一种可做特殊用途的字符串匹配算法

纪福全 朱战立
(西安石油大学计算机学院，西安 710065)

摘 要 现有的字符串匹配算法按照模式串从左至右或从右至左的顺序匹配，都是直接进行比较，本文提出了一种可做特殊用途的字符串匹配算法——ZZL 算法。对于频繁使用的要匹配的主串和模式串来说，ZZL 算法的匹配速度会非常快。

关键词 字符串; 模式匹配; 算法

字符串匹配就是在一个字符串中查找模式串的一个或所有出现。字符串匹配用途很广泛。例如，在拼写检查、语言翻译、数据压缩、搜索引擎、网络入侵检测、计算机病毒特征码匹配以及DNA序列匹配等应用中，都需要进行字符串匹配。已经提出了许多字符串匹配算法。传统的有BF算法[2,3]、KMP算法[2,3]等，最近提出的有BM算法[2,4]、Sunday算法[2,3]等。本文提出一种可做特殊用途的字符串匹配算法——ZZL算法。

1 相关算法分析

字符串模式匹配的含义是：在主串 S 中，从位置 start 开始查找是否存在模式串（也称作模式串）T，如在主串 S 中查找到一个与模式串 T 相同的模式串，则模式串与主串匹配；如在主串 S 中未查找到一个与模式串 T 相同的模式串，则不匹配[1]。

首先作如下假设：

主串 S：S[1...N]，长度为 N；模式串 T：T[1...M]，长度为 M；N≥M；

1.1 BF 算法

BF（Brute Force）算法核心思想是：首先 S[1]和 T[1]比较，若相等，则再比较 S[2]和 T[2]，一直到 T[M]为止；若 S[1]和 P[1]不等，则 T 向右移动一个字符的位置，再依次进行比较。如果存在 k，1≤k≤N，且 S[k+1...k+M]=T[1...M]，则匹配成功；否则失败。该算法最坏情况下要进行 M*(N-M+1)次比较，时间复杂度为 O(M*N)。

1.2 KMP 算法

KMP（Knuth-Morris-Pratt）算法核心思想是：在发生失配时，主串不需要回溯，而是利用已经得到的“部分匹配”结果将模式串右移尽可能远的距离，继续进行比较。这里要强调的是，模式串不一定向右移动一个字符的位置，右移也不一定必须从模式串起点处重新试匹配，即模式串一次可以右移多个字符的位置，右移后可以从模式串起点后的某处开始试匹配。

假设发生失配时，S[i]≠T[j]，1≤i≤N，1≤j≤M。则下一轮比较时，S[i]应与 T[*next*[*j*]]对齐往后比较：

$$next[j]=\begin{cases}0, & \text{当 } j=1 \text{ 时}\\ Max\{k \mid 1<k<j, \text{且} T[1\ldots k-1]=T[j-k+1\ldots j-1]\\ 1, & \text{其它情况}\end{cases}$$

如 T=“abaabcac”，则

j:	1	2	3	4	5	6	7	8
Next[j]:	0	1	1	2	2	3	1	2

1.3 BM 算法

对于给出的长度为 m 的模式串 T=T1 ··· Tm 和主串 S=S1 ··· Sn，实现 BM 算法需要一个辅助数组 bm[ ]。它用字符值作为数组的下标，数组的大小依赖于可能出现的字符多少，与模式串的大小无关。对于需要进行中文关键字的匹配，需要扩充 ASCII 字符集，数组大小则为 256。对于任意 x 属于集合 Σ→{1，2，···，256}，bm[x]的值为：

$$bm[x]=\begin{cases}0; \text{若字符} x \text{不在} T \text{中}\\ j; \text{字符} x \text{在} T \text{中，其中} j=\max\{i \mid Ti=x, 1<=i<=m\}\end{cases}$$

该数组每个字符对应的项记录着该字符在模式串中最后一次出现的位置。

BM 算法思想是：假如在执行主串位置 i 起“返前”的一段与模式串 T 从右向左的字符匹配中，如果模式串 T 全部字符匹配，则匹配成功；否则需要右移，开始新的一轮匹配，假设匹配不成功发生在模式串中的位置 j，由主串匹配不成功字符 Si-m+j 查找辅助数组得到该字符在模式串 T 中的最后出现的位置值 bm[Si-m+j]。如果 bm[Si-m+j]等于零，表示字符 Si-m+j 不在模式串 T 中，则模式串跳过该字符，在该字符下一个位置对齐；如果 bm[Si-m+j]大于 j，表示这个字符在模式串中最后出现的位置在 j 的左边，则模式串 T 右移对齐字符 Si-m+j；如果 bm[Si-m+j]小于 j，表示这个字符在模式串中最后出现的位置在 j 的右边，模式串不能左移，就右移一格。移动量为 shift=max(1，m-bm[i-m+j])。

1.4 Sunday 算法

Sunday算法是Daniel M.Sunday于1990年提出的一种比BM算法搜索速度更快的算法。其核心思想是：在匹配过程中，模式串并不被要求一定要按从左向右进行比较还是从右向左进行比较，它在发现不匹配时，算法能跳过尽可能多的字符以进行下一步的匹配，从而提高了匹配效率。

假设在发生不匹配时S[i]≠T[j]，1≤i≤N，1≤j≤M。此时已经匹配的部分为u，并假设字符串u的长度为L。如图1。明显的，S[L+i+1]肯定要参加下一轮的匹配，并且T[M]至少要移动到这个位置(即模式串T至少向右移动一个字符的位置)。

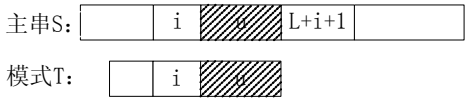

图1　Sunday算法不匹配的情况

分如下两种情况：

(1) S[L+i+1]在模式串T中没有出现。这个时候模式串T[0]移动到S[T+i+1]之后的字符的位置。如图2。

图2　Sunday算法移动的第1种情况

(2)S[L+i+1]在模式串中出现。这里S[L+i+1]从模式串T的右侧，即按T[M-1]、T[M-2]、…T[0]的次序查找。如果发现S[L+i+1]和T中的某个字符相同，则记下这个位置，记为k，1≤k≤M，且T[k]=S[L+i+1]。此时，应该把模式串T向右移动M-k个字符的位置，即移动到T[k]和S[L+i+1]对齐的位置。如图3。

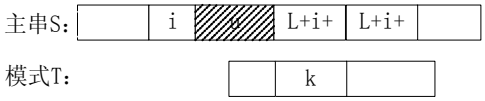

图3　Sunday算法移动的第2种情况

依次类推，如果完全匹配了，则匹配成功；否则，再进行下一轮的移动，直到主串S的最右端结束。该算法最坏情况下的时间复杂度为O(N*M)。对于短模式串的匹配问题，该算法执行速度较快。

2　ZZL 算法

现有的字符串匹配算法不论是按照模式串从左至右还是从右至左的顺序匹配，都是直接进行比较，而ZZL算法的核心思想是：首先在主串S中查找模式串T的首字母，每找到一个则将它的位置存储，然后依次提取这些位置，从这些位置开始继续匹配模式串T。对于频繁使用的要匹配的主串和模式串来说，由于预先保存了模式串在主串中的所有存储位置，所以匹配速度会非常快。

2.1 预处理

预处理主要完成查找模式串首字符在主串中的所有出现位置，并将其保存在一个数组中。

查找模式串首字符算法如下：

```
k=0;
for(i=start;i<S.length-T.length;i++)
{
    if(S.str[i]==T.str[0])
    {
        x[k]=i;
        k++;     // k为模式串首字母在主串中出现的次数
    }
}
```

2.2 匹配

在预处理的基础上，字符串匹配算法就可以从查找到的模式串在主串中的位置开始，匹配模式串首字母之后的其余部分。此时，采用BF算法即可，并可设置一个计数器，记录匹配次数。

匹配算法如下：

```
v=0;
for(m=0;m<k;m++)
{
    for(j=1;j<T.length;j++)
    {
        if(S.str[x[m]+1]==T.str[j])
        {
        v++;
        x[m]++;
        }
        else
        {
        v++;
        break;
        }
    }
}
```

3　算法性能分析及实验结果分析

3.1 算法性能分析

如果不考虑算法的预处理过程，若模式串首字母在主串中出现k次，则ZZL算法最坏情况下比较次数为k*(M-1)<k*M。如果考虑算法的预处理过程，则总的比较次数需再加上N次，即为k*M+N。

3.2 实验结果

为了评测该算法的性能，随机的抽取一段文本和模式串，并在同一台计算机上用不同的算法进行匹配。测试文本主串S="From automated teller machines and atomic clocks to

下转第85页

```
        {
            p=p->next;
        }
        ……
    }
```

某块空间数据使用结束后，数据可能不再需要保存，则将该数据块节点删除，并释放数据块所占用空间。代码如下：

```
    void Cell::delete( )
    {
          property *p，*c,*n;
          ……
           if（p->next!=NULL）
           {
               c=p;
              delete c;
               p=p->next;
           }
          ……
    }
```

如对某节点数据修改，只需找到该数据节点，修改即可。代码实现参照类中 query（）函数。

图像分析中，分析步骤可能较多，在操作过程中，很多是人工画图等，难免产生失误。由于某步操作失误，往往需要回退到某一步，动态数组链表做节点的删除、指针的回退等动作，就可很好地处理这种情况。例如图像分析操作到第 7 步时，发现在第 6 步的操作过程中产生了测量错误，或其他原因导致第 7 步操作无效，而 1 至 5 步的操作是可行有效的，并且需要用到第 5 步操作后的参数结果，此时需要回退到第 5 步，并调出第 5 步操作后的数据，重新开始第 6 步操作。程序处理时只需将指针从第 7 个节点位置重新指向第 5 个位置，并且释放第 6 和第 7 个节点所占用空间即可，即动态数组链表节点的删除。代码实现参照类中 delete()函数。

如在处理节点数据的过程中需要插入数据，代码如下：

```
 void Cell::insert()
 {    property *p,*p1;
       while (p->next!=NULL)
       {
            if (……)
            {
                p1=new property ();
                p1->next=p->next;
                p->next=p1;
           }
              ……
       }
 }
```

4 结束语

图像分析过程中，一般产生的数据量大，数据处理步骤繁杂，计算机要用较少内存，处理较多数据，且对数据的操作要及时、准确，并作存储等，动态数组在图像分析中较有实用性。如果需要存储的数据超过计算机系统块内存容量，并且需要一次性处理，不能对这些数据进行分块的话，动态数组就不能考虑使用了，可考虑多线程或多进程解决方法等。若节点数据的类型实现具有一般性，可考虑使用动态模板数组。

参考文献

[1] 钱能 .《C++程序设计教程》. 清华大学出版社，2005.9

[2] 张岳新.《Visual C++程序设计》.苏州大学出版社，2002.1

[3] 黄维通 游建波.《Visual C++面向对象与可视化程序设计》.清华大学出版社，2003.12

收稿日期:4 月 10 日 修改日期:4 月 26 日

上接第82页

mammograms and semiconductors,innumerable products and services rely in some way on technology,measurement,and standards provided by the National Institute of Standards and Technology"，模式串T="products and services"。分别用BF算法、KMP算法、BM算法、Sunday算法和ZZL算法在同一台计算机上进行匹配计算，并统计每种算法匹配时总的字符匹配次数。测试结果如表1。

表1 匹配算法实验结果

算法	BF	KMP	BM	Sunday	ZZL
一次匹配的总的字符匹配次数	116	95	108	110	23

4 结论

对于频繁使用的要匹配的主串和模式串来说，由于预先保存了模式串在主串中的所有存储位置，所以ZZL算法的匹配速度会非常快。

参考文献

1 朱战立编著. 数据结构——使用C语言（第3版）[M]. 西安：西安交通大学出版社，2004

2 王成，刘金刚. 一种改进的字符串匹配算法[J]. 计算机工程，2006，32(2):62-64

3 Christian Charras, Thierry Lecroq. Exact String Matching Algorithms[Z]. http://www-igm.univmlv.fr/~lecroq/string/

4 黄中清，汪文勇，黄鹂声. 基于 WinPcap 和改进的 BM 算法的网络信息审计系统的实现[Z]. http://www.ahcit.com/lanmuyd.asp?id=1180

收稿日期:3 月 17 日 修改日期:4 月 1 日

作者简介：纪福全(1981-)，男，硕士生，主研方向：人工智能与专家系统，神经网络；朱战立，教授、硕导