第 17 卷第 3 期　　广 西 科 学 院 学 报　　Vol. 17, No. 3
2001 年 8 月　　Journal of Guangxi Academy of Sciences　　August 2001

KMP 扫描算法的改进
Improvement of KMP Scan Algorithm

蒋文沛
Jiang Wenpei

（广西机电职业技术学院　南宁　530007）
(Guangxi Mechanical & Electronic Engineering College, Nanning, 530007)

摘要　通过对字符串模式匹配 BF 和 KMP 算法的分析，提出了改进 KMP 扫描算法的方法，并通过对算法的复杂性进行分析，结果表明：改进后的算法 KMPA 比算法 KMP 更有效。

关键词　字符串　模式匹配　算法
中图法分类号　TP 301.6

Abstract　KMP scan algorithm is improved by means of the analysis of string pattern matching BF and KMP. The complexity analysis reveals that the improved algorithm, KMPA, is more effective than KMP.

Key words　string, pattern matching, algorithm

“算法＋数据结构＝程序”[1]，算法的好坏，决定了程序质量的高低。在算法研究或程序开发时，选用或设计一个算法总要思考下列问题：这个算法有多好？是否有更好的算法，用什么技巧获得更好的算法？寻找好的算法旨在提高程序质量。

字符串是一种线性表，它在计算机应用系统中如文本编辑、情报检索、自然语言翻译有着广泛的应用。在这些应用中常常需要在一正文字符串中检测是否有一指定的字符串。设 s 和 t 是给定的 2 个字符串，在字符串 s 中寻找等于 t 的子串的过程称为模式匹配，其中字符串 s 称为主串，字符串 t 称为模式。如果在字符串 s 中找到等于 t 的子串，则称匹配成功，否则匹配失败。比较著名的模式匹配算法有 BF 算法、KMP 算法、RK 算法和 BM 算法，本文主要对字符串模式匹配 BF 算法和 KMP 算法进行研究。

1　模式匹配的 BF 算法

模式匹配的最简单、直观的算法是 BF（Brute-Force）算法。该算法的基本思想是从主串 s 的第 start 个字符起和模式的第一个字符比较，如相等，则继续逐个比较后续字符，否则从主串的第 start＋1 个字符起再重新和模式 t 的字符比较。依此类推，直到模式 t 中的每个字符依此和主串 s 中的一个连续的字符序列相等，则称匹配成功，否则称匹配不成功。BF 算法如下[2]。

2001-05-28 收稿。

算法 1 模式匹配的 BF (Brute-Force) 算法

```
int index(char s [ ], char t [ ], int start)
{
    int i,j,m,n;
    m=len(s); n=len(t); i=start; j=0;
    while(i<m && j<n)
      if(s [i]= =t [j]) { i++;j++;}
      else {i=i-j+1;j=0;}
    if(j>=n)return(i-n);
    else return(0);
}
```

根据以上算法,对于字符串模式匹配实例 t []="abaabc"和 s []="abaabghjwabaabch"我们有:

第一趟匹配结束时:

t []="abaabc" s []="abaabghjwabaabch"

j 指针的值为 5 *i* 指针的值为 5

第二趟匹配开始时:

t []="abaabc" s []="abaabghjwabaabch"

j 指针的值为 0 *i* 指针的值为 1

上述算法的复杂性为 $O((n-m)*m)$[2],其特点是直观、简单,但涉及多次回溯,算法效率低。

2 KMP 算法

KMP 算法是由 Knuth、Morris 和 Pratt 于 1969 年夏天提出的快速串匹配算法。这个算法由两部分组成[3]:KMP 流程构造算法和 KMP 扫描算法。现将这两部分分别用 C 语言描述[2]如下。

算法 2 KMP 流程构造算法

```
get—next(char t [ ], int next [ ], int n)
{int j,k;
j=0;k=-1;next [0]=-1;
while (j<n)
  {if(k= =-1 || t [j]= =t [k]) {j++;k++;next [j]=k;}
      else k=next [k];}
}
```

数组 next [*j*] 表示当模式中第 *j* 个字符与主串中相应字符匹配失败时,在模式中需重新和主串中该字符进行比较的字符的位置,其值取决于模式本身,与主串无关。该数组称为失败链接数组[3]。

算法 3 KMP 扫描算法

```
int  kmp(char s [ ], char t [ ], int next [ ])
{ int i=0,j=0;
```

```
  m=len(s); n=len(t);
  while(i<m && j<n)
      if(j==-1 || t[i]==s[j]) { i++; j++;}
      else j=next[j];
if(j>=n)return(i-n);
else return(0);
}
```

根据以上算法，对于字符串模式匹配实例 t[]="abaabc"和 s[]="abaabghjwabaabch"我们有：

第一趟匹配结束时（j=next[j]=2）：

t[]="abaabc"　　s[]="abaabghjwabaabch"

j 指针的值为 5　　i 指针的值为 5

第二趟匹配开始时：

t[]="abaabc"　　s[]="abaabghjwabaabch"

j 指针的值为 2　　i 指针的值为 5

模式匹配 KMP 算法的复杂性为 $O(m+n)$[2]，它的引入避免了 BF 算法中频繁的回溯，普遍提高了模式匹配的工作效率。因此有些文献也称之为不回溯的字符串搜索算法[4]。

3　对 KMP 扫描算法的改进

虽然模式匹配 KMP 算法的引入避免了 BF 算法中频繁的回溯，普遍提高了模式匹配的工作效率，但是经过分析，对 KMP 算法扫描部分还可做进一步的改进，让我们来看以下思路与算法。

思路：每当某趟匹配失败时，i 指针不必回溯，而是利用已经得到的"部分匹配"结果，看看是否有必要将 i 的值进行调整，然后再将模式向右"滑动"若干位置后继续比较。

算法 4　对 KMP 扫描算法的改进算法

```
int kmpa(char s[], char t[], int next[])
{
  int i=0,j=0,k=0;
  m=len(s);
  n=len(t);
while(i<m && j<n)
    if(j==-1 || t[j]==s[j]){i++;j++;}
    else
      {k=next[j];
      if(i+j-k<=m && t[j]! =s[i+j-k]) i=i+j-k;
      j=next[j];
      }
if(j>=n)return(i-n);
else return(0);
}
```

根据以上算法，对于字符串模式匹配实例 t[]="abaabc"和 s[]="abaabghjwabaabch"我

们有：

第一趟匹配结束时（k =next [j] =2）：

t []="abaabc" s []="abaabghjwabaabch"

↑ ↑

j 指针的值为 5 i 指针的值为 5

第二趟匹配开始时（第一趟匹配结束时，$i = i + j - k = 8$）：

t []="abaabc" s []="abaabghjwabaabch"

↑ ↑

j 指针的值为 2 i 指针的值为 8

可见，算法 4 中的 i 不是保持不变，而是增加了，这就意味着加快了模式匹配的进度。

4 对算法 3 与算法 4 的复杂性分析

算法 3 的算法复杂性可分两部分来进行分析，一是扫描算法中 $s[i]$ 与 $t[j]$ 的比较次数，一是根据模式来计算失败链接数组 next [j]。计算 next [j] 与模式 t[] 的长度 n 有关，与主串无关，其算法复杂性为 $O(n)$。因为算法 3 与算法 4 涉及的计算 next [j] 的工作量相同，所以这两个算法的复杂性主要区别在是否有 "if (i+j−k) <=m && t [j]! =s [i+j−k]) i=i+j−k" 这句上，设 $p(m,n)$ 与 $q(m,n)$ 分别为算法 3、算法 4 中 $s[i]$ 与 $t[j]$ 的平均比较次数，则考虑到：(a) if 语句的两个分支出现的概率相等，平均出现次数为 $m/2$；(b) else 分支取得 $m/2$ 个 next [j] 后，i 不得增加却又投入了比较，使得在同一个 i 位置上增加了一次比较；因此：

$$p(m,n) = m + m/2 + n = 1.5m + n.$$

而算法 4 中的语句 "if (i+j−k<=m && t [j]! =s [i+j−k]) i=i+j−k" 的平均出现次数为 $m/2$，该分支语句使得 i 向右"滑动"的机会增加了 $m/4$ 次，这些机会不是使 i 保持不变，而是使 i 朝着目的地 m 前进了，亦即促进了模式匹配的进度。故有

$$q(m,n) = 1.5m - m/4 + n = 1.25m + n,$$

令 n 为定值，则有

$$\lim_{m\to\infty} \frac{q(m,n)}{p(m,n)} = \lim_{m\to\infty} \frac{1.25m + n}{1.5m + n} = 0.833,$$

可见，$p(m,n)$ 与 $q(m,n)$ 同阶，它们的复杂性都为 $O(m+n)$，但 $q(m,n)$ 的系数要比 $p(m,n)$ 的低。

5 结语

通过对算法复杂性的分析可知，KMP 扫描算法（即算法 3）及其改进算法 KMPA（即算法 4）的算法复杂性同阶，都为 $O(m+n)$，改进算法 KMPA 比 KMP 扫描算法的算法复杂性系数要低，因此它更能提高字符串模式匹配的工作效率。

参考文献

1 沃思著 N. 算法+数据结构=程序. 曹德和等译. 北京：科学出版社，1984.

2 谈春媛，江 红. 数据结构. 北京：电子工业出版社，2000. 06.

3 巴斯·萨拉著. 计算机算法、设计和分析引论. 朱洪等译. 上海：复旦大学出版社，1985.

4 张福炎等. 程序员级高级程序员级程序设计. 北京：清华大学出版社，1992. 08.

（责任编辑：黎贞崇）