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：abcdab

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：

主串与模式串某位不匹配时模式串退回的位置.

如：

abaabc （主串） → 不匹配了

abc （模式串）

1 2 3

注：next[ ] 下标从1开始

则据定义 next[3] 应为1

a b a a b c

a b c

$j=1=next[3]$

4. next[j] 值的计算式：

第j位字符前j-1位构成的串的前后缀重合字符数+1 (Max)

如： a b a a b c a c

3

1 2 3 4 5 6 7 8

进一步解释：

如果 next[6]=3 表明

若主串与模式串在$^{(6)}$处不匹配时

应返回模式串的第3位开始：

如： a b a a b a b c a

a b a a b c

(a b) a a b c

第3位

KMP算法分析：

对于主串的遍历保证不回退，遇到不匹配的，通过调用next数组来找到模式串的对应位置，重新进行匹配，直到匹配成功，或者遍历结束仍未匹配成功、

next数组分析：

a b a a b a b c a

a b a a b c

本质：找最长的相同前后缀

当然，这里的前后缀可以有交集也可无交集，

如：abab，aba，ababa

注意到next[]只与模式串本身有关，下面分析next[]的求法：

对于模式串

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

先考虑极端临界情况：

① 对于模式串的第一位，其意义为第1位就不匹配，那么

只能从模式串的第一位和主串的下一位重新比较

(i++, j++) ⇒ 为了方便使

得值一，令 j++ = 1

则只需进行重新赋值如

⇒ $next[0] = 0$

(如：

```
abcdab
   abd
```
)

⇒

```
abcdab
    abd
```

→如果与第一位不同

②对于模式串的第二位，其意义为第二位不匹配而第1位匹配了，那么应该将模式串的第一位与主串的该位比对，此时应看模式串前子串为一个字符时，前后缀相同的最大长度为0，令next[i] = 0+1 = 1即可。

如：abca
　　　ba
⇒ abca
　　　ba

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：

a b c a b $\Rightarrow$ a b c a b

  b b   b b

那么由①可知令next[1]=0

更方便统一处理

注：在求修正的next中，将第二位统一设置为1

③ 对于模式串的最后一位：

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

$$next[1]\ [2] \cdots\cdots [n-2]\ [n-1]\ [n]$$

考虑 $next[n-1]$：

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

表示 $t_1 \sim t_{n-1}$ 的相等前后缀最长为 $next[n-1]-1 = Lmax$

next定义

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

(1) 若 $t_{Lmax+1} = t_{n-1}$,

那么 $next[n] = (L_{max}+1)+1$
$= next[n-1]+1$

如: a b a b a
next 0 1 1 2 3

对 next[4] 时

与 $t_{next[4]} = t_2 = b$ 相等

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

也即 $t_1 \cdots\cdots t_{Lmax}\ t_{Lmax+1}$

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

在此基础接着找最长的相等前后缀:

那么关键是在于 t_{n-2} 应该与前缀的第几位开始重新匹配，→ 方便利用 next

这里固定 $t_1 \cdots t_{Lmax} t_{Lmax+1}$

移动 $t_{n-1-Lmax} \cdots t_{n-2}\ t_{n-1}$，

那么不难发现，只需找到

$t_1 \cdots t_{Lmax=next[n-1]-1}$.

最长的相等前后缀长度

即为 $next[next[n-1]-1]-1 = next[k]-1$

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

$t_{pos} \cdots t_{n-2}\ t_{n-1}$

注意此时仍需继续考虑 t_{n-1}!

那么只有两种结果:

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

继续上述操作即可,

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

下面考虑上述操作的终止条件: ① $t_{next[k]} = t_{n-1}$

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

$$= next[k]+1$$

② $next[k] = 1$，且 $t_1 \neq t_{n-1}$

那么 $next[n] = 0 + 1 = 1$

至此，KMP算法分析结束.

附next算法出报：（未修正）

```
void get_next(SString 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];
    }
}
```

next[++i] = ++j

初始化：next[1]=0, i=1, j=0

⇒ next[2]=1, i=2, j=1

求next[3]，判断 T[2] 与 $T_{next[2]}$ = T[1]

T[2]=T[1]，next[3] = next[2]++
= 2，j=2

T[2]≠T[1]，判断 T[2]与 $T_{next[1]}$

j=0 ⇒ next[3]=++j = T[0]
= 1，j=1

★ next[++i]=++j 保证了
求下一位 next 时，j 为上一位的 next 值

附KMP算法函数:

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

j=0表明i和j都移一位再匹配