string-searching / Locating 称为串的"模式匹配"：主串 S 为目标串，子串 T 为模式串，记为 Index(S,T)

- 直接方法：一个个移动地比。

（顺序表，第0位为表长）（加入这个参数便于继续找下一个匹配位置。）

```
int INDEX (Sstring S, Sstring T, int pos)
{
  int i = pos;
  while (i ≤ S[0] && j ≤ T[0])
  {
    if( S[i] == T[j])          // 匹配上了
      {i++; j++;}              // next
    else                       // 没匹配上
      {i = i-j+1 = i-(j-1)+1 = i-j+2; j = 1;}
  }
  if (j > T[0]) return (i-T[0]);
  else return 0;
}
```

while 即一便便反复执行其内的这些行代码。每执行完一轮判断一下条件，满足则继续执行下一遍

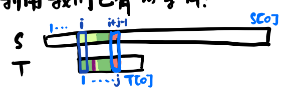
能不套循环就不套。

j从1开始则为j-1，从0开始则为j-0。

i-j+1：下一位。[刷新目标串 i 的比较开始位置] j=1：归0（开头）

讨论循环跳出究竟是因为没满足哪个条件。

- 但并没有充分利用我们已有的条件：



比较到某位上失配，反过来即知此位之前全都一一匹配上了；我们想利用这一既知信息，让T一次往前多移几步

K.M.P.三人发现，我们可以考虑 T上 1…j-1 位的最长相同前后缀 [指的其实是模式串的[1,j-1]子串的]

利用后缀与S的匹配直接把前缀移到后缀对应位置上。尝试用前缀的下一个字符与S上失配字符匹配。递归使用该方法即可。

即不对i作回退，不直接让j返回1；而是先让i不动，递归修改j（往前），拿j处与i对齐比较（一样则继续比，不一样则再往前找j）

↓ 边界为，都试了j=1了还匹配不上。这证明把T的头拉到S失配处i还无法匹配。i处自增，此时i++到下一位，j++到1，重新头头匹配。

★ Pf.

假设最长相同前后缀为 ▬

假设直接作KMP的移动会漏移更少时的全匹配，假设出这种 supposing string 的移动位置。

上面箭头形成的等推关系使模式串中有了 ▬▬ 这一更长的相同前后缀，与假设矛盾！

因此，不会漏！KMP作的移动就是第1个可能全匹配的位置。□

（被比较的字符）注："移动"由对齐确定

不断利用左述结论，保证检查的都是有可能情况

这样，核心问题就转为求拿哪个j和i尝试匹配（也即求模式串各子串最长相同前后缀）

在此之前，我们不妨先搭建一下KMP的框架。

NEXT(SString T, j) 直接用数组存储就好呀！不必再调函数

该数组下标即对应于T上的字引值

```
int index KMP (SString S, SString T, int pos)
{ int i = pos; j = 1;
  while (i ≤ S[0] && j ≤ T[0])
  { if ( S[i] == T[j] || j == 0)
      {i++; j++;}
    else
      { j = NEXT[j]; }
  }
  if (j > T[0]) return (i-T[0]);
  if (i > S[0]) return 0;
}
```

```
void getNEXT ( SString T, int NEXT[])
{ NEXT[1] = 0; int j = 1; int k = 0;
  while (j < T[0])
  { if(k == 0 || T[j] == T[k])
      { j++; k++; NEXT[j] = k; }
    else { k = NEXT[k]; }
  }
}
```

引入中间变量k记录NEXT[j]，避免索引越界，且更简洁

如何获得NEXT数组呢？

- 直接看赋值 :)

我们在匹配过程中关注的是j位前面的[1,j-1]子串的最长相同前后缀，所以赋给后一位的NEXT值。

	a	b	a	a	b	c	a	c
NEXT	0	1	1	2	2	3	1	2

start 0，0+1

+1

maxL其对齐+1，送"下一位"

- 判断新的+结合递归。利用KMP本身思想！（移位使用的）

看新检查的字符与其位置上的NEXT（即前后缀相同前后缀的下一位）是否一致。

一致，则下一位的NEXT为NEXT索引到的字符的NEXT+1

不一致，则递归再往前找更小的最大相同前后缀的前缀的下一位来比较

使用NEXT[NEXT[j]]（前也偏，后也偏）

	a	b	a	a	b	c	a	c
NEXT	0	1	1	2				

为0，赋0+1，则不比，直接

a、a≠b，NEXT[1]=0，赋0+1

KMP算法改进

若 i,j 失配, j 回到[NEXT[j]]来和 S[i] 比, 但此时的和 T[j] 一样, 则肯定还匹配不上, 得再递归找下一个NEXT再比。

都是T的事儿, 我们可以优化 get NEXT 来直接获得目标与 T[j] 不等的 NEXT 值.

```
void get_nextval (SString T, int next[ ])
  { NEXT [1] = 0;  int j = 1;  int k = 0;
    while (j < T[0])
    { if (k == 0 || T[j] == T[k])
        { j++; k++;
          if (T[j] != T[k]) nextval [j] = k;
          else    nextval[j] = nextval[k];
        }
      else    k = nextval [k];
    }
  }
```

因为从一开始我们就这样赋值,
保证了再取一层nextval指引的索引处的
字符与k处字符不同 或指引的索引为0.