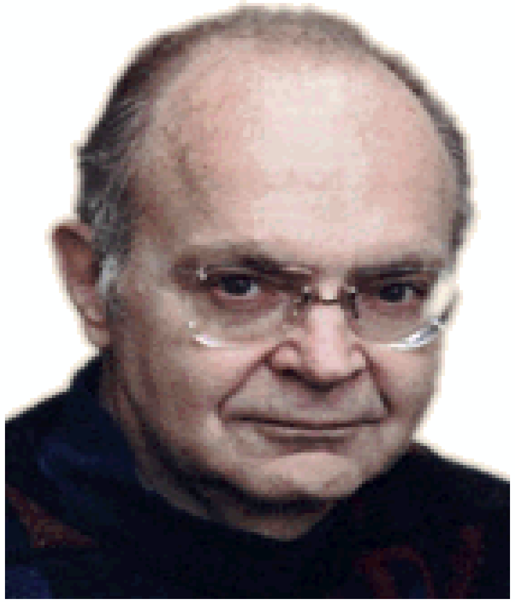
KMP（Knuth Morris Pratt）算法

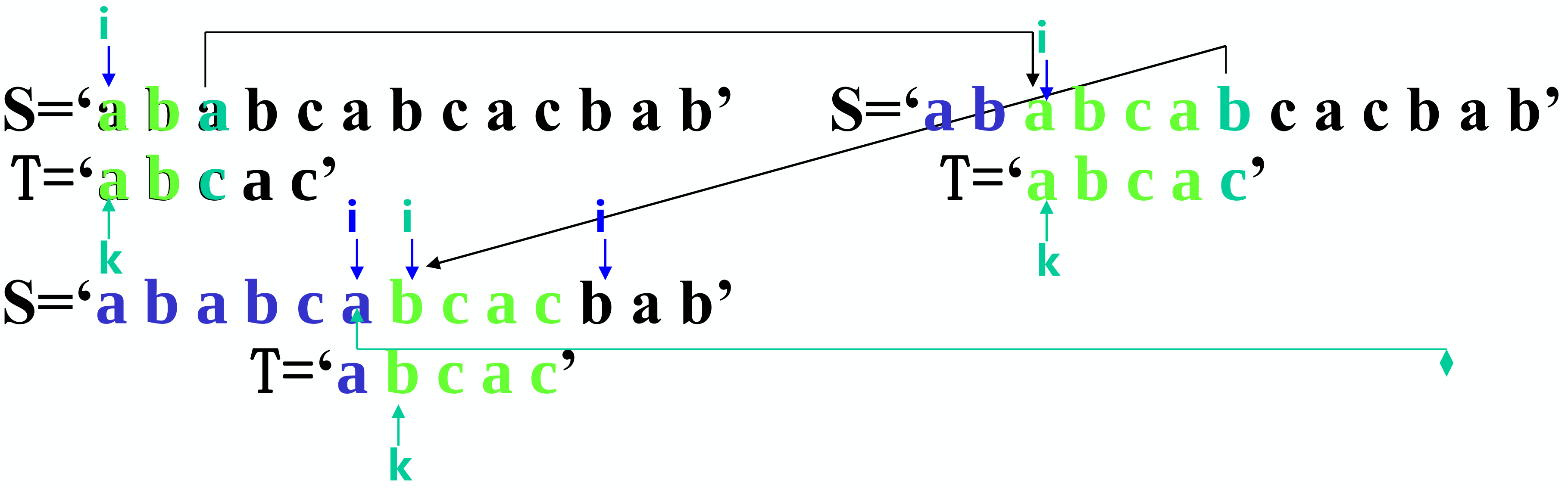
《计算机程序设计艺术 第1卷 基本算法》
《计算机程序设计艺术 第2卷 半数值算法》
《计算机程序设计艺术 第3卷 排序与查找》

http://www-cs-faculty.stanford.edu/~knuth/



KMP算法设计思想

利用已经部分匹配的结果而加快模式串的滑动速度？
且主串S的指针i不必回溯！可提速到O(n+m)！



第 1 次匹配　　s=abacaba　　**i=4**

失败

p=abab　　**j=4**

p=abab　　**j=2**

因$p_1 \neq p_2$, $s_2=p_2$, **必有**$s_2 \neq p_1$, **又因**$p_1=p_3$, $s_3=p_3$, **所以必有**$s_3=p_1$**。因此, 第二次匹配可直接从** $i=4$， $j=2$**开始。**

改进：每趟匹配过程中出现字符比较不等时，不回溯主指针i，利用已得到的“部分匹配”结果将模式向右滑动尽可能远的一段距离，继续进行比较。

$$
\begin{array}{ccccccccccc}
s_1 & s_2 & s_3 & \cdots & s_{i-j+1} & s_{i-j+2} & \cdots & s_{i-2} & s_{i-1} & s_i & s_{i+1} \\
 & & & & \| & \| & & \| & \| & \color{red}{\neq} & \\
 & & & & p_1 & p_2 & \cdots & p_{j-2} & p_{j-1} & p_j & p_{j+1} \\
 & & & & & & \| & & \| & & \\
 & & & & & & p_1 & \cdots & p_{k-1} & p_k & p_{k+1}
\end{array}
$$

① "$p_1p_2\ldots p_{k-1}$" = "$s_{i-k+1}s_{i-k+2}\ldots s_{i-1}$"

② "$p_{j-k+1}p_{j-k+2}\ldots p_{j-1}$" = "$s_{i-k+1}s_{i-k+2}\ldots s_{i-1}$" (部分匹配)

③ "$p_1p_2\ldots p_{k-1}$" = "$p_{j-k+1}p_{j-k+2}\ldots p_{j-1}$" (真子串)

为此,定义next[j]函数，表明当模式中第j个字符与主串中相应字符“失配”时，在模式中需重新和主串中该字符进行比较的字符的位置。

$$
next[j]=\begin{cases}
\max\{\,k \mid 1<k<j,\text{且}\ \text{“}p_1\ldots p_{k-1}\text{”}=\text{“}p_{j-k+1}\ldots p_{j-1}\text{”}\,\} & \text{当此集合非空时} \\
0 & \text{当}\ j=1\ \text{时} \\
1 & \text{其他情况}
\end{cases}
$$

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

- 如何求next函数值

1. next[1] = 0;表明主串从下一字符s_{i+1}起和模式串重新开始匹配。i = i+1; j = 1;

2. 设next[j] = k，则next[j+1] = ?

① 若$p_k=p_j$，则有 "$p_1 \cdots p_{k-1}p_k$"="$p_{j-k+1} \cdots p_{j-1}p_j$"，如果在

j+1发生不匹配，说明next[j+1] = k+1 = next[j]+1。

② 若$p_k \neq p_j$，可把求next值问题看成是一个模式匹配问题，整个模式串既是主串，又是子串。

$$
\begin{array}{ccccccccl}
p_1 & p_2 \cdots & p_{j-k+1} & \cdots & p_{j-1} & p_j & p_{j+1} & & \text{next}[j]=k \\
 & & \| & & \| & \neq & & & \\
 & & p_1 & \cdots & p_{k-1} & p_k & p_{k+1} & & \text{next}[k]=k' \\
 & & & p_1 \cdots\cdots & & p_{k'} & p_{k'+1} & & \text{next}[k']=k'' \\
 & & & & p_1 \cdots & p_{k''} & p_{k''+1} & & \text{next}[k''']=k'''
\end{array}
$$

- **若$p_{k'}=p_j$，则有“$p_1\cdots p_{k'}$”=“$p_{j-k'+1}\cdots p_j$”，**
  $next[j+1]=k'+1=next[k]+1=next[next[j]]+1$.
- **若$p_{k''}=p_j$，则有“$p_1\cdots p_{k''}$”=“$p_{j-k''+1}\cdots p_j$”，**
  $next[j+1]=k''+1=next[k']+1=next[next[k]]+1$.
- **$next[j+1]=1$.**

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
模式串	a	b	c	a	a	b	b	c	a	b	c	a	a	b	d	a	b
next[j]	0	1	1	1	2	2	3	1	1	2	3	4	5	6	7	1	2

```
void get_next(SString T, int &next[])
{
    i= 1; next[1] = 0; j = 0;
    while( i<T.length){
        if(j==0 || T[i] == T[j]){
            ++i; ++j;
            next[i] = j;
        }
        else
            j = next[j];
    }
}
```

●KMP算法的时间复杂度

设主串s的长度为n，模式串t长度为m，在KMP算法中求next数组的时间复杂度为O(m)，在后面的匹配中因主串s的下标不减即不回溯，比较次数可记为n，所以KMP算法总的时间复杂度为O(n+m)。