

# Lecture 14: KMP算法

## 14.1 KMP 算法基础实现

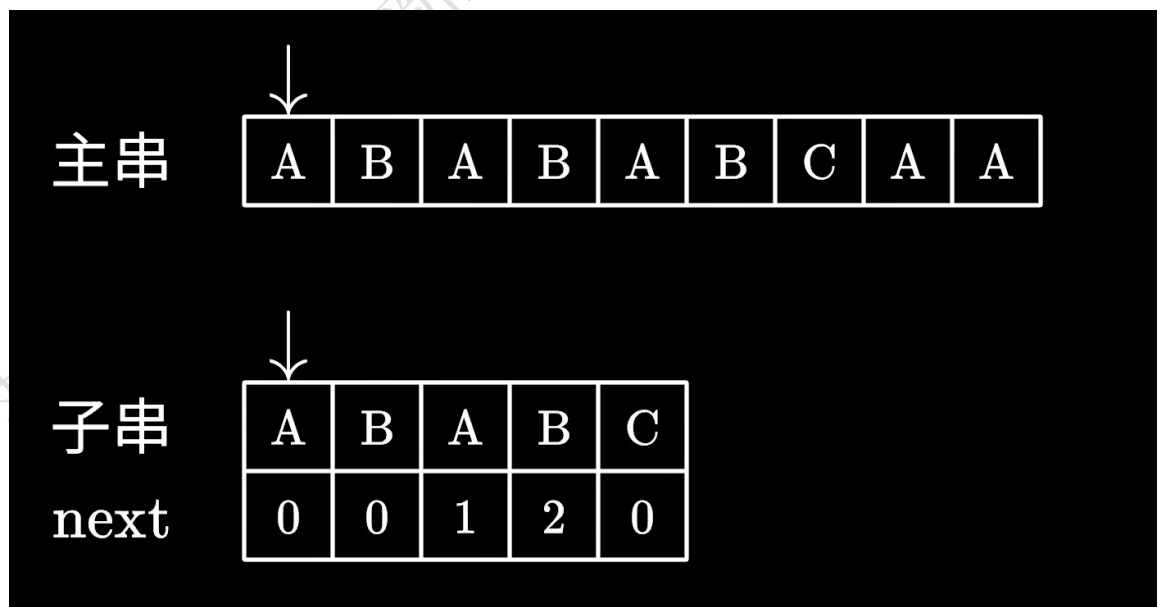
### 14.1.1 基本概念

引入：朴素模式匹配算法的缺点

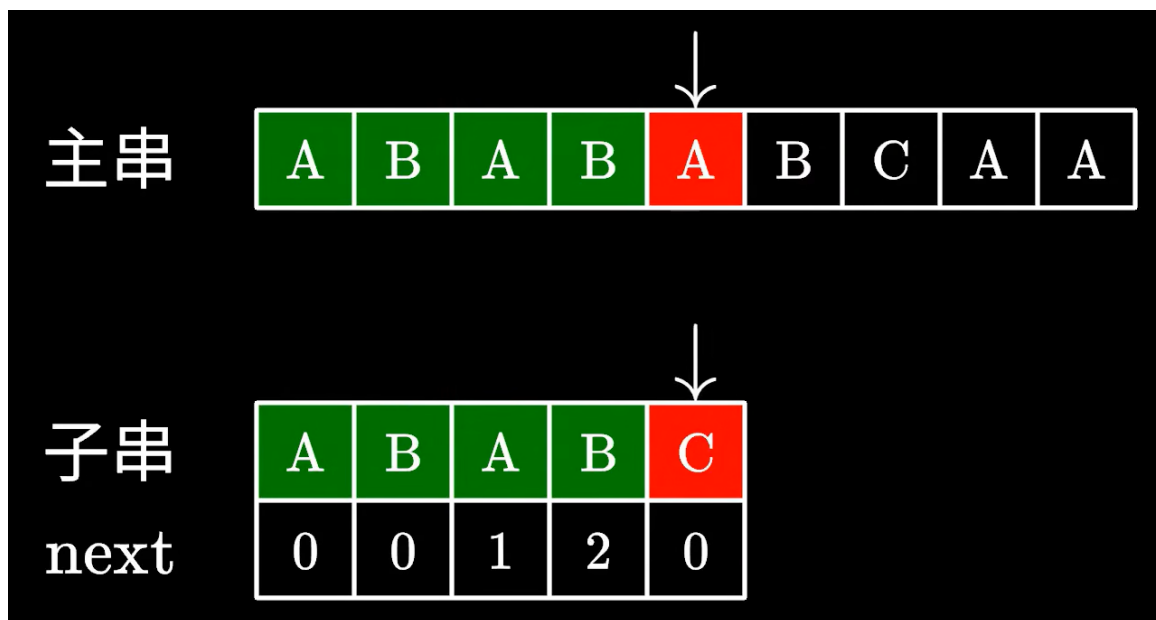
- 引入：
  - 不匹配的字符之前，一定是和模式串一致的；
  - 当某些字串与模式串能部分匹配时，主串的扫描指针  $i$  经常回溯，导致时间开销增加。
- 改进：
  - 利用好模式串本身自带的一些信息，在模式串部分适配的时候，利用好这些信息；

概念：KMP 算法

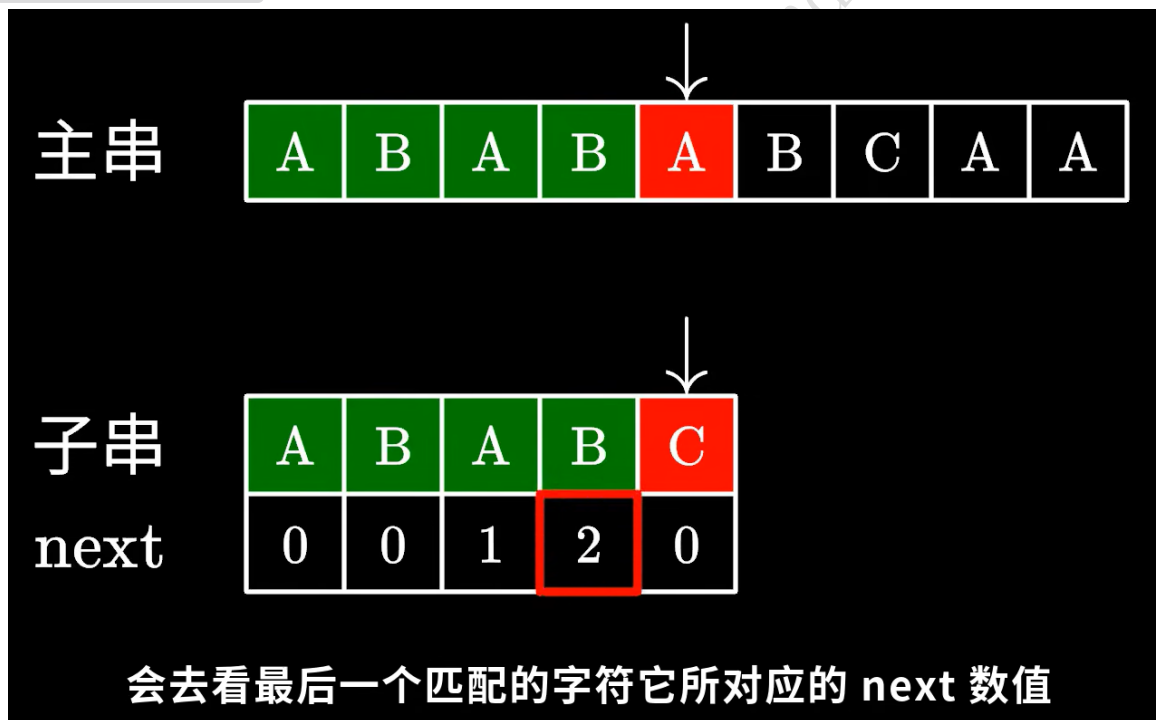
- 概念：
  - KMP 算法，主串指针不回溯，只有模式串指针回溯；
  - 主串很长，模式串一般都很短，所以这样性能更好；
- 内容：
  - 0. 预处理：根据当前模式串，求出当前模式串所使用的 next 数组；
  - 1. 开始运行：最开始时， $j=0, i++, j++$
  - 2. 使用指针  $i$  指向主串  $S$ ，使用指针  $j$  指向模式串  $T$ ；



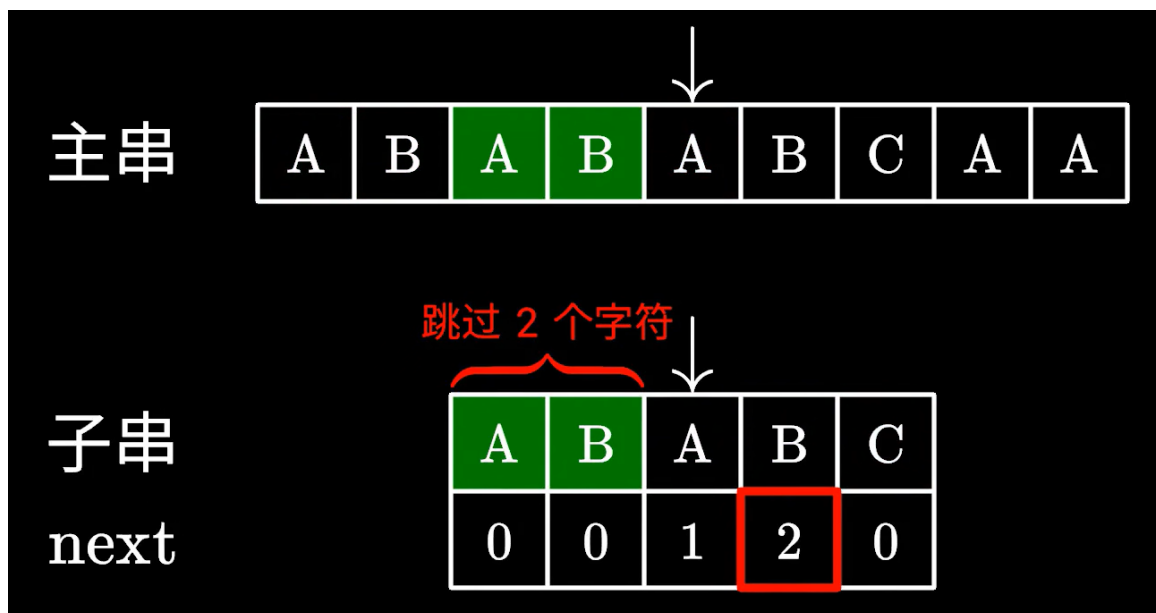
- 3.1 如果相同，同时移动  $i$ 、 $j$  指针往后移
- 3.2 如果  $i$  和  $j$  不匹配的时候，移动字串；



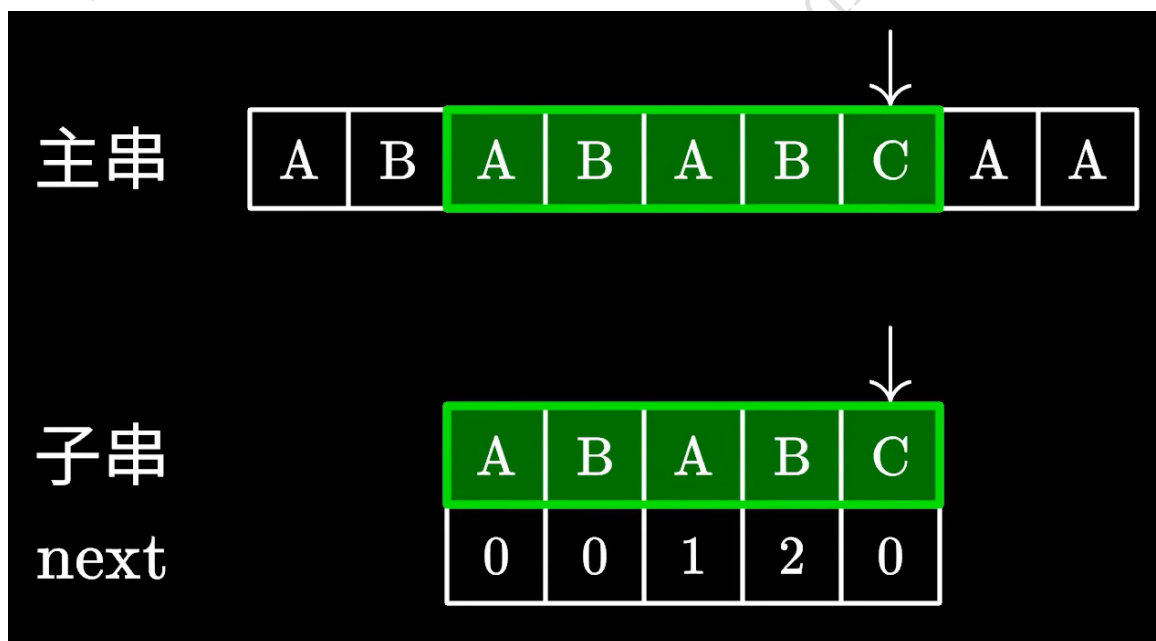
- 4. 进行下一此匹配时, i 指针的位置不变, j 指针的位置根据 next 数组得到: `if (S[i] != T[j]) j=next[j];`



- 这里的 2 代表子串中我们可以“跳过匹配”的字符个数



- 5. 然后从此位置开始继续匹配，如果  $j$  指向了字串的末尾，则表示模式串  $T$  当中内容全部匹配；



- next 数组：
  - 概念：
    - 使用 next 数组计算回溯的模式串指针：
  - 举例：对于模式串  $T=ababc$ 
    - 当第 6 个元素匹配失败时，可令主串指针  $i$  不变，模式串指针  $j=3$ ；
    - 当第 5 个元素匹配失败时，可令主串指针  $i$  不变，模式串指针  $j=2$ ；
    - 当第 4 个元素匹配失败时，可令主串指针  $i$  不变，模式串指针  $j=2$ ；
    - 当第 3 个元素匹配失败时，可令主串指针  $i$  不变，模式串指针  $j=1$ ；
    - 当第 2 个元素匹配失败时，可令主串指针  $i$  不变，模式串指针  $j=1$ ；
    - 当第 1 个元素匹配失败时，直接匹配下一个相邻子串，令  $j=0$ ， $i++$ ， $j++$ ；
    - 将以上的数据存储成 next 数组：

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

- 特征：
  - next 数组的作用：修改 j 的值为多少；
  - next 数组只和短的模式串有关，和长长的主串无关；

### 14.1.2 算法实现

代码：KMP 代码实现

```
//KMP算法
int Index_KMP(SSString S,SSString T){
    int i=1,j=1;
    int next[T.length+1];
    get_next(T,next); //求模式串的next数组
    while(i<=S.length && j<=T.length ){
        //注意j==0的情况，此时是直接往后
        if(j==0||S.ch[i]==T.ch[j]){
            ++i;
            ++j; //继续比较后继字符
        }
        else //不匹配时，在next数组中取出j的值
        {
            j=next[j]; //模式串向后移动
        }
    }
    if(j>T.length )
        return i-T.length ; //匹配成功
    else
        return 0;
}
```

算法性能

- KMP 算法平均时间复杂度： $O(n + m)$ 
  - 求 next 数组的部分： $O(m)$
  - 模式匹配过程最坏时间复杂度： $O(n)$
- 注意：主要用来解决子串与模式串的匹配问题；
- 注意：数组传参只需要传数组名即可，不用加地址符号；

### 14.1.3 求 next 数组

概念：求 next 数组

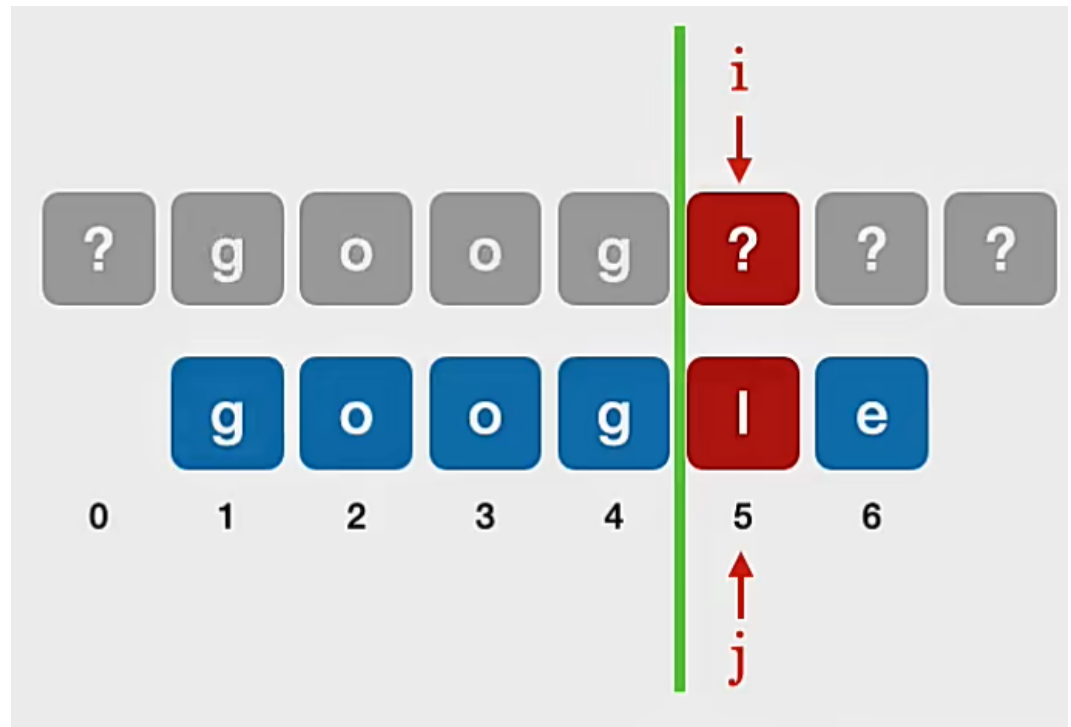
- next 数组的作用：

- 当模式串的第  $j$  个字符匹配失败时，令模式串跳到 `next[j]` 再继续匹配；这样主串的指针不需要回溯；
- 基本概念：
  - 串的前缀：包含第一个字符，且**不包含**最后一个字符的字串；
  - 串的后缀：包含最后一个字符，且**不包含**第一个字符的字串；
- 方法：
  - 求 `next` 数组的本质，其实就是寻找子串中相同前后缀的长度，并且一定是最长的前后缀；
    - 末尾的 ABA 的最长相同前缀是 ABA，因此 `next` 指向前缀 ABA 的下一个位置，即 3；

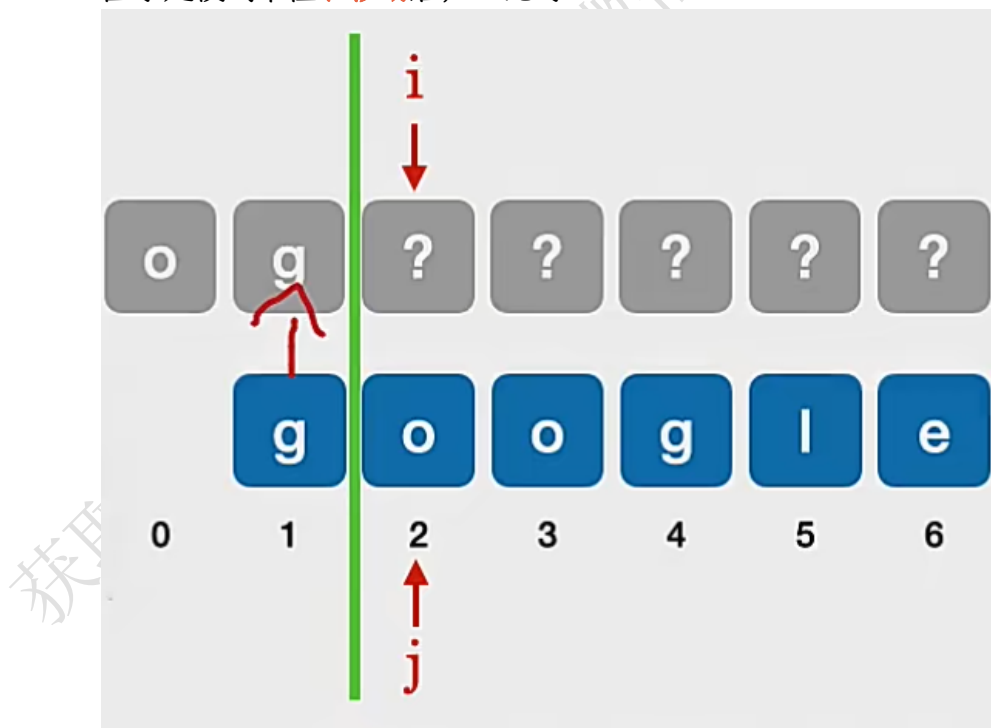
|      |   |   |   |   |   |   |   |   |
|------|---|---|---|---|---|---|---|---|
| 子串   | A | B | A | C | A | B | A | B |
| next | 0 | 0 | 1 | 0 | 1 | 2 | 3 | 2 |

- 任何模式串都一样，第 1 个字符不匹配时，只能匹配下一个子串，因此，`next[1]` 都无脑写 0；
- 任何模式串都一样，第 2 个字符不匹配时，应尝试匹配模式串的第 1 个字符因此，`next[2]` 都无脑写 1；
- 不匹配时的计算方法：
  - 1. 在不匹配的位置前边，划一根分界线；如图

- 如图是第五个位置：



- 2. 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；
- 图示是模式串往**右移动**后，匹配时：



- 3. 此时  $j$  指向哪儿， $next$  数组值就是多少；
- 当第  $j$  个字符匹配失败，有前  $1 \sim j-1$  个字符组成的串记为  $S$ ，则：
- $next[j]=S$  的最长相等前后缀长度+1
- 特别的， $next[1]=0$

## 方法示例

求 next 数组方法.

① 画图

|      |   |   |   |   |   |   |   |   |   |   |   |
|------|---|---|---|---|---|---|---|---|---|---|---|
| s    | a | b | a | b | a | a | b | a | b | a | a |
| next | 0 | 1 | 1 | 2 | 3 |   |   |   |   |   |   |

直接写.      后续计算出来后, 直接填.

② 求 next

j=3 :

|   |   |   |   |
|---|---|---|---|
| a | b |   | x |
| a | b | a |   |
| x | a | b | a |
|   | a | b | a |

↑ j=1

j=4 :

|   |   |   |   |
|---|---|---|---|
| a | b | a | x |
| a | b | a | b |
| a | b |   | a |
| a | b | b | a |
| a | b | a | b |

↑ j=2

j=5 :

|   |   |   |   |   |
|---|---|---|---|---|
| a | b | a | b | x |
| a | b | a | b | a |
| a | b | a |   | b |
| a | b | a | b | a |
| a | b | a | a | b |

↑ j=3

后续同理.

## 代码实现

```
//求模式串T的next数组
void get_next(SString T, int next[]) {
    int i=1, j=0;
    next[1]=0;
    while(i<T.length){
        if(j==0 || T.ch[i]==T.ch[j]){
            ++i;
            ++j;
            //若pi=pj, 则next[j+1]=next[j]+1
            next[i]=j;
        }
        else
            //否则令j=next[j], 循环继续
    }
}
```

```
j=next[j];  
}  
}
```

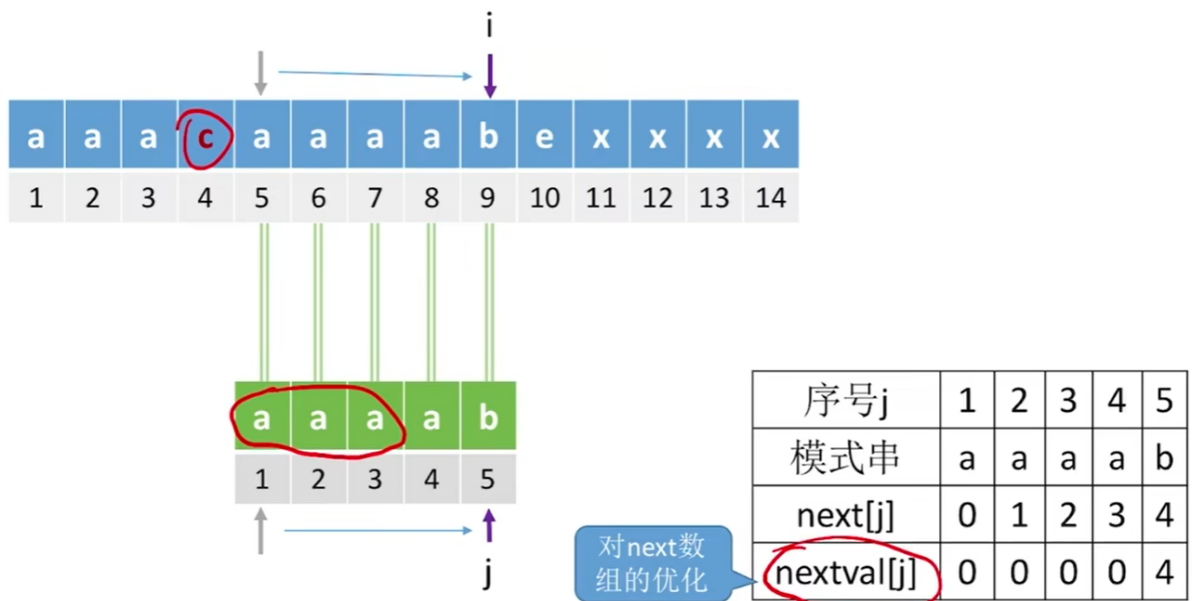
## 14.2 KMP 算法优化实现

**概念：** KMP 算法优化实现

- 概念：优化思路
  - 判断 next 数组所指向的字符，和原本适配的字符是否相等；
  - 如果不相等，则保存原本 next 中的值不变；
  - 如果当前指向的字符和原本 next 数组中某位置的值相等，则此时对 next 数组进行优化；可以另 next 的数值直接变为这一相等位置的值；
- 优化图示：



- 图示：



### 求 nextval 的方法

- 先求 next 数组，再由 next 数组求 nextval 数组
- 1. nextval[1] 的值直接写 0；
- 2. 第 2 个字符如果和第 1 个字符不同，则写 1；
- 3. 如果第 3 个字符和第 1 个字符相同，则写第 1 个字符的 next 数值 0；
- 4. 如果第 4 个字符的位置和第 2 个字符相同，则写第 2 个字符的 next 数值 1；

|            |   |   |   |   |   |   |
|------------|---|---|---|---|---|---|
| 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 |
| nextval[j] | 0 | 1 | 0 | 1 | 0 | 4 |

- 图示：

### 代码实现

```
//nextval数组的求法：
//先算出next数组
//先令nextval[1]=0
for(int j=2;j<=T.length;j++){
    if(T.ch[next[j]]==T.ch[j])
        nextval[j] = nextval[next[j]]; //相等就赋值，避免重复匹配
    else
```

```
nextval[j] = next[j];  
}
```

获取笔记持续更新可加微信: WandD2277