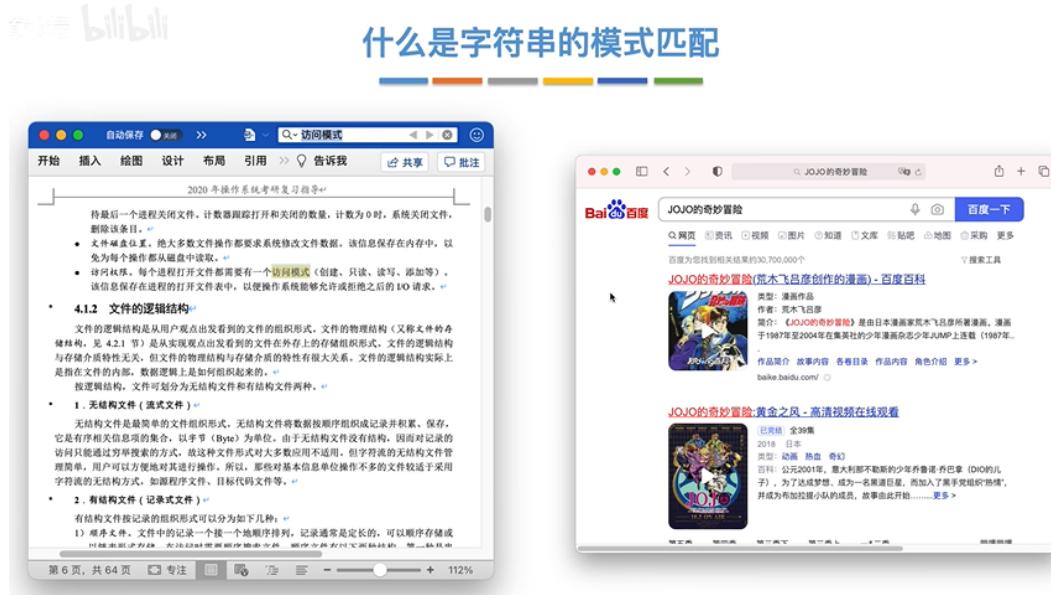


朴素模式匹配算法和KMP算法

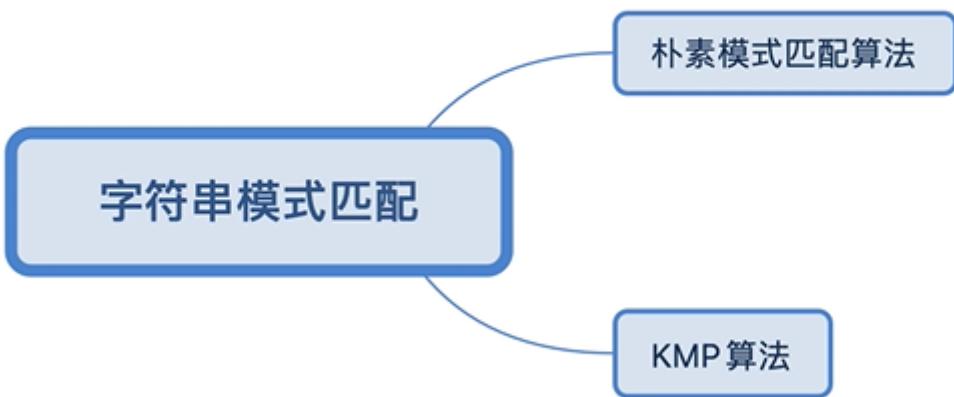
一、朴素模式匹配算法

- 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



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); //取出从位置 i 开始,
        if(StrCompare(sub, T)!=0) ++i; //长度为m的子串
        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
}

```

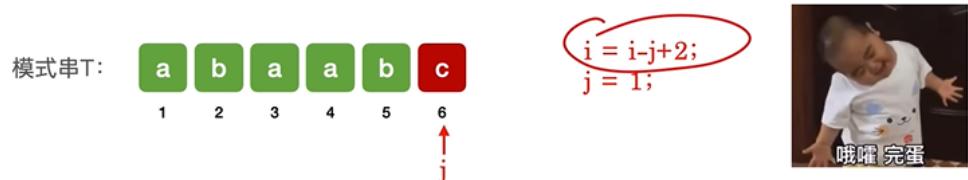
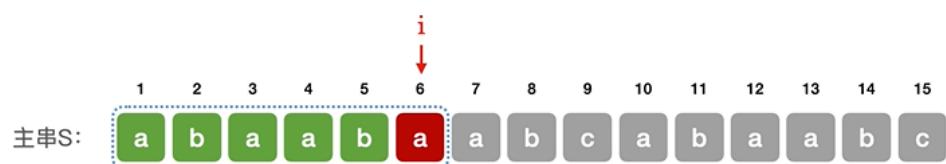
原来是这样啊



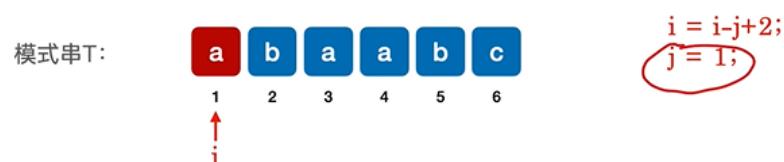
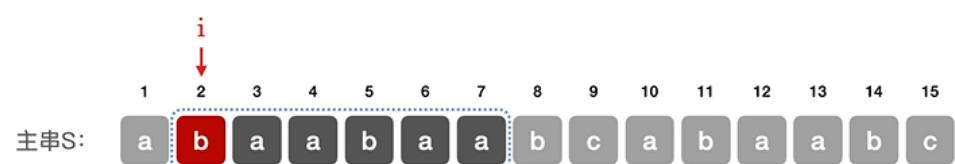
接下来: 不使用字符串的基本操作, 直接通过数组下标实现朴素模式匹配算法

王道考研/CSKAOYAN.COM

朴素模式匹配算法

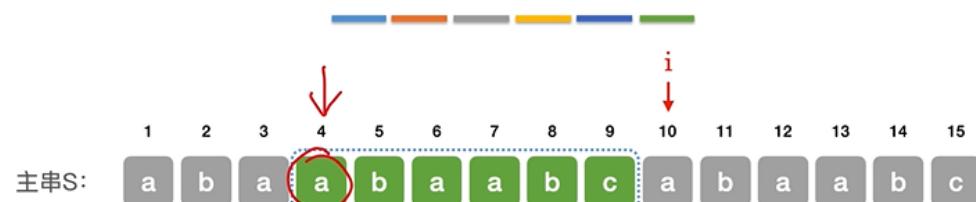


若当前子串匹配失败，则主串指针 i 指向下一个子串的第一个位置，模式串指针 j 回到模式串的第一个位置



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



模式串T:



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 --- $i - T.length$

```

1 int Index(SString S,SString T)
2 {
3     int i = 1,j = 1;
4     while(i <= S.length && j <= T.length)
5     {

```

```

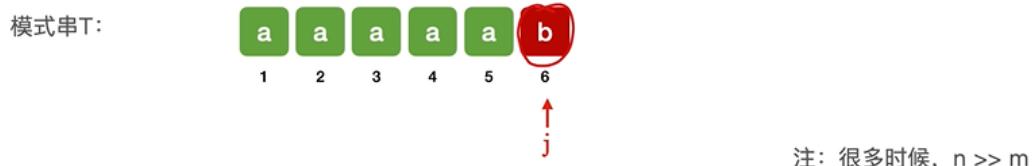
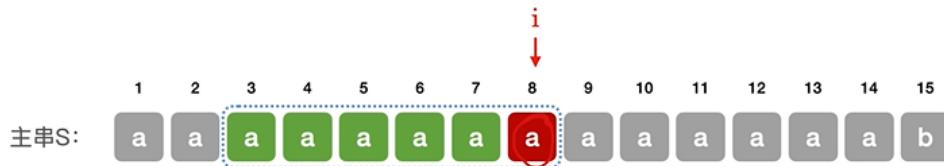
6     if(s.ch[i] == T.ch[j])
7     {
8         ++i;      //继续比较后继字符
9         ++j;
10    }
11    else
12    {
13        i = i - j + 2;
14        j = 1;    //指针后退重新开始匹配
15    }
16 }
17 if(j > T.length)
18     return i - T.length;
19 else
20     return 0;
21 }

```

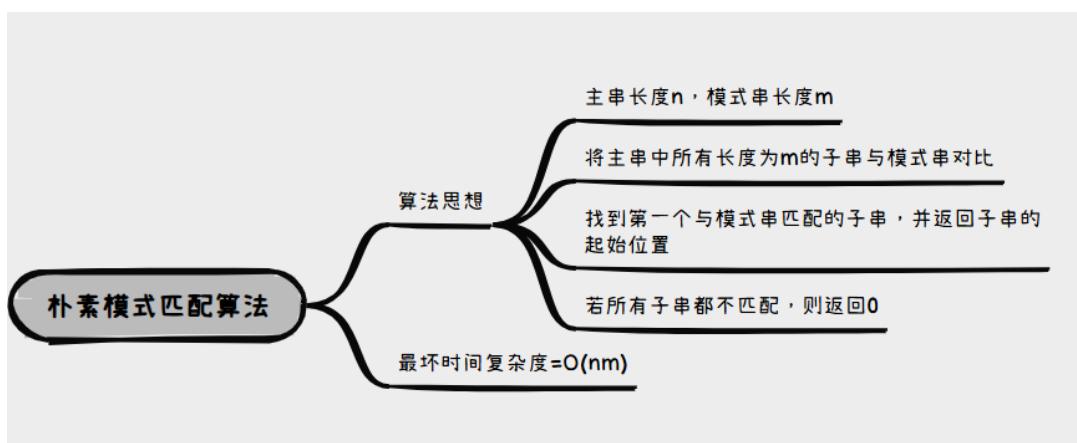
设主串长度为 n , 模式串长度为 m , 则

最坏时间复杂度 = $O(nm)$

最坏时间复杂度 = $O(nm)$



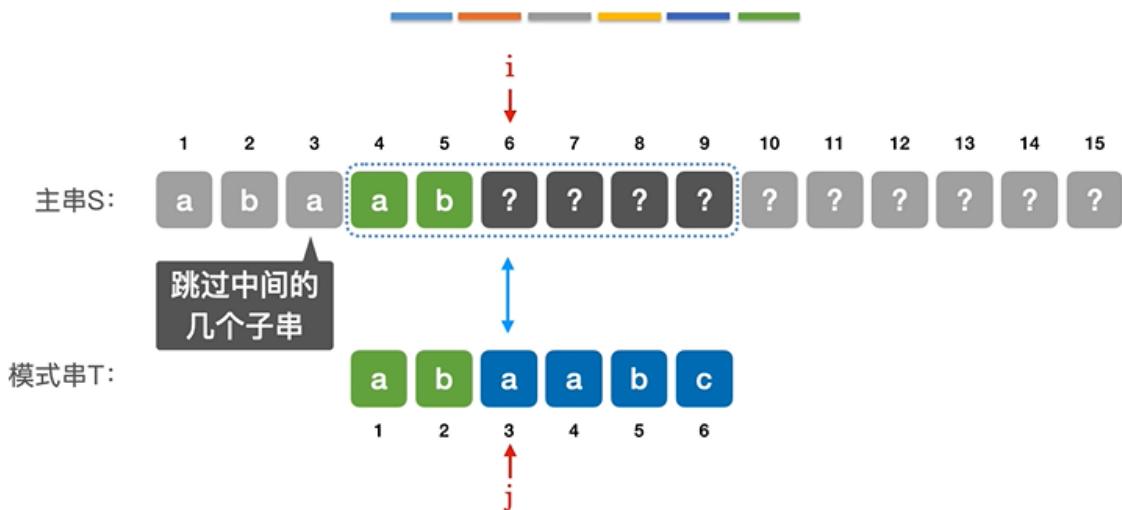
最坏的情况, 每个子串都要对比 m 个字符, 共 $n-m+1$ 个子串, 复杂度 = $O((n-m+1)m) = O(nm)$



二、KMP算法



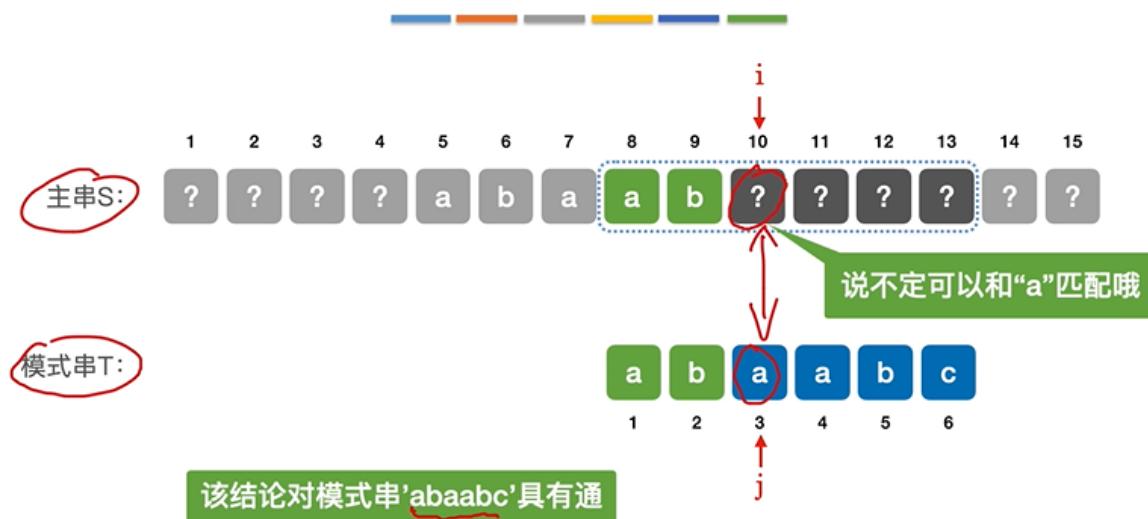
朴素模式匹配算法优化思路



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

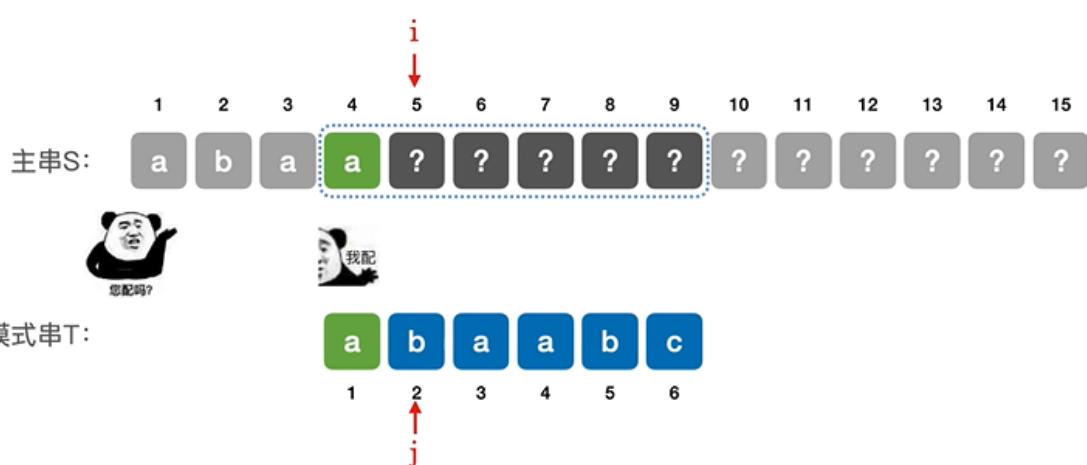


朴素模式匹配算法优化思路



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

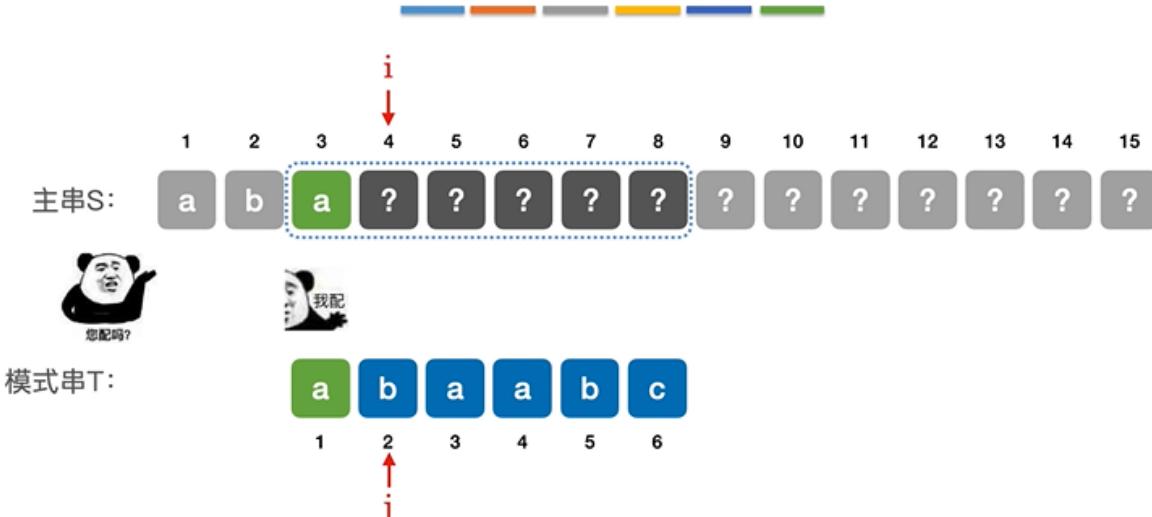
如果其他位置不匹配呢？



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 $j = 2$

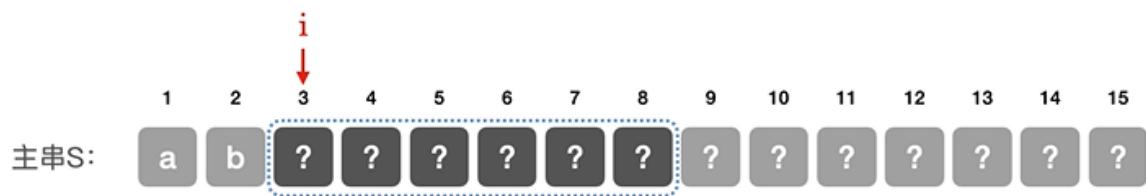
如果其他位置不匹配呢？



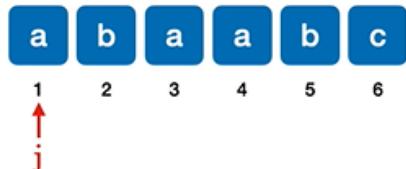
对于模式串 $T = 'abaabc'$, 当第4个元素匹配失败时? 怎么搞?

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？



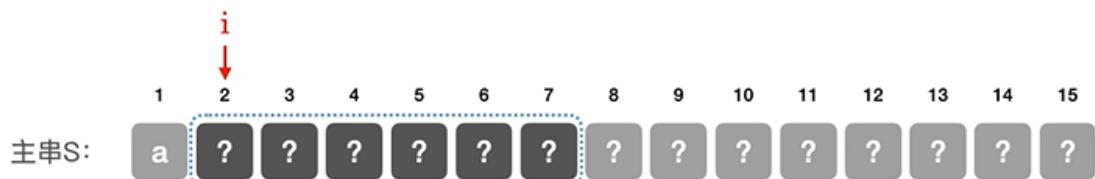
模式串T:



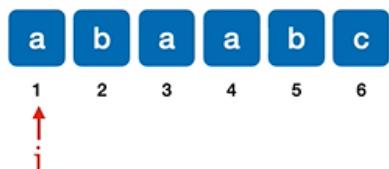
对于模式串 $T = 'abaabc'$, 当第3个元素匹配失败时? 怎么搞?

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？



模式串T:

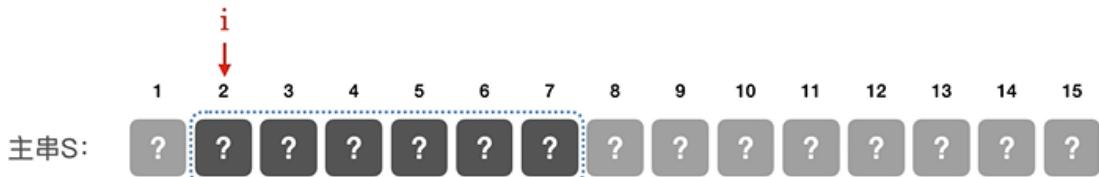


对于模式串 $T = 'abaabc'$, 当第2个元素匹配失败时? 怎么搞?

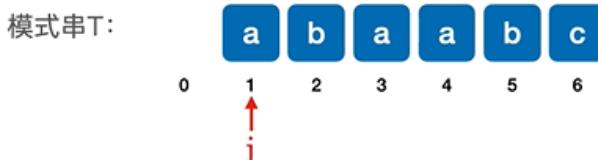
可令主串指针 i 不变, 模式串指针 $j = 1$



如果其他位置不匹配呢？



j=0, i++, j++

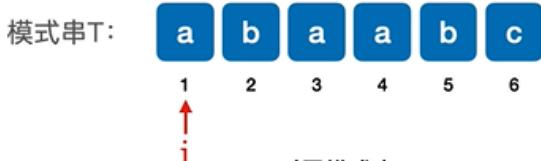
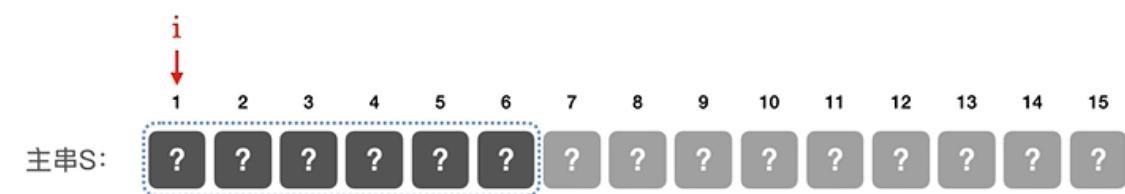


对于模式串 $T = 'abaabc'$, 当第1个元素匹配失败时? 怎么搞?

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 $j=2$

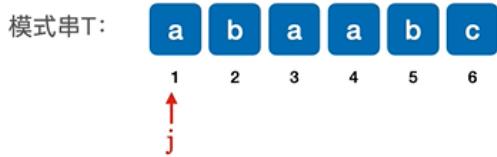
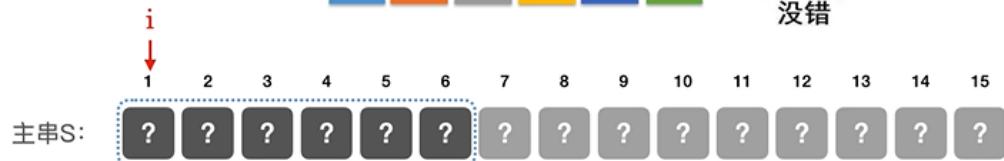
当第3个元素匹配失败时, 可令主串指针 i 不变, 模式串指针 $j=1$

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 $j=0, i++, j++$



KMP算法



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数组只和短短的模式串有关，和长长的主串无关

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 $j=0, i++, j++$



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)

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数组只和短短的模式串有关，和长长的主串无关

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 $j=0, i++, j++$

```
1 int Index_KMP(SSString S, SSString T, int next[])
2 {
3     int i = 1, j = 1;
4     while(i <= S.length && j <= T.length)
5     {
6         if(j == 0 || S.ch[i] == T.ch[i])
7         {
8             ++i;
9             ++j; //继续比较后继字符
10        }
11        else
12        {
13            j = next[j]; //模式串向右移动
14        }
15        if(j > T.length)
16            return i - T.length; //匹配成功
17    }
18}
```

```

17         else
18             return 0;
19     }
20 }

```



朴素模式匹配 v.s. KMP算法

```

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; ++j; //继续比较后继字符
        }
        else{
            i=i-j+2;
            j=1;
        }
    }
    if(j>T.length)
        return i-T.length;
    else
        return 0;
}

```

朴素模式匹配算法, 最坏时间复杂度 $O(mn)$

```

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;
        }
        else
            j=next[j];
    }
    if(j>T.length)
        return i-T.length;
    else
        return 0;
}

```

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 $O(n)$

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 $\text{next}[j]$ 的继续往后匹配



0 1 2 3 4 5 6

任何模式串都一样, 第一个字符不匹配时, 只能匹配下一个子串, 因此, 往后余生, $\text{next}[1]$ 都无脑写 0

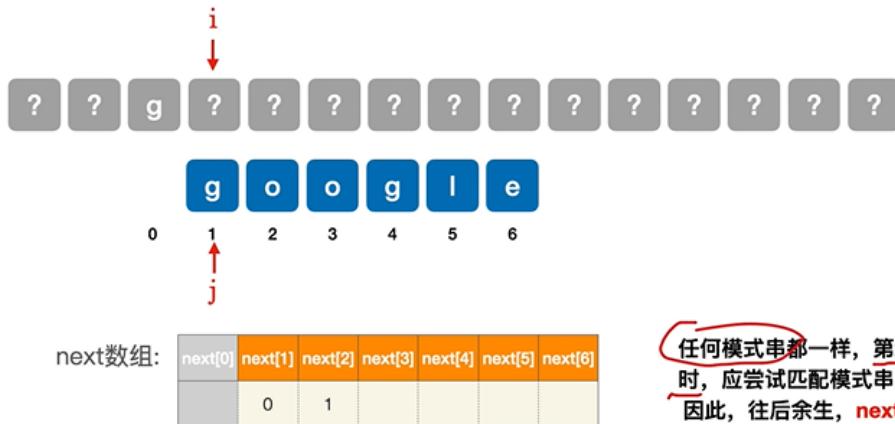
next数组:	next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
		0					



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 $\text{next}[j]$ 的继续往后匹配



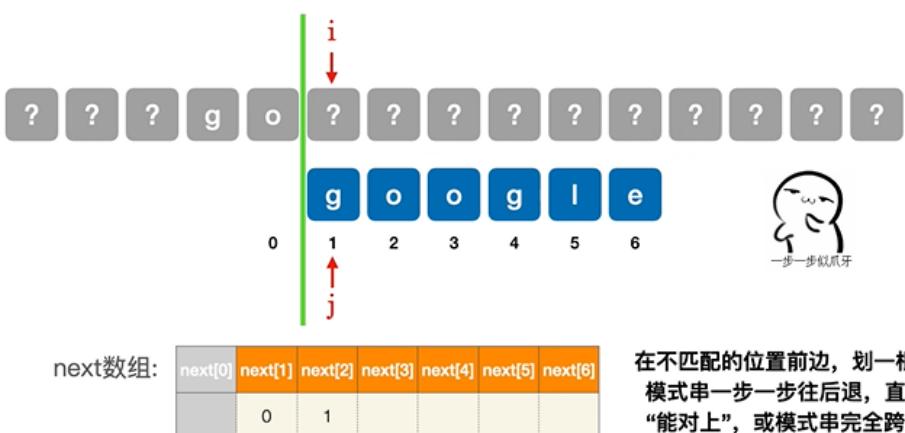
任何模式串都一样，第2个字符不匹配时，应尝试匹配模式串的第1个字符，因此，往后余生，next[2]都无脑写1



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 $\text{next}[j]$ 的继续往后匹配

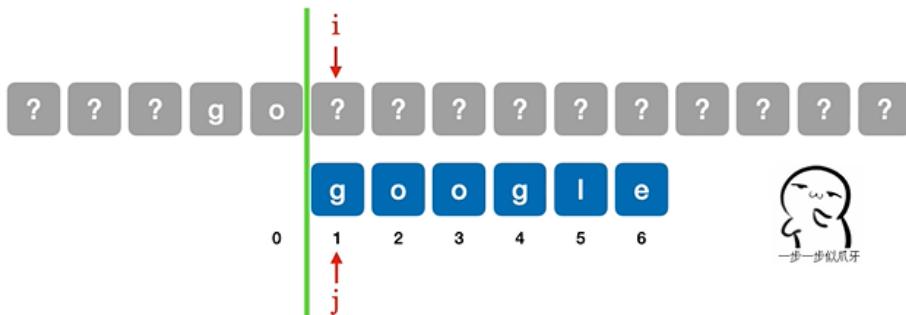


在不匹配的位置前边，划一根美丽的分界线
模式串一步一步往后退，直到分界线之前
“能对上”，或模式串完全跨过分界线为止



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 $\text{next}[j]$ 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				

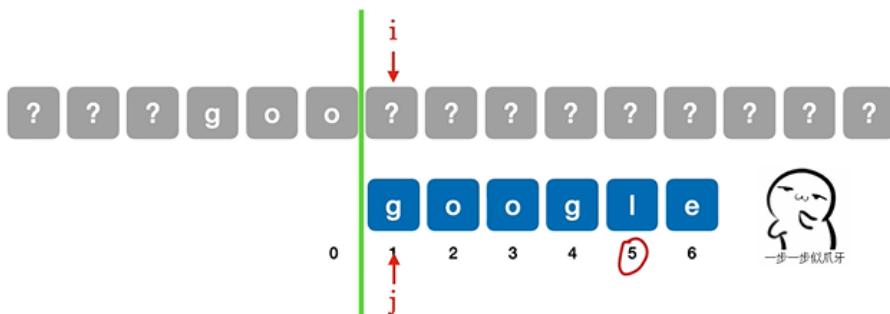
在不匹配的位置前边，划一根美丽的分界线
模式串一步一步往后退，直到分界线之前
“能对上”，或模式串完全跨过分界线为止

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 $\text{next}[j]$ 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

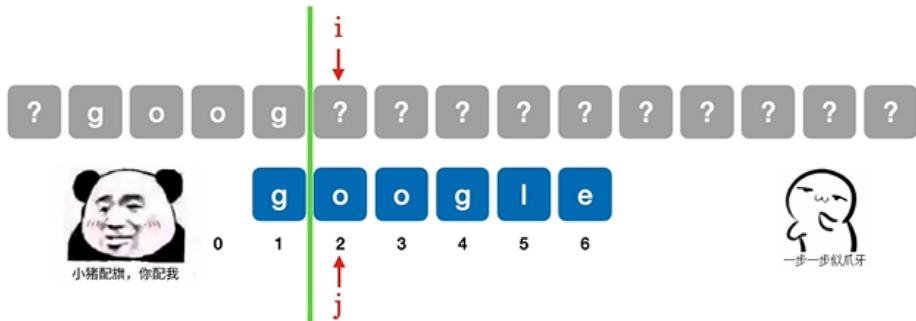
此时 j 指向哪儿，next数组值就是多少



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 $\text{next}[j]$ 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

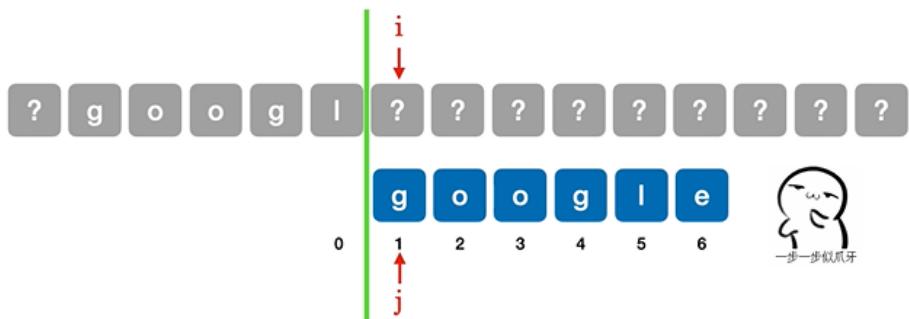
此时 j 指向哪儿，next数组值就是多少



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 $\text{next}[j]$ 的继续往后匹配

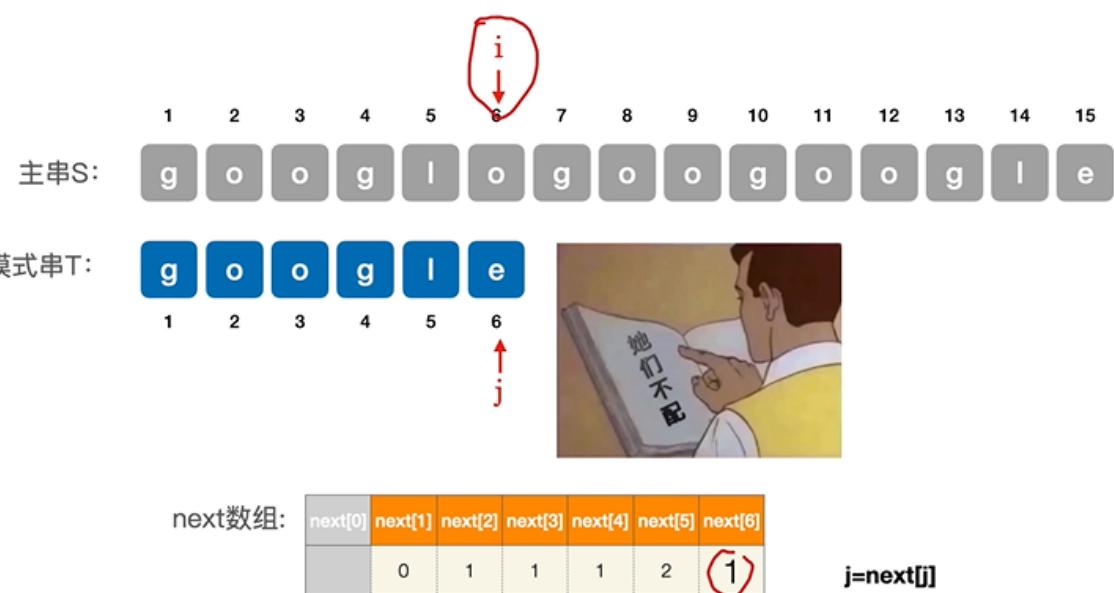


next数组:	next[0]	next[1]	next[2]	next[3]	next[4]	next[5]	next[6]
	0	1	1	1	2	1	

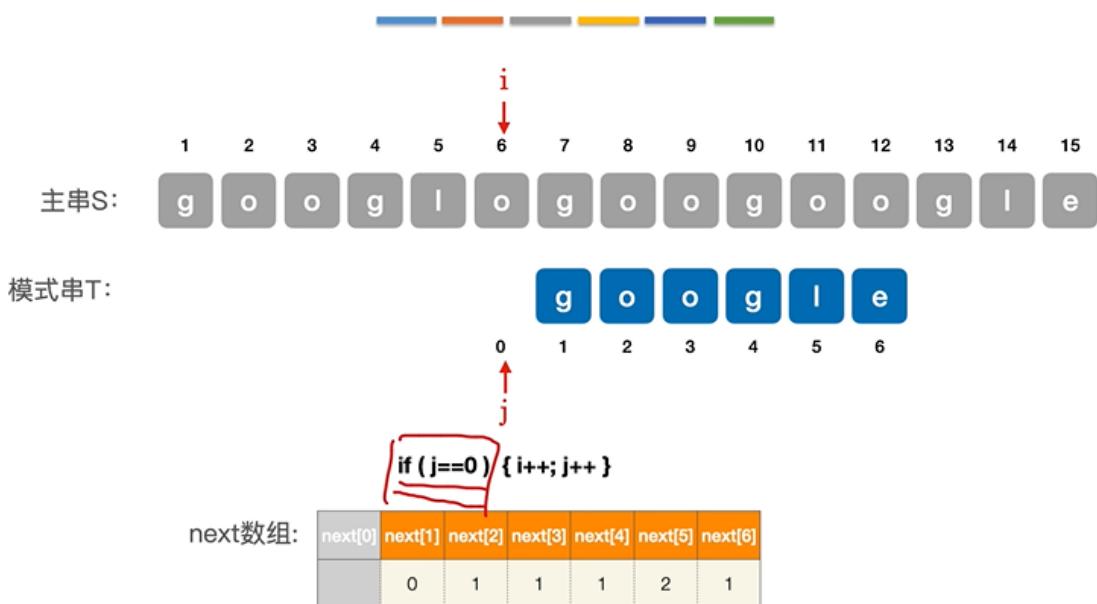
在不匹配的位置前边，划一根美丽的分界线
模式串一步一步往后退，直到分界线之前
“能对上”，或模式串完全跨过分界线为止

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



使用next数组进行模式匹配



KMP算法——求next数组

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)

四、KMP算法优化

KMP算法的进一步优化

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

利用next数组进行匹配
(主串指针不回溯)

使用nextval数组

nextval数组:

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

优化

```
int index_KMP(SSString 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;
        }
        else
            i=next[j];
    }
    if(j>T.length)
        return i-T.length;
    else
        return 0;
}
```

练习1：求nextval数组

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];
}
```

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[i] 0 1 0 1 0 4