next数组？

- 对 next [3] 赋值，i 和 j 匹配，此时 j 为 1，代表有 2 个字符匹配 (j+1)，赋值 2

如何计算模式串的next数组？

- 对 next [4] 赋值，i 和 j 不匹配，此时 j 为 2，可以得知 j 前面的字符是 ab，而 ab 的最长公共前后缀长度就是 next[1]=0，所以 j 回退到位置 0，用代码表示就是 j=next[j-1]



如何计算模式串的next数组？

- 此时 i 和 j 仍然不匹配，但是 j 已为 0，无法继续回退，所以直接对 next [4] 赋值为 0