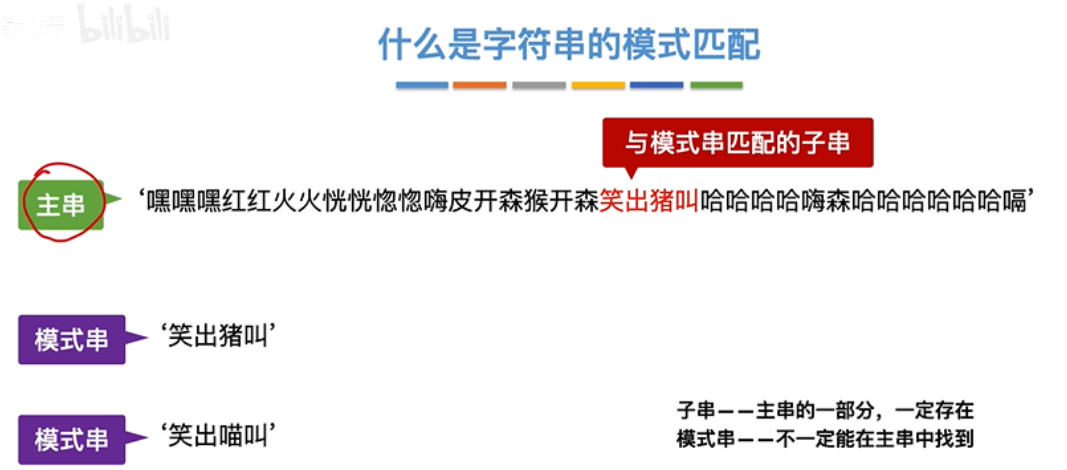
朴素模式匹配算法和KMP算法

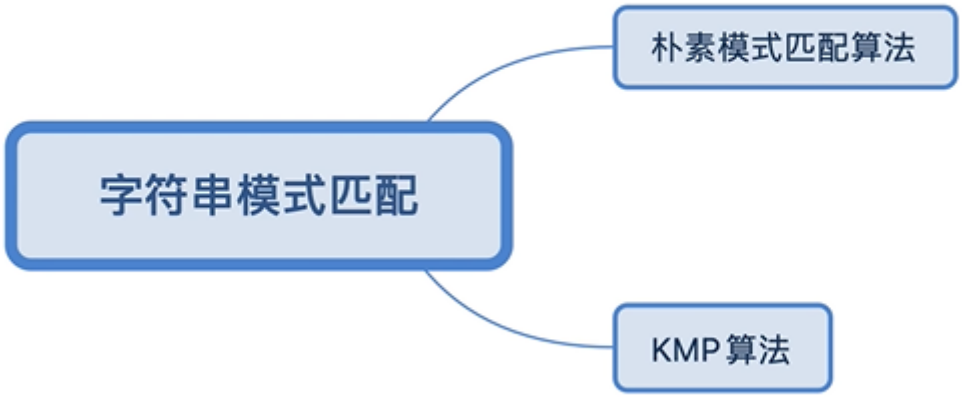
一、朴素模式匹配算法

- **什么是字符串的模式匹配**



字符串模式匹配：在主串中找到与模式串相同的子串，并返回其所在位置。

- **两种模式匹配算法**



- **朴素模式匹配算法**

朴素模式匹配算法

主串S:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a	b	a	a	b	a	a	b	c	a	b	a	a	b	c

模式串T:

1	2	3	4	5	6
a	b	a	a	b	c

主串长度为n，模式串长度为 m

朴素模式匹配算法：将主串中所有长度为m的子串依次与模式串对比，直到找到一个完全匹配的子串，或所有的子串都不匹配为止。

朴素模式匹配算法

Index(S,T)：定位操作。若主串S中存在与串T值相同的子串，则返回它在主串S中第一次出现的位置；否则函数值为0。

```
int Index(SString S, SString T){
    int i=1, n=StrLength(S), m=StrLength(T);
    SString sub;    //用于暂存子串
    while(i<=n-m+1){
        SubString(sub,S,i,m);
        if(StrCompare(sub, T)!=0)  ++i;
        else return i;  //返回子串在主串中的位置
    }
    return 0;   //S中不存在与T相等的子串
}
```

最多对比 n-m+1 个子串

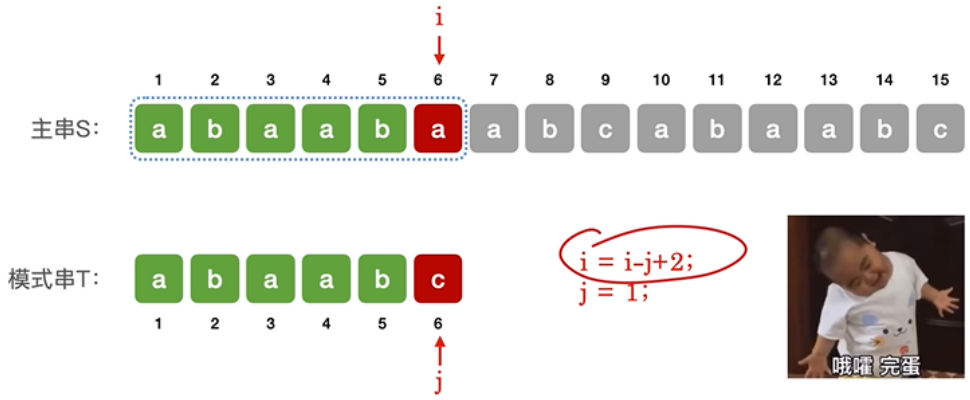
取出从位置 i 开始，长度为m的子串

子串和模式串对比，若不匹配，则匹配下一个子串

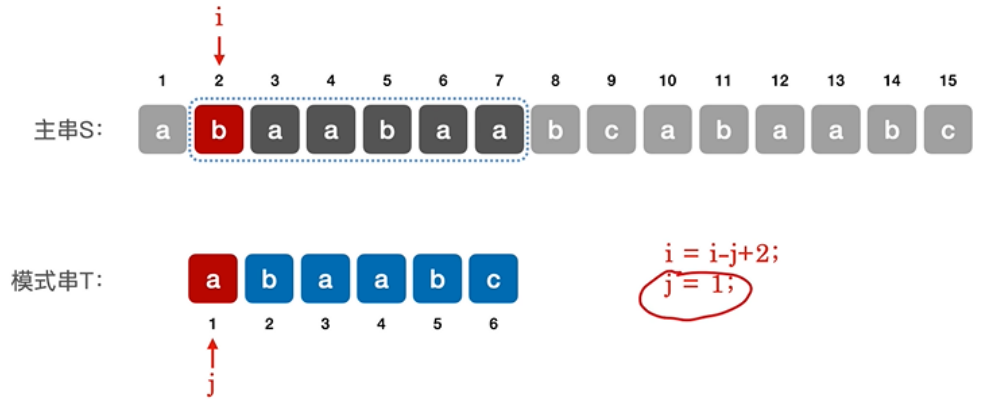
原来是这样啊

接下来：不使用字符串的基本操作，直接通过数组下标实现朴素模式匹配算法

朴素模式匹配算法

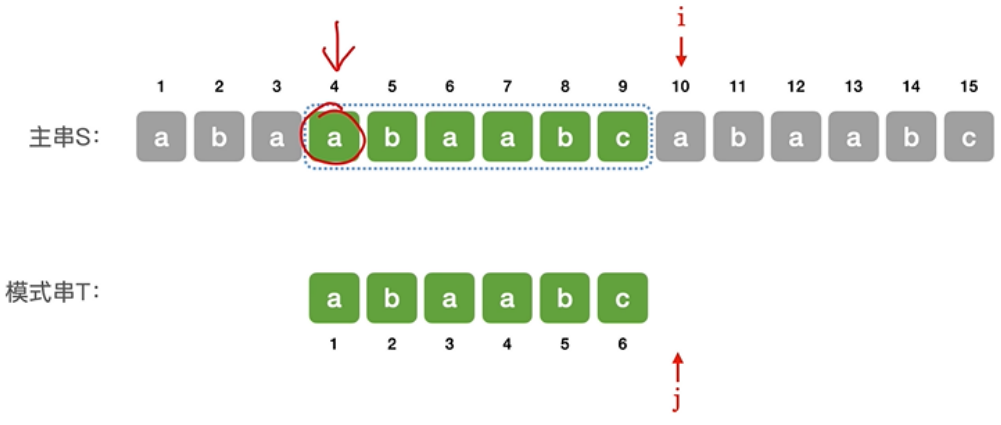


若当前子串匹配失败，则主串指针 i 指向下一个子串的第一个位置，模式串指针 j 回到模式串的第一个位置



若当前子串匹配失败，则主串指针 i 指向下一个子串的第一个位置，模式串指针 j 回到模式串的第一个位置

朴素模式匹配算法



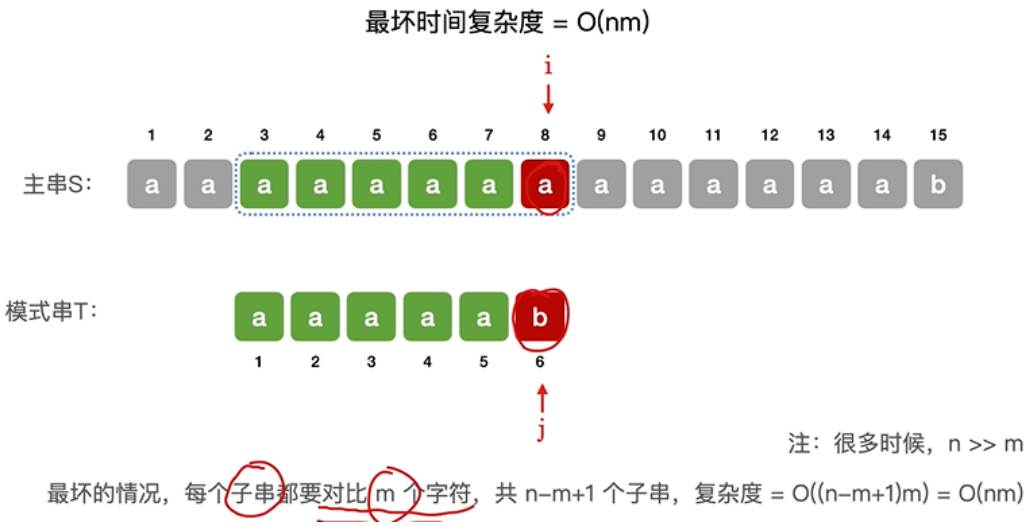
若 j > T.length，则当前子串匹配成功，返回当前子串第一个字符的位置 —— i - T.length

```
int Index(SString S,SString T)
{
    int i = 1,j = 1;
    while(i <= S.length && j <= T.length)
    {
```

```
        if(S.ch[i] == T.ch[j])
        {
            ++i;      //继续比较后继字符
            ++j;
        }
        else
        {
        i = i - j + 2;
        j = 1;    //指针后退重新开始匹配
        }
    }
    if(j > T.length)
        return i - T.length;
    else
        return 0;
}
```

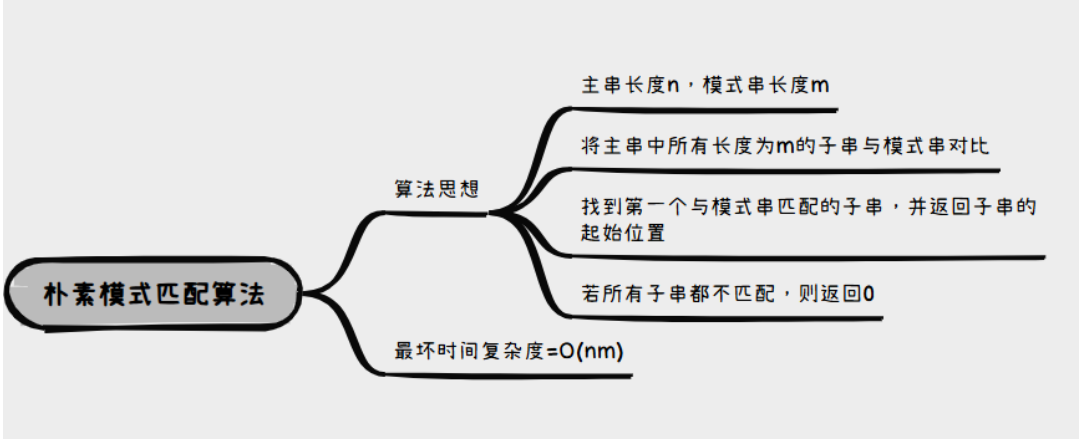
设主串长度为 n，模式串长度为 m，则

最坏时间复杂度 = O(nm)

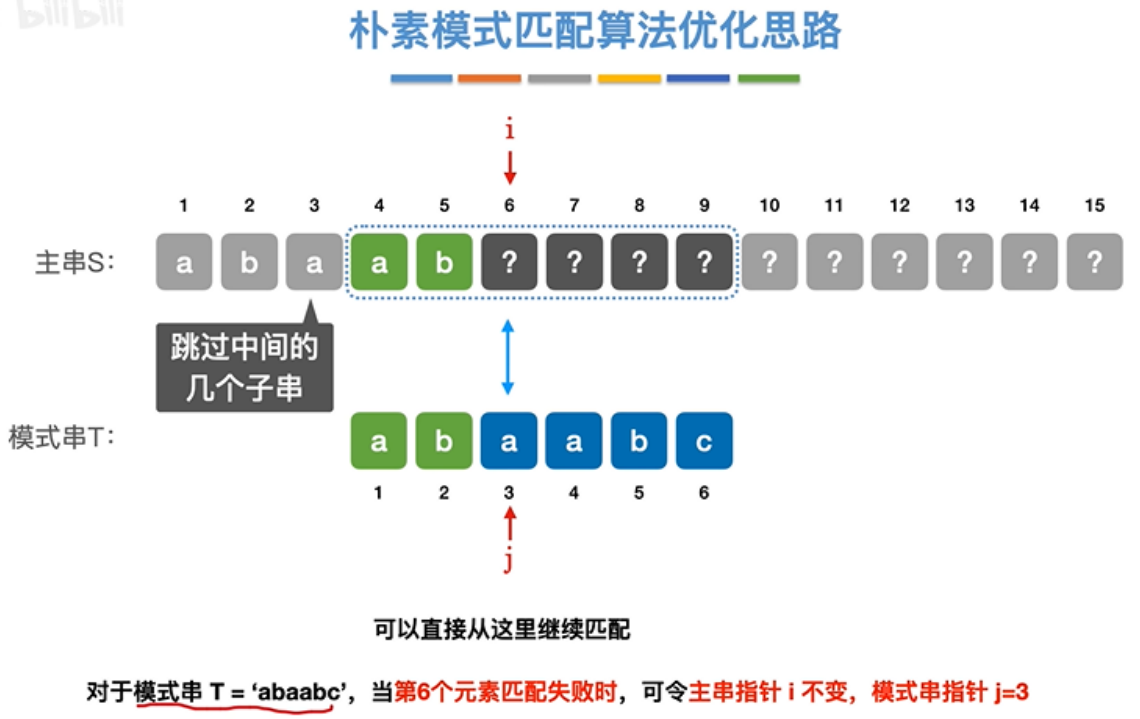


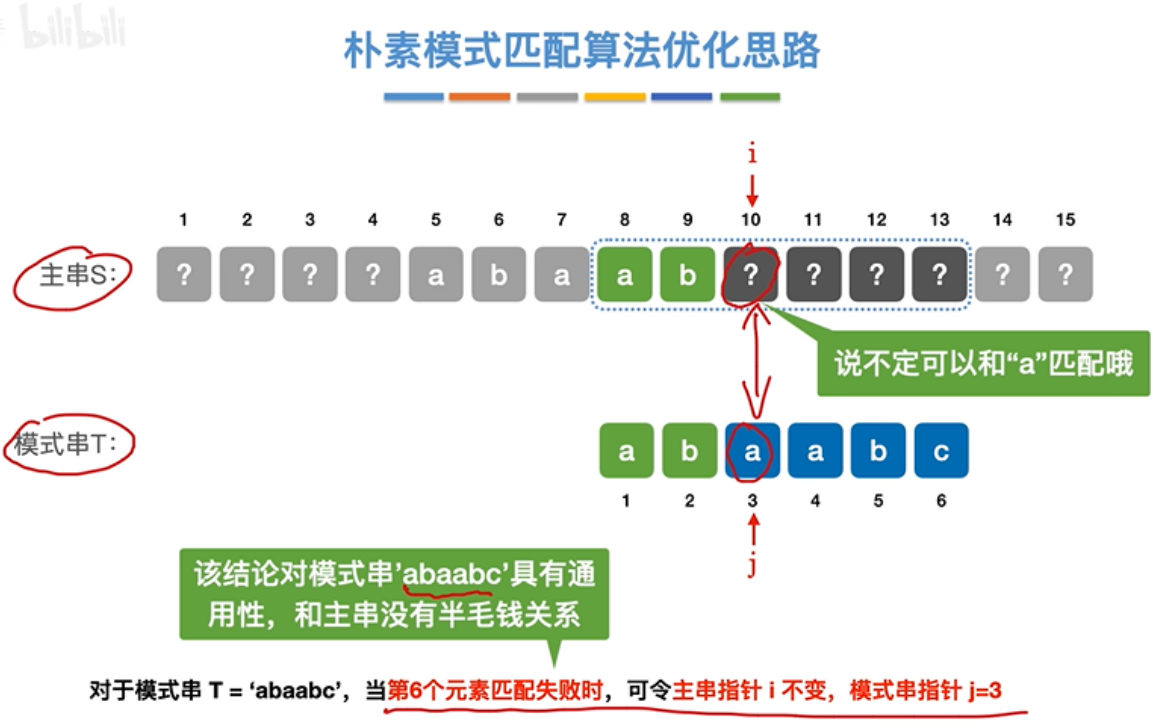
注：很多时候，n >> m

最坏的情况，每个子串都要对比 m 个字符，共 n−m+1 个子串，复杂度 = O((n−m+1)m) = O(nm)

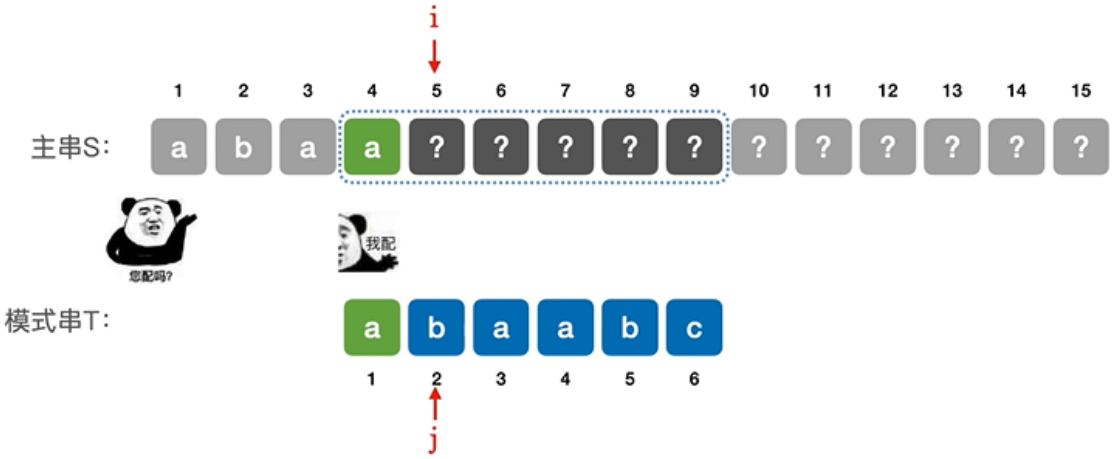


二、KMP算法


朴素模式匹配算法优化思路
i
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
主串S:
a b a a b ? ? ? ? ? ? ? ? ? ?
跳过中间的
几个子串
模式串T:
a b a a b c
1 2 3 4 5 6
j
可以直接从这里继续匹配
对于模式串 T = 'abaabc'，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


朴素模式匹配算法优化思路
i
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
主串S:
? ? ? ? a b a a b ? ? ? ? ? ?
说不定可以和“a”匹配哦
模式串T:
a b a a b c
1 2 3 4 5 6
j
该结论对模式串'abaabc'具有通
用性，和主串没有半毛钱关系
对于模式串 T = 'abaabc'，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

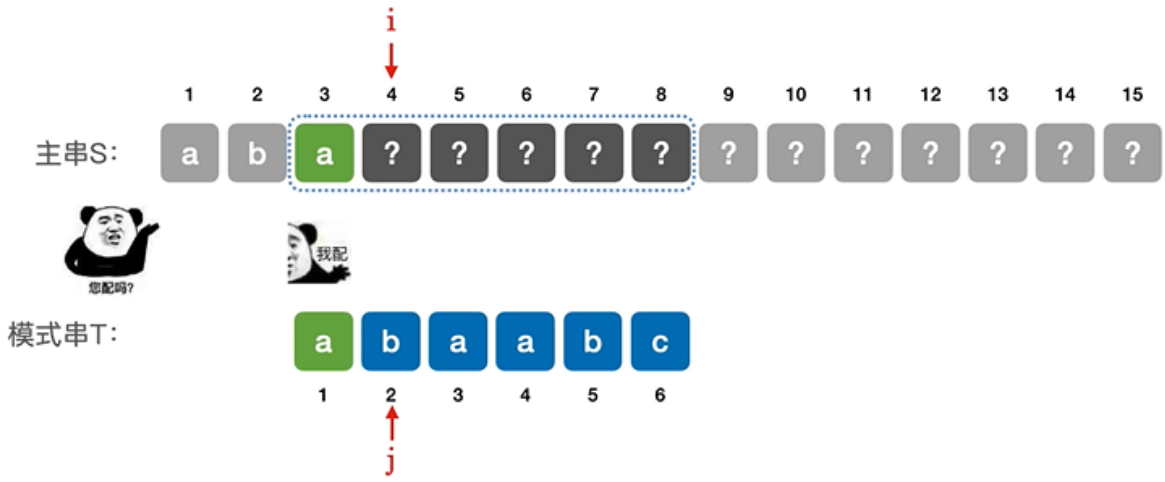
如果其他位置不匹配呢?



对于模式串 T = 'abaabc'，当第5个元素匹配失败时? 怎么搞?

可令主串指针 i 不变，模式串指针 j = 2

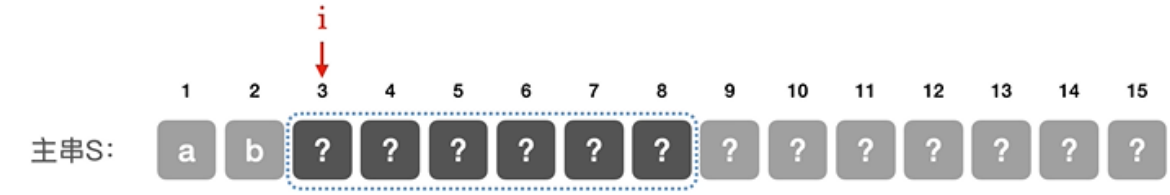
如果其他位置不匹配呢?

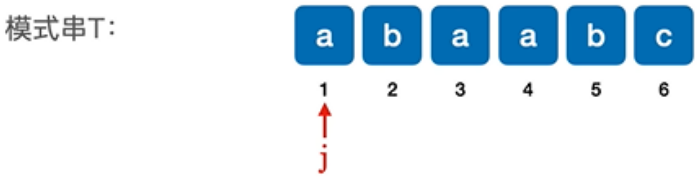


对于模式串 T = 'abaabc'，当第4个元素匹配失败时? 怎么搞?

可令主串指针 i 不变，模式串指针 j = 2

如果其他位置不匹配呢?

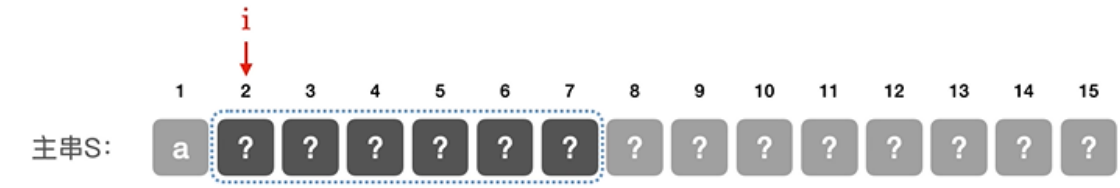


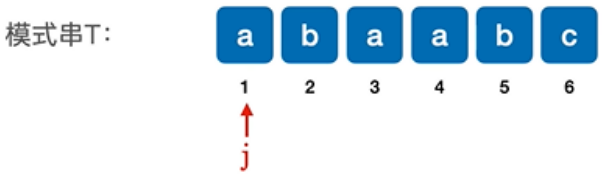


对于模式串 T = 'abaabc'，当第3个元素匹配失败时? 怎么搞?

可令主串指针 i 不变，模式串指针 j = 1

如果其他位置不匹配呢?

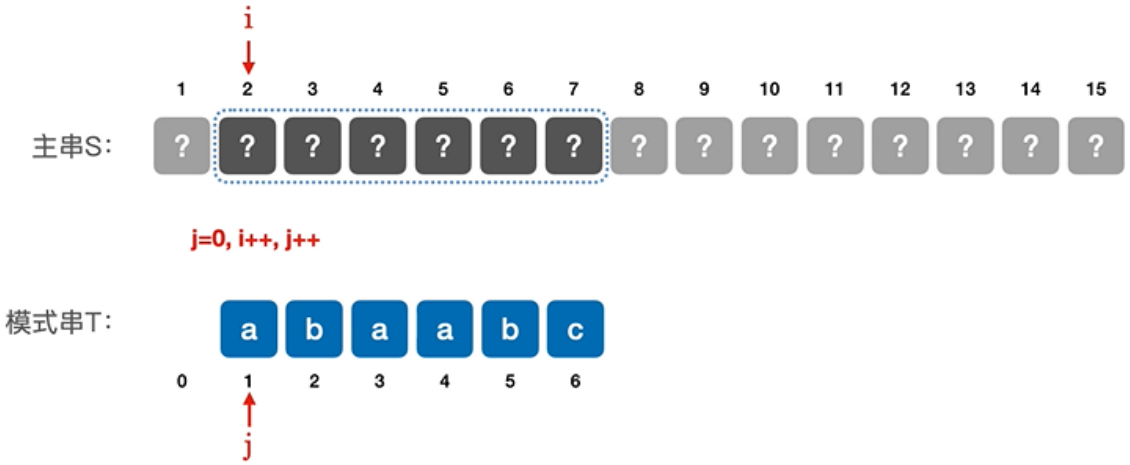




对于模式串 T = 'abaabc'，当第2个元素匹配失败时? 怎么搞?

可令主串指针 i 不变，模式串指针 j = 1

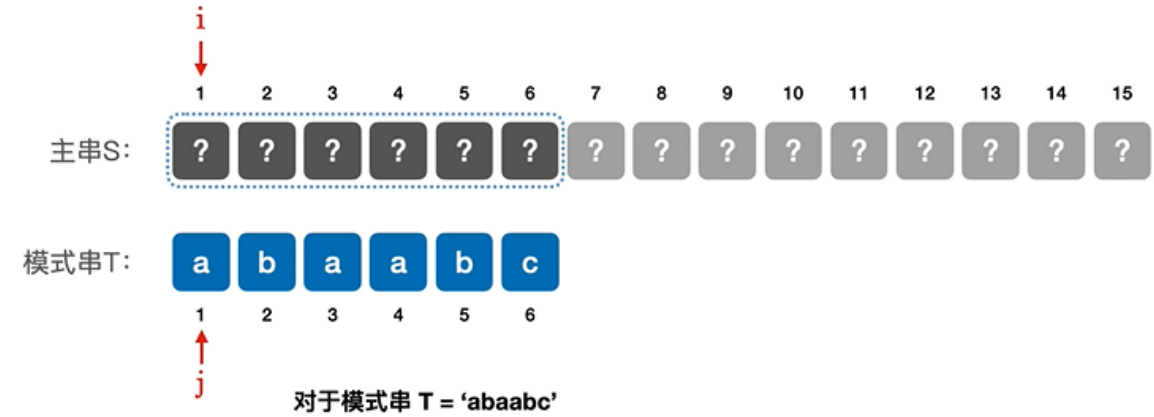
如果其他位置不匹配呢?



对于模式串 T = 'abaabc'，当第1个元素匹配失败时? 怎么搞?

匹配下一个相邻子串

结论



对于模式串 T = 'abaabc'

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

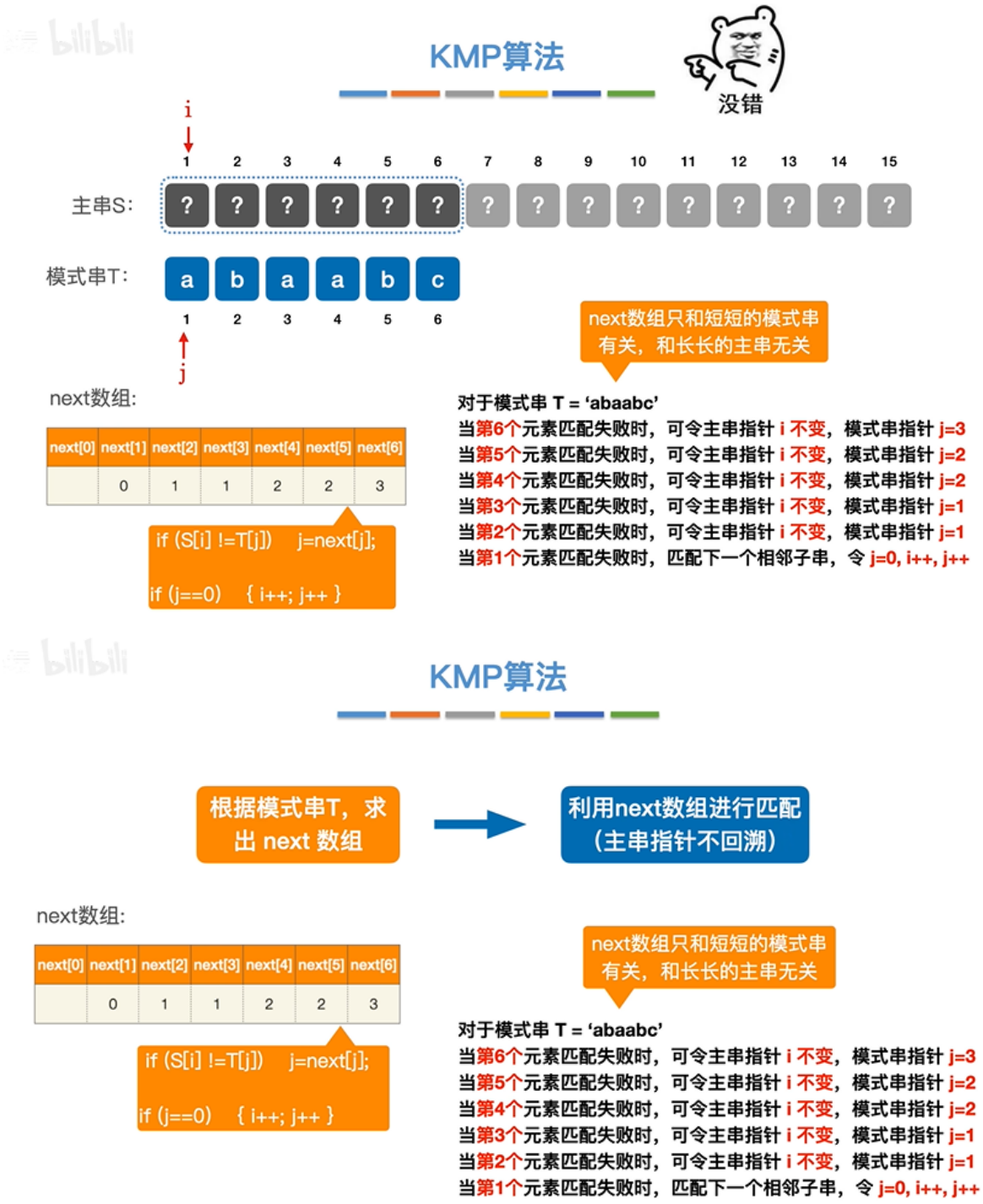
当第5个元素匹配失败时，可令主串指针 i 不变，模式串指针 j=2

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

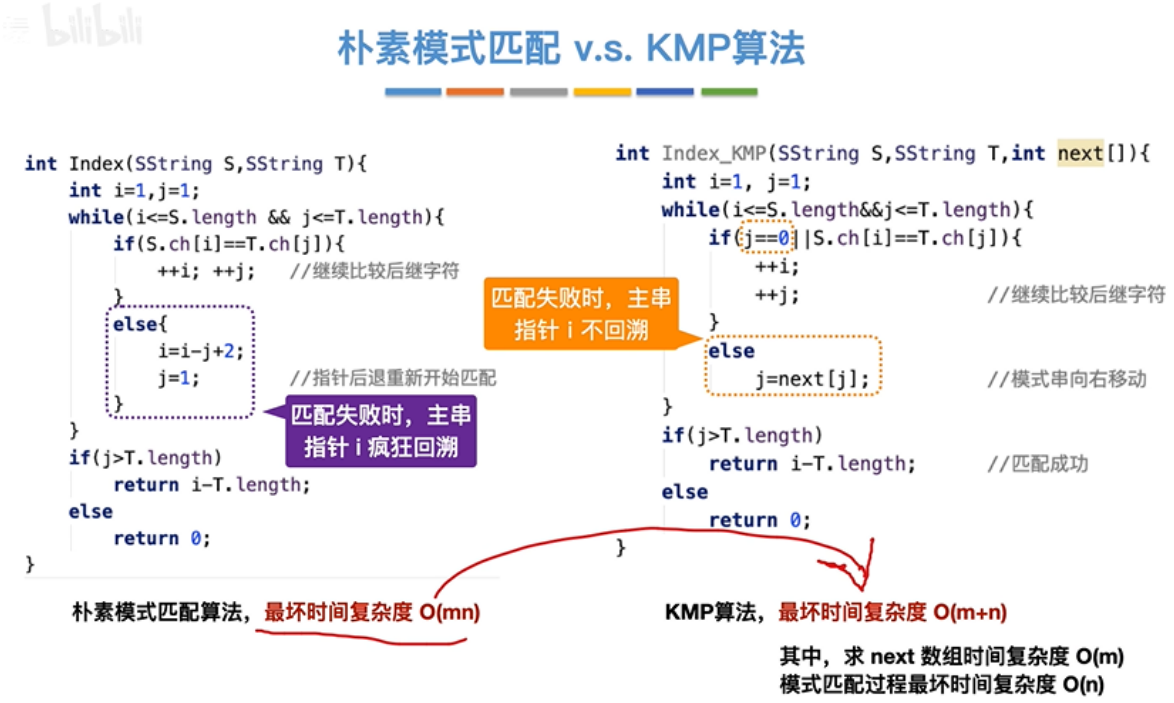
当第2个元素匹配失败时，可令主串指针 i 不变，模式串指针 j=1

当第1个元素匹配失败时，匹配下一个相邻子串，令 j=0, i++, j++

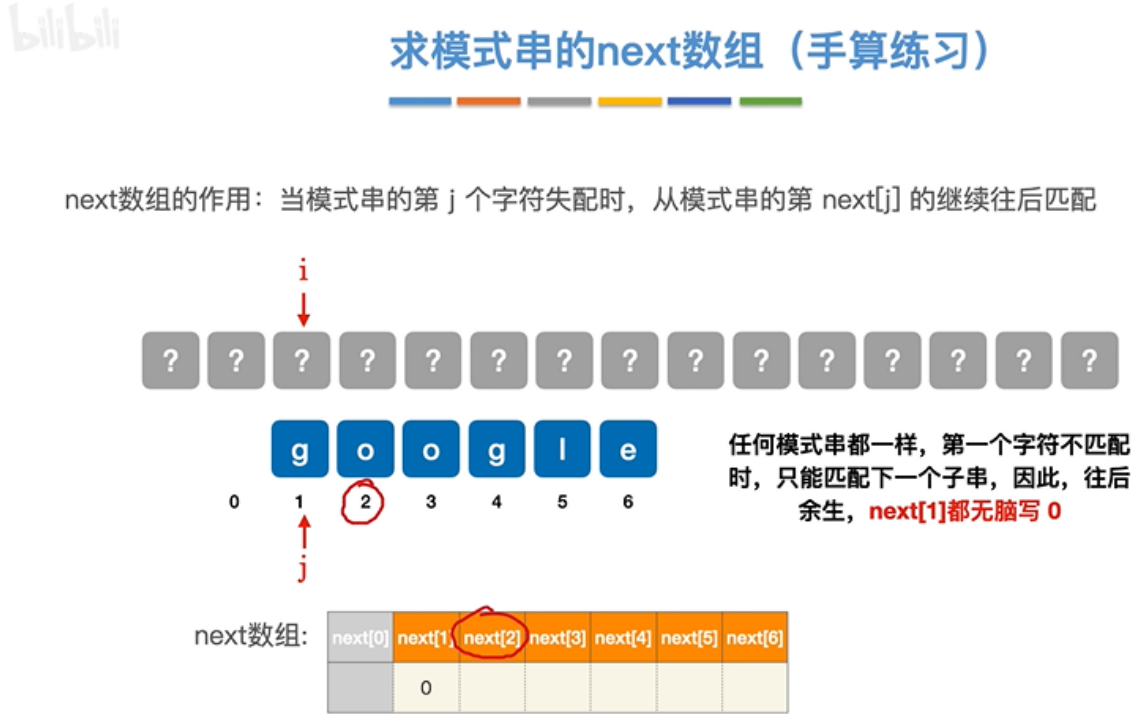



```c
int Index_KMP(SString S,SString T,int next[])
{
    int i = 1,j = 1;
    while(i <= S.length && j <= T.length)
    {
        if(j == 0 || S.ch[i] == T.ch[i])
        {
            ++i;
            ++j;                    //继续比较后继字符
        }
        else
        {
            j = next[j];      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

```
        else
            return 0;
    }
}
```

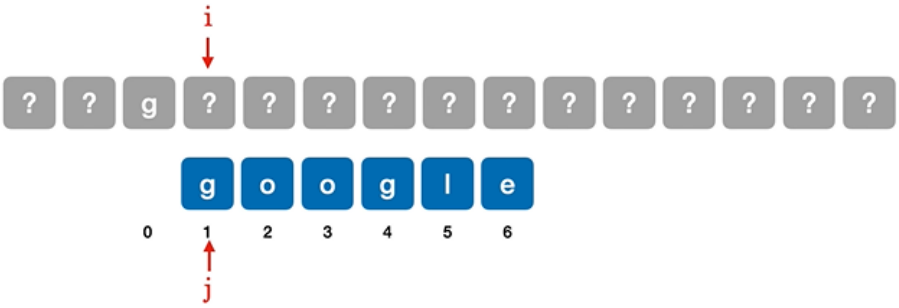


三、KMP算法求next数组



求模式串的next数组（手算练习）

next数组的作用：当模式串的第 j 个字符失配时，从模式串的第 next[j] 的继续往后匹配



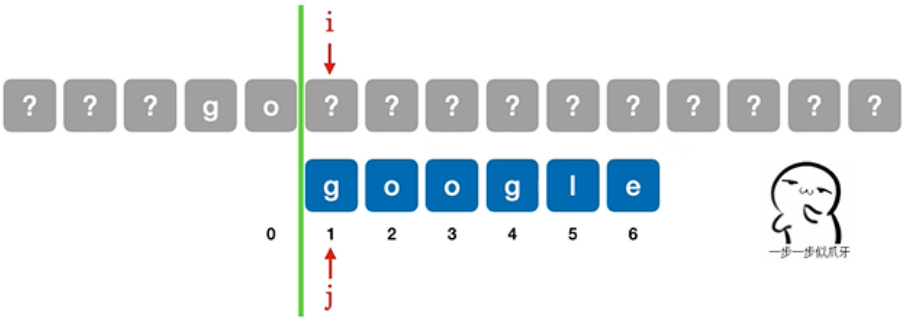
next数组：

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

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，往后余生，next[2]都无脑写 1

求模式串的next数组（手算练习）

next数组的作用：当模式串的第 j 个字符失配时，从模式串的第 next[j] 的继续往后匹配



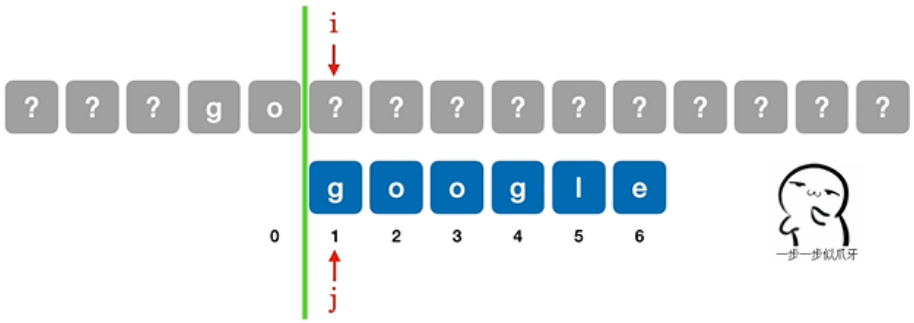
next数组：

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

在不匹配的位置前边，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

求模式串的next数组（手算练习）

next数组的作用：当模式串的第 j 个字符失配时，从模式串的第 next[j] 的继续往后匹配



next数组：

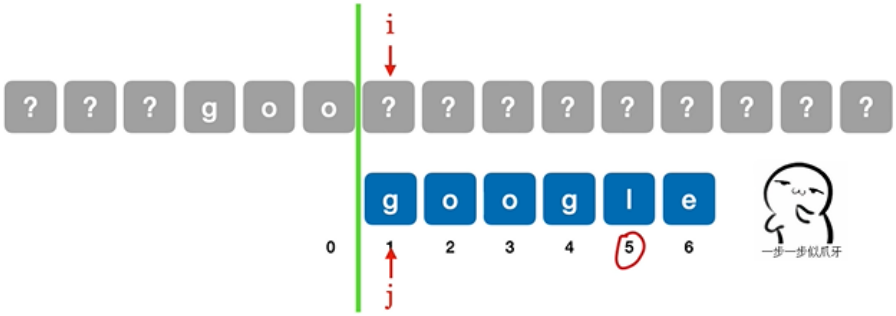
next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
	0	1	1			

在不匹配的位置前边，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

求模式串的next数组（手算练习）

next数组的作用：当模式串的第 j 个字符失配时，从模式串的第 next[j] 的继续往后匹配



next数组：

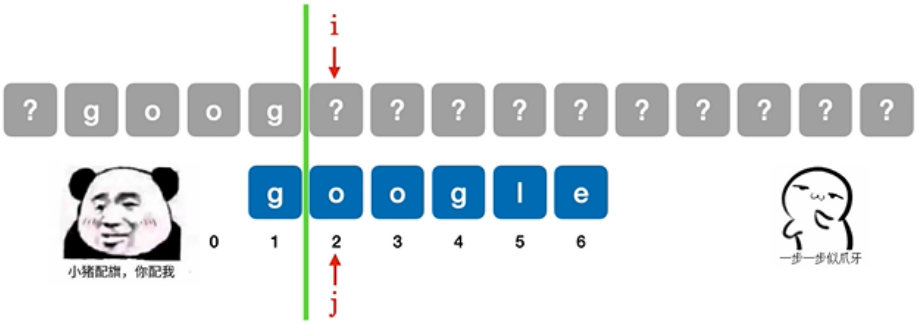
next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
	0	1	1	1		

在不匹配的位置前边，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

求模式串的next数组（手算练习）

next数组的作用：当模式串的第 j 个字符失配时，从模式串的第 next[j] 的继续往后匹配



next数组：

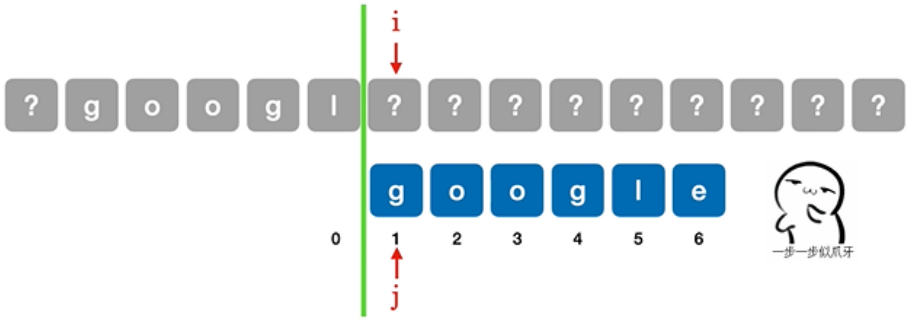
next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
	0	1	1	1	2	

在不匹配的位置前边，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

求模式串的next数组（手算练习）

next数组的作用：当模式串的第 j 个字符失配时，从模式串的第 next[j] 的继续往后匹配



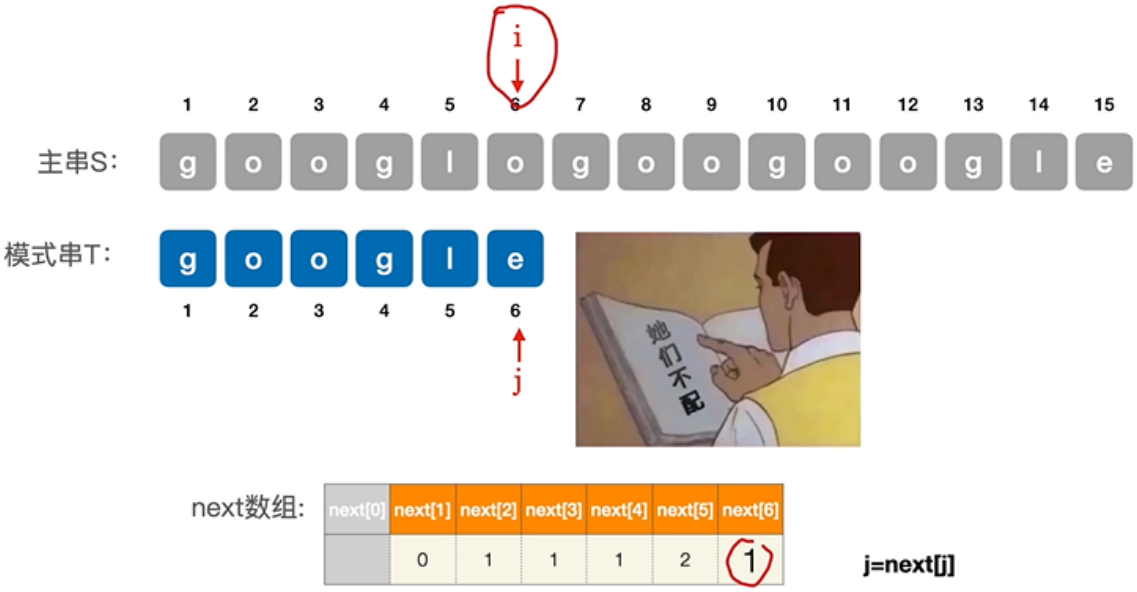
next数组：

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

在不匹配的位置前边，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

使用next数组进行模式匹配

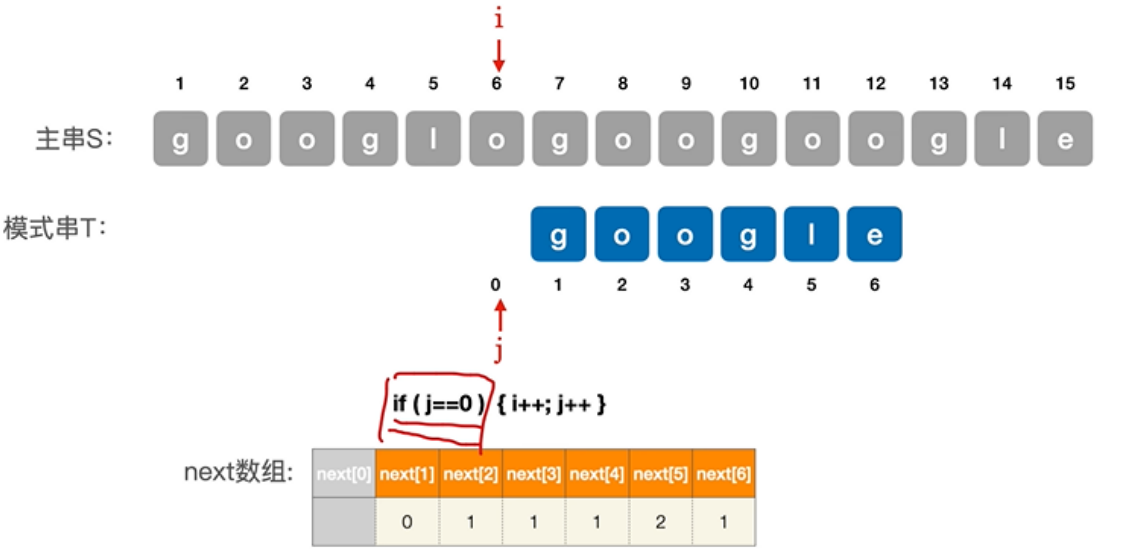


next数组:

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

j=next[j]

使用next数组进行模式匹配



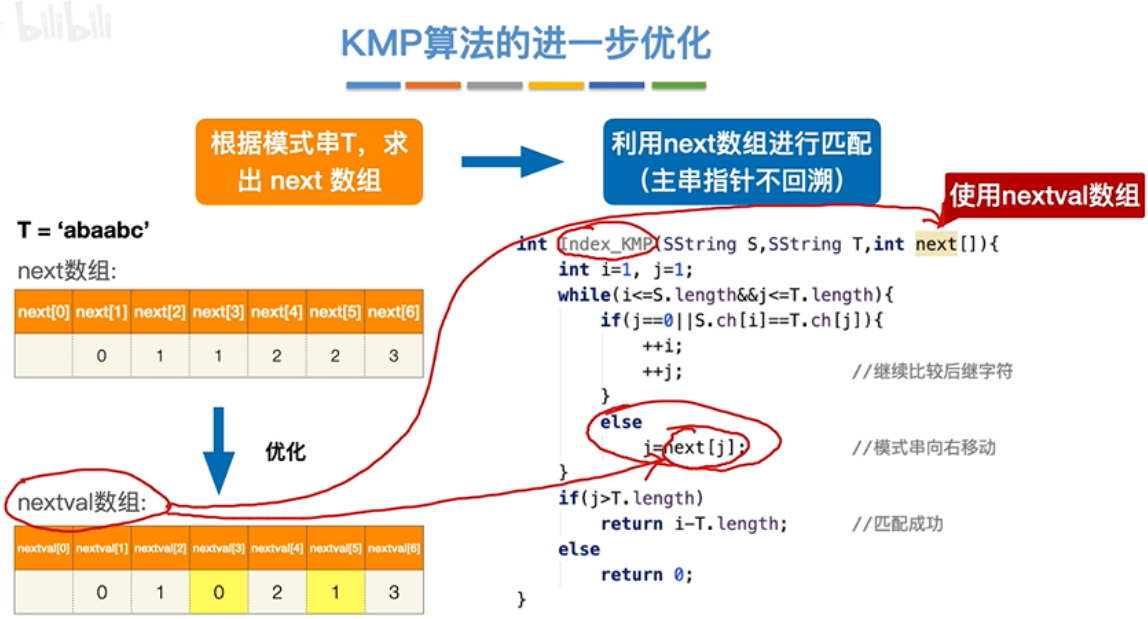
next数组:

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

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)

四、KMP算法优化



练习1：求nextval数组

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
nextval[j]	0	1	0	1	0	4

ababaa

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];
}
```