

几个概念：

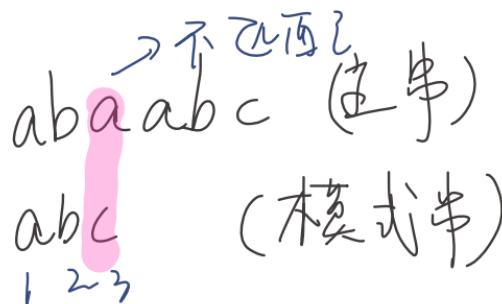
1. 前缀：包含首位字符但不包含末位字符的字符串。

如： ab c d ab
为前缀

2. 后缀：包含末位字符但不包含首位字符的字符串

3. next 数组定义：

主串与模式串某次匹配时不匹配时
模式串退回的位置。

如：

注: next[] 下标以 1 开始

则据定义 $\text{next}[3]$ 必为 1

ab $\overset{\downarrow i}{a}$ ab c

ab c

↑

$j=1 = \text{next}[3]$

4. $\text{next}(j)$ 值的计算公式:

第 j 位字符前 i-1 位构成子串

的前后缀重合字符串 + 1

(Max)

$\text{str}:$

ab a ab c a c.

3

1 2 3 4 5 6 7 8

进一步解释：

如果 $\text{next}[b] = 3$ 表明

若主串与模式串在^(b)处不匹配时

应返回模式串的第3位开始：

如： abaaab a b c a

abaaab c

↙

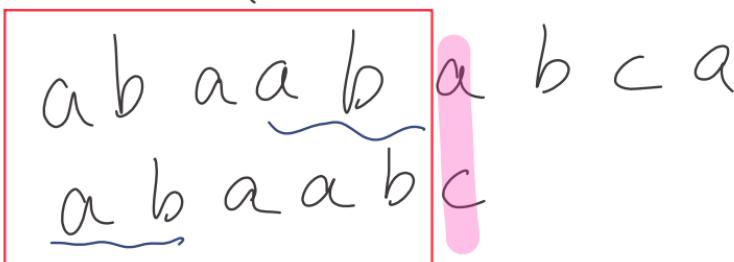
(ab) a a b c

↑
第3位

KMP 算法分析：

对于主串的遍历保证不返回，遇到不匹配的，通过利用 next 序列来找到模式串的对位位置，重新进行了匹配，直到匹配成功，或者遍历结束仍未匹配成功。

next 序列分析：



本质：找最长的相同前后缀

当然，这里的前缀段可以有
文集也可无文集，

如： $\underline{ab} \underline{ab}$, $\underline{ab} \underline{a}$, $\underline{ab} \underline{aba}$

注意到 $next[i]$ 只与模式串本身
有关，下面分析 $next[i]$ 的求
法：

对于模式串

$t_1 t_2 t_3 t_4 \dots t_{n-2} t_{n-1} t_n$

先考虑极端情况：

① 对于模式串中的第一个，其意义
为第一个就不匹配，那么

又能从模式串的第一位和主串的下一位重新比较

($i++$; $j++$) \Rightarrow 为了方便

字符串一，令 $j++ = 1$

则只需让 j 重新赋值为 0

$\Rightarrow \text{next}[0] = 0$

($f_n:$ abc d ab)
 a b d)

\Rightarrow abc d ab
 a b d

→如果与第一次匹配

②对于模式串的第二位，其
意义为第二位不匹配而第
一位匹配了，那么应该将模
式串的第一位与主串的该
位比对，也即查看模式
串的子串为一个字符串时，前
后缀相同的最大长度为 0，

令 $\text{next}[1] = 0 + 1 = 1$ 已可。

如： ab c a \Rightarrow ab c a
 b a.

如果与第一位相同：

那么应该将模式串第一位与主串下一位对比，如：

$\begin{array}{cccccc} a & b & c & a & b \\ & b & b \end{array} \Rightarrow \begin{array}{cccccc} a & b & c & a & b \\ & & & & b & b \end{array}$

那么同①可知令 $\text{next}[1] = 0$

之后便统一处理

注：在未修改的 next 中，将第二位统一设置为 1

③ 对于模式串的最后一个位置：

$t_1 t_2 \dots t_{n-2} t_{n-1} t_n$

$\text{next}[1] [2] \dots [n-2] [n-1] [n]$

考虑 $\text{next}[n-1]$:

$t_1 t_2 \dots t_{\text{Lmax}} t_{\text{Lmax}+1} \dots t_{n-2} t_{n-1}$

表示 $t_1 \sim t_{n-1}$ 的相等前缀 且不包含 t_{n-1}

长为 $\text{next}[n-1] - 1 = \text{Lmax}$

next 这个

现在加入 t_{n-1} ，那么其实只
需要考虑 $t_{\text{Lmax}+1}$ 与 t_{n-1} 是
否相等即可：

(1) 若 $t_{L_{\max}+1} = t_{n-1}$,

$$\begin{aligned}\Rightarrow \text{若 } next[n] &= (L_{\max}+1)+1 \\ &= next[n-1]+1\end{aligned}$$

图: ab ab a a
next 0 1 1 2 3
 与 $t_{next[4]}$
 = $t_2 = b$
 相等

(2) 若 $t_{L_{\max}+1} \neq t_{n-1}$ 全 $t_{L_{\max}+1}$

也即 $t_1 \dots t_{L_{\max}} t_{L_{\max}+1}$

$t_{n-1-L_{\max}} \dots t_{n-2} t_{n-1}$

在此基础上接着找最长的相等

前 后 缩 :

那么关键就在于 t_{h-2} 必须与前缀

和第 i 位开始重新匹配，
→ 方便利用 next

这里固定 $t_1 \dots t_{l_{\max}} t_{l_{\max}+1}$

移除 $t_{n-1-l_{\max}} \dots t_{n-2} t_{n-1}$

那么不难发现，只需找到

$t_1 \dots t_{l_{\max}} = \text{next}[h-i] - 1$

即 k 的相等前缀长度

即为 $\text{next}[\text{next}[n-1]-1] - 1 = \text{next}[k] - 1$

即 $t_1 \dots t_{\text{next}[k]-1} t_{\text{next}[k]}$

$t_{\text{pos}} \dots t_{h-2} t_{h-1}$

注意此时仍需继续考虑 t_{n-1} ！

那么只有两种结果：

$$t_{\text{next}[k]} = t_{n-1} \text{ 和 } t_{\text{next}[k]} \neq t_{n-1}$$

继续上述操作即可。

令 $k = \text{next}[k] - 1$ ，判断 $t_{\text{next}[k]}$ 和 t_{n-1}

下面考虑上述操作的终止条件

$$\text{1. } \text{① } t_{\text{next}[k]} = t_{n-1}$$

$$\text{那么 } \text{next}[n] = (\text{next}[k] - 1) + 1 + 1$$

$$= \text{next}[k] + 1$$

② $\text{next}[k] = 1$, 且 $t_1 \neq t_m$

$\Rightarrow \text{next}[n] = 0 + 1 = 1$

至此, KMP 算法分析结束.

附 next 算法分析：(未修正)

```
void get_next(SSString T, int next[])
```

//求模式串 T 的 next 函数值并存入数组 next

```
i=1;next[1]=0;j=0;  
while(i<T[0])  
{  
    if(j==0 || T[i]==T[j]) {++i; ++j; next[i]=j;}  
    else j=next[j];  
}
```

$\text{next}[i] = ++j$ -
 ↑
    ~~~~~

初始化： $\text{next}[1]=0, i=1, j=0$

$\Rightarrow \text{next}[2]=1, i=2, j=1$

$\nexists \text{next}[3]$ , 判断  $T[2]$  与  $T[\text{next}[2]]=T[1]$

$\begin{cases} T[2]=T[1], \text{next}[3]=\text{next}[2]+1 \\ = 2 - j=2 \end{cases}$

$T[2] \neq T[1]$ , 判断  $T[2]$  与  $T[\text{next}[1]]$

$j=0 \Rightarrow \text{next}[3]=+j=T[0]$   
 $= 1, j=1$

★  $\text{next}(++i) = ++j$  保住了

下一行  $\text{next}[i], j$  为上一行的  $\text{next}$

# 附KMP算法分析：

```
1. int Index_KMP (SString S, SString T, int pos){  
2.     i=pos, j=1; S.strlen T.strlen  
3.     while (i<S[0] && j<T[0]) {  
4.         if (j==0 || S[i]==T[j]) { i++;j++; }  
5.         else j=next[j]; T.strlen // i不变, j后退  
6.     } T.strlen // 匹配成功  
7.     if (j>T[0]) return i-T[0]; // 返回不匹配标志  
8.     else return 0;  
9. }
```

↓ *j=0* 表明 i 和 j 都移到了  
再匹配