一种非常快速的字符串匹配算法

罗大光，郝玉洁，刘乃琦

(电子科技大学计算机科学与工程学院 成都 610054)

【摘要】结合Karp-Rabin和Boyer-Moore字符串匹配算法的优点，提出了一种非常快速的字符串匹配算法。该算法在匹配过程中与传统的直接比较模式及正文子串不同，与KR算法一样，比较的是模式与子串对应的散列值；该算法同时吸取了BM算法的特点，能在扫描正文的过程中跳过尽可能多的字符。理论分析表明，模式串较短时，该算法在最坏情况下的时间复杂度也可以达到$O(n)$。实验表明，该算法所需时间约为KR算法的1/10。

关 键 词 匹配; 散列函数; 字符串匹配; 快速匹配

中图分类号 TP 311 文献标识码 A

A Very Fast String Search Algorithm

LUO Da-guang，HAO Yu-jie，LIU Nai-qi

(School of Computer Science and Engineering, UEST of China Chengdu 610054)

Abstract A very fast algorithm to perform pattern matching in strings was described. The proposed match algorithm combines the advantages of Karp-Rabin's algorithm and Boyer-Moore's algorithm. Instead of checking at each position of the text if the pattern occurs, it is only to check the resemblance between the contents of the string and the pattern by using a hashing function and can skips as many characters as possible. In the cases of short patterns its time complexity can theoretically reach $O(n)$ even in the worst cases. In actual experiments, the time it takes to search a string is about 1/10 of the time KR algorithm takes.

Key words pattern match; hashing function; string searching; quick search

查找用户指定的模式，即所谓字符串(模式)匹配是字符串所有运算的基础。模式匹配有多种经典的算法，如Knuth-Morris-Pratt(KMP)算法[3]，Boyer-Moore(BM)算法和Karp-Rabin(KR)算法等等[1-6]。BM算法能充分利用匹配过程中的信息跳过尽可能多的字符，是一种快速有效的模式匹配算法。KR算法是一种直观的基于某个散列函数的模式匹配算法。本文结合两者的优势，提出并实现了一种快速有效的字符串匹配算法。

1 BM算法及KR算法简介

1.1 模式匹配定义

为方便此后的描述，先给出模式匹配定义，即给定一长度为n的字符串(以下称正文)$t[0\cdots n-1]$，以及另个长度为$m(m\leq n)$的字符串(以下称模式)$p[0\cdots m-1]$，要求查找出模式在正文中首次出现或所有出现的起始位置(下标)。一旦在正文中找到一个模式，则称发生了一次匹配。本定义中的m, n，t, p的含义将适用于全文。

收稿日期：2003 – 10 – 08
作者简介：罗大光(1974 –)，男，硕士，主要从事信息安全、人工智能方面的研究.

1.2　BM算法描述和KR算法描述

BM算法的基本思想是，将正文子串$t[i\cdots i+m-1]$与模式$p[0\cdots m-1]$对齐进行自右至左的匹配，若发现不匹配，设在正文中该位置字符为c，则下次应从正文的$i+m-1+dist[c]$位置开始重新进行BM算法匹配，其效果相当于把模式、正文向右滑过一段距离$dist[c]$，即跳过$dist[c]$个字符而无需比较。$dist[c]$的计算如下：

$$dist[c]=\begin{cases} m & \text{若}c\text{不出现在}p\text{中或者}c=p[m-1],\text{但}c\neq p[j] \quad (0\le j\le m-2) \\ m-j & \text{若}c\text{出现在}p\text{中},j=\max\{j:p[j]=c,0\le j\le m-2\} \end{cases}$$

KR算法的基本思想是，定义一个散列函数，求出模式对应的散列值，在正文中只有那些与模式具有相同散列值的子串才有可能与模式匹配，因而没有必要同时考查正文中长为m的全部子串。只有当两者的散列值相等时，才逐个字符地比较模式与正文子串是否确实匹配。

2　字符串匹配算法

2.1 算法描述

字符串匹配算法的基本思想是，依次从正文中每次“筛选”一长度为m且其中每一字符均出现在模式中的子串A，然后比较A与模式的散列值，如果二者相等，且模式长度较短(当$d^m<q$时)，则一次匹配成功；否则将A与模式对齐逐个字符比较，验证是否确实匹配。重复以上过程直到结束。

“筛选”方法即对正文子串$t[i\cdots i+m-1]$进行自右至左的扫描，即“筛选”，若发现有字符不在模式中，设在正文中该字符位置为v，则下次应从正文子串$t[i+m\cdots v+m]$开始重新进行“筛选”。

设模式中出现的字符集合为Σ，$d=|\Sigma|$，取q为某个适当大的素数，并设map表用于将Σ映射为集合{0，1，2，…，$d-1$}，则模式$p[0\cdots m-1]$可表示为一个d 进制数：

$$x=map(p[0])\times d^{m-1}+map(p[1])\times d^{m-2}+\cdots+map(p[m-2])\times d+map(p[m-1])$$

对应的散列函数为$h(p)=x \bmod q$，mod为模运算。

同理，可将正文子串$t[i\cdots i+m-1]$ (i=0，1，2，…，$n-m$)表示为：

$$y_i=map(t[i])\times d^{m-1}+map(t[i+1])\times d^{m-2}+\cdots+map(t[i+m-2])\times d+map(t[i+m-1])$$

则子串$t[i+1\cdots i+m]$(与$t[i\cdots i+m-1]$相邻的下一个子串)的y_{i+1}应满足：

$$y_{i+1}=(y_i-map(t[i])\times d^{m-1})\times d+map(t[i+m])$$

同理，子串对应的散列函数形式上与模式对应的散列函数相同。下面给出本算法用C++的具体实现。

输入：$t[0\cdots n-1]$，$p[0\cdots m-1]$

输出：所求子串的位置

```
    初使化部分(构造map表，求模式散列值x，w=d^(m-1)，求mark)；
  L=0；R=m-1；i=R; next=0；
  while (R<n=
  { for(i=R；i>=L；i- -)          // “筛选”子串
    { if(map[t[i]]<0=             //t[i]是否出现在模式p中
       {   L=R + 1;   R= I + m; i=R + 1;
           if (r>=n)     return; } }
    L=R-m+1;
    if (next<R)                   //求子串散列值
    { y=0;                        //前后子串不相继时
     for(j=L; j<=R; j++=
          y = (y*d + map[t[j]])%q; }
    else                          //前后子串相继时
          y=((y-w*map[t[L-1]])*d + map[t[R]])%q;
     next=R + 1;
     if (x==y)                    //比较模式与子串的散列值
```

```
{ if (mark)    j=m;     //mark为真，不必进行字符匹配
else
    { for(j=0; j＜m; j++)                    //字符匹配
          if (t[L+j]!=p[j])     break;}
  if (j==m)              //一次匹配成功
  { cout＜＜"address: "＜＜L＜＜'\n' ;
          L=R + 1; R=R + m; continue;}}
L=R + 1; R=R + 1;}
```

2.2 初始化部分

初始化部分需要完成以下操作(以下使用的m,n,Σ，d及q的含义同上)。

2.2.1 构造一个*map*表

*map*表用于将Σ映射为集合｛0，1，2，…，d–1｝，对模式中所有未出现的字符，*map*表均映射为–1，则*map*表的构造如下：

```
for(i=0; i＜256; i ++)     map[i]=–1;
d = 0;
for(i=0; i＜m;i ++=
if (map[p[i]]＜0=
{ map[p[i]]=d; d ++; }
```

2.2.2 求*mark*

使用散列函数不可避免地会出现散列冲突，但是当$d^m<q$时，$h(p)$却是唯一的，此时$h(p)$可以看作是个d进制数。因此当$d^m<q$时，若散列值相同，则模式与子串是两个相同的串，此时不必再进行字符匹配。算法1中用*mark*为真来表示$d^m<q$成立，此处实现略去。

2.3 一个使用本算法的实例

在正文t="A simple example to complete patterns match in a string"中查找模式p="patterns"，则有n=55，m=8，Σ={p,a,t,e,r,n,s}，d=7，取q=33 554 393。因为$w=d^m=7^8$=5 764 801＜q，所以*mark*=*TRUE*。构造的*map*表见表1；执行过程见图1，图中S为自右向左已扫描字符的个数。

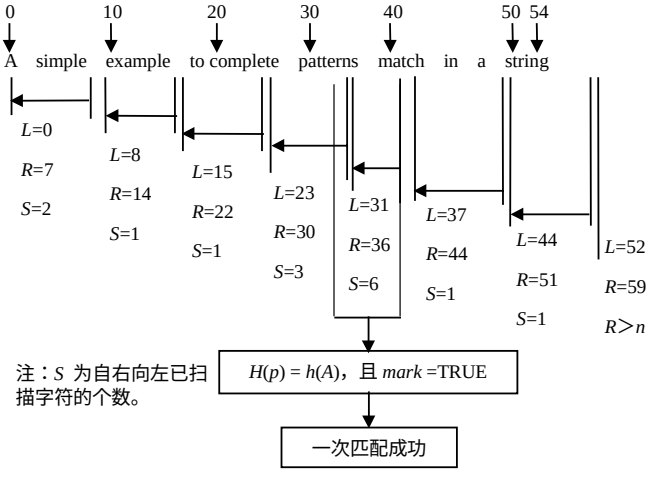

图1 本例执行过程

表1 *map*表的值 单位：ms

c	*p*	*a*	*t*	*e*	*r*	*n*	*s*	其他
map[*c*]	0	1	2	3	4	5	6	–1

3 算法复杂度分析

3.1 初始化阶段复杂度分析

构造*map*表的时间复杂度为$O(m)$，同时需要256个单位的存储空间。求解模式的散列函数、w及*mark*的时间复杂度也为$O(m)$，所以初始化阶段的时间复杂度为$O(m)$。

3.2　查找阶段复杂度分析

在“筛选”阶段，正文中的每个字符最多只被扫描一次，因此总的扫描次数≤n。由前面的描述可知，求解子串散列值所需的总计算次数≤n。而比较子串与模式之间的散列值所需的总比较次数不会超过$n-m$。另外，子串与模式进行字符匹配所需的总比较次数为m。因此，根据模式长度的不同，该算法的分析可分为两种情况，即当$d^m \geq q$时，因为需要进行字符匹配，所以在最坏情况下其时间复杂度为$O(nm)$，而实际情况下常为$O(m+n)$；当$d^m < q$时，因为不需要再进行字符匹配，所以即使在最坏情况下，其时间复杂度也为$O(n)$。因此从理论上说，如果模式较短或者说模式中出现的字符个数$d=|\Sigma|$较少，使得$d^m < q$成立时，该算法具有优越性。

4　实验比较与分析

为了对本字符串匹配算法的特性有一个感性的了解，本文设计了一个实验，以便将字符串匹配算法与KR算法及BM算法进行比较。实验随机选取一个长度为1 870 168 bytes的英文文本，从文本中选取8个不同字符长度的模式串，见表2。对每个模式串、每个算法都运行多次，取较稳定的值。实验在PⅢ500 128 M内存，Windows2000环境下进行，所有算法均用C++实现(KR算法中d取值为32，q的取值与本字符串匹配算法相同，均为33 554 393)，运行时间以ms计。

表2　查找时间表　　单位：ms

长度	KR算法	BM算法	本算法
4	352.74	52.17	52.01
10	351.27	37.23	37.16
17	353.63	33.47	32.23
35	357.69	30.99	25.11
58	352.23	29.67	27.63
100	357.39	24.25	23.79
300	354.05	23.58	26.97
800	353.03	21.70	24.30

从表2可以看出，就本文实验而言，字符串匹配算法与BM算法所需时间约为KR算法的1/10，这是因为字符串匹配算法与BM算法在查找时都可以尽可能多地跳过待查字符，而KR算法不具备这一特性。不过KR算法的性能比较稳定，对于不同长度的模式串，查找时间变化不大。字符串匹配算法与BM算法在性能上非常接近，在模式串较小时，该算法具有微弱的优势。表2中只给出了粗略的数据，实际上这两种算法的性能对模式串取值的依赖性比较大，因此表2中的数据无法准确地说明两者的差异。另外，由于实验的局限性，无法充分体现字符串匹配算法的优势。例如，从前面的理论分析知，当模式串较小时，在最坏的情况下字符串匹配算法也能在$O(n)$时间内完成查找，而这在实验中难以体现。

5　结 束 语

结合BM算法和KR算法的优点，本文提出了一个非常快速的模式匹配算法。字符串匹配算法既引入了与KR算法类似的采用散列函数的方法，又同BM算法一样能够在匹配过程中尽可能多地跳过待查正文字符。当模式串较小时，即使在最坏的情况下，该算法也有无可比拟的优势。在实际应用中，字符串匹配算法可以应用于包括英文、汉字在内的各种字符，并可以根据需要应用到各种相关领域。

参　考　文　献

[1] Knuth D E, Morris J H, Pratt V R. Fast pattern matching in strings [J]. SIAM Journal on Computing , 1977, 6(1):323-350

[2] Boyer R S,Moore J S. A fast string searching algorithm [J]. Commun ACM, 1977, 20(10):762-772

[3] Karp R M, Rabin M O. Efficient randomized pattern-matching algorithms [J]. IBM J. Res. ,1987, 31(2):249-260

[4] Sunday D M, A very fast substring search algorithm [J]. Communications of the ACM, 1990, 33(8):132-142

[5] 苏德富, 钟　诚. 计算机算法设计与分析[M]. 北京: 电子工业出版社, 2001

[6] 周培德. 算法设计与分析[M]. 北京: 机械工业出版社, 1992

编　辑　熊思亮