允许错误的(汉字)字符串快速检索技术*

邹旭楷 王素琴

(郑州大学计算机科学系,郑州 450052)

摘要 在计算机应用的诸多领域中都会遇到字符串相似检索问题.本文提出了一种技术,它通过应用搜索状态向量及字符—模式匹配向量,将字符串匹配比较转化为简单的整数字位运算,有效地解决了字符/汉字串的相似匹配问题.文中也给出了实现算法并分析了算法的复杂性.

关键词 文本,模式,相似匹配,搜索状态向量,字符—模式匹配向量,编辑距离.

字符串检索是指在文本 $Text=t_1\cdots t_n$ 中检索子串 $Pat=p_1\cdots p_m$(称为模式)的所有出现,求解该问题的两个最著名算法是 Boyer—Moore(BM 算法)及 Knuth—Morris—Pratt(KMP 算法),最近 Baeya—Yates R 与 Gonnet G. H 又提出了比 BM 及 KMP 算法更好的算法[1](称为 BYG 算法).然而有很多情况,其模式或文本是不确定的.例如,我们可能忘记要检索名字的确切拼法;在文本中某些名字可能拼错;在一个 DNA 分子库中找出与某一结构相似的所有分子等.允许错误的串匹配(又称相似串匹配)是指给出一定的相似标准,在 $Text$ 中找出所有与 Pat 相似的子串.通用的相似标准是编辑距离[2]:一个子串 P 离子串 Q 的距离为 k,是指对子串 P,最多进行 k 个字符的插入/删除/替代,可得到子串 Q.许多不同的相似匹配算法已被提出[3-6].最近,基于 BYG 的确切匹配算法,Sun Wu 和 Vdi Manber 提出了一种快速相似匹配算法[7](称 WM 算法).上面所有算法都是以字节为单位的字符串检索,无法适应以两字节为单位的汉字串的检索.基于 BYG 确切匹配思想,本文又提出一种快速相似匹配算法,可实现对字符串、汉字串及二者混合串的有效检索;其复杂性等于 $O(kn)$(同 WM 算法),与文本内容及模式无关;占内存附加空间 $O(64+m)$(比 WM 算法少);到目前为止是第一个快速有效的汉字串相似检索技术.

1 字符串确切匹配[1]

设模式 $Pat=p_1\cdots p_m$(m 为长度);文本 $Text=t_1\cdots t_n$(n 为长度),其中,$p_i(i=1,\cdots,m)$,$t_j(j=1,\cdots,n)$都取自某个字母表 Σ.

引入搜索状态向量 S:S 有 m 个分量,S_j 表示在 $Text$ 当前位置 j 其前的字符与 Pat 的任意前缀的匹配情况.记 $S_j[i]$为搜索状态向量的第 i 个分量,如果 $p_1\cdots p_i=t_{j-i+1}\cdots t_j$,有

* 本文 1993—05—17 定稿

作者邹旭楷,31 岁,副教授,主要研究领域为算法和复杂性,数据库与知识库.王素琴,女,30 岁,讲师,主要研究领域为算法与复杂性,软件工程.

本文通讯联系人:邹旭楷,郑州 450052,郑州大学计算机科学系

$S_j[i]=0$,否则 $S_j[i]=1$,显然 $S_j[m]=0$ 表示存在一个确切匹配($t_{j-m+1}\cdots t_j$).

引入字符-模式匹配向量 MT:对任意 $a\in\Sigma$,定义 $MT(a)=\langle MT_1(a),\cdots,MT_i(a),\cdots,MT_m(a)\rangle$,如果 $a=p_i$,则 $MT_i(a)=0$,否则 $MT_i(a)=1$. MT 可预先根据 Σ 及 Pat 求出.假定已求出 S_j,前进一个字符到 t_{j+1},S_{j+1} 可由 S_j 与 MT 求得:即 $S_{j+1}[i]=S_j[i-1]|MT_i(t_{j+1})$,其中,|表示"按位或".约定:对所有 $j(0\leqslant j\leqslant n)S_j[0]=0$.

S_j 可表示为共有 m 位的 0,1 位串,进而该位串可看作是一个整数,如果 S_j 的最低(右)位为 0,则找到一个匹配.同理 MT 可表示为整数,则 S_j 到 S_{j+1}的转换可表示为 $S_{j+1}=S_j\gg 1|MT(t_{j+1})$,其中$\gg 1$ 表示右移 1 位.

当 $Text$ 和 Pat 含有汉字时,为防止误匹配,遇到汉字时,要前进 2 个字节.即假定已计算出 S_j,前进 1 个字符到 t_{j+1},如果发现 $t_{j+1}t_{j+2}$是 1 个汉字,则继续前进 1 个字符到 t_{j+2},计算 S_{j+2},$S_{j+2}=(S_j\gg 1|MT(t_{j+1}))\gg 1|MT(t_{j+2})$.

2 字符(汉字)串相似匹配

定义. 一个符号是指一个字符(单字节)或一个汉字(双字节).

以下分字符与汉字两种情况讨论错 1 个或 k 个符号的相似匹配,其中括在括号里的是对汉字而言的.

令 S、MT 同上,从 S_j 到 S_{j+1}(或 S_{j+2})转换同上.

2.1 允许 1 个插入错的匹配

设 S^1,S_j^1 表示处理完 t_j 后所有允许 1 个插入错的匹配情况,即:如果 $p_1\cdots p_i$ 与 t_j 向前存在最多插入 1 个符号的匹配,则 $S_j^1[i]=0$,否则 $S_j^1[i]=1$,显然:$S_j^1[m]=0$ 表示存在一个最多插入 1 个符号的匹配,从 S_j^1 到 S_{j+1}^1(或 S_{j+2}^1)的转换方法如下:(分 2 种情况)

(1)到 t_j,与 $p_1\cdots p_i$ 已确切匹配,这相当于插入 t_{j+1}(或 $t_{j+1}t_{j+2}$)后产生允许 1 个插入错的匹配,则 $S_{j+1}^1[i]=S_j[i]$(或 $S_{j+2}^1[i]=S_j[i]$),即 $S_{j+1}^1=S_j$(或 $S_{j+2}^1=S_j$).

(2)到 t_j,与 $p_1\cdots p_{i-1}$有允许 1 个插入错的匹配且 $t_{j+1}=p_i$(或 $t_{j+1}t_{j+2}=p_ip_{i+1}$),则 $S_{j+1}^1[i]=S_j^1[i-1]|MT_i(t_{j+1})$(或 $S_{j+2}^1[i+1]=S_j^1[i-1]|MT_i(t_{j+1})|MT_{i+1}(t_{j+2})$),即 $S_{j+1}^1=S_j^1\gg 1|MT(t_{j+1})$(或 $S_{j+2}^1=(S_j^1\gg 1|MT(t_{j+1}))\gg 1|MT(t_{j+2})$).

说明:在插入匹配的情况下,有初值 $S_0^1=1\cdots 1$(m 个 1).

2.2 允许 1 个删除错的匹配

设 S^1,S_j^1 表示处理完 t_j 后所有允许 1 个删除错的匹配情况,即:如果 $p_1\cdots p_i$ 与 t_j 向前存在最多删除 1 个符号的匹配,则 $S_j^1[i]=0$,否则 $S_j^1[i]=1$,显然:$S_j^1[m]=0$ 表示存在一个最多删除 1 个符号的匹配,从 S_j^1 到 S_{j+1}^1(或 S_{j+2}^1)的转换方法如下:

(1)到 t_{j+1}(或 t_{j+2}),与 $p_1\cdots p_{i-1}$已确切匹配,这相当于删除 p_i(或 p_ip_{i+1})后产生允许 1 个删除错的匹配,则 $S_{j+1}^1[i]=S_{j+1}[i-1]$(或 $S_{j+2}^1[i+1]=S_{j+2}[i-1]$),即 $S_{j+1}^1=S_{j+1}\gg 1$(或 $S_{j+2}^1=S_{j+2}\gg 2$).

(2)到 t_j,与 $p_1\cdots p_{i-1}$有允许 1 个删除错的匹配且 $t_{j+1}=p_i$(或 $t_{j+1}t_{j+2}=p_ip_{i+1}$),则 $S_{j+1}^1[i]=S_j^1[i-1]|MT_i(t_{j+1})$ (或 $S_{j+2}^1[i+1]=S_j^1[i-1]|MT_i(t_{j+1})|MT_{i+1}(t_{j+2})$),即 $S_{j+1}^1=S_j^1\gg 1|MT(t_{j+1})$(或 $S_{j+2}^1=(S_j^1\gg 1|MT(t_{j+1}))\gg 1|MT(t_{j+2})$).

说明:在删除匹配的情况下,有初值 $S_0^1=01\cdots1$ ($m-1$ 个 1).

2.3 允许 1 个替代错的匹配

设 S^1,S_j^1 表示处理完 t_j 后所有允许 1 个替代错的匹配情况,即:如果 $p_1\cdots p_i$ 与 t_j 向前存在最多替代 1 个符号的匹配,则 $S_j^1[i]=0$,否则 $S_j^1[i]=1$. 显然:$S_j^1[m]=0$ 表示存在一个最多替代 1 个符号的匹配,从 S_j^1 到 S_{j+1}^1(或 S_{j+2}^1)的转换如下:

(1)到 t_j,与 $p_1\cdots p_{i-1}$ 已确切匹配,这相当于用 t_{j+1} 替代 p_i(或 $t_{j+1}t_{j+2}$ 替代 p_ip_{i+1})后产生允许 1 个替代错的匹配,则 $S_{j+1}^1[i]=S_j[i-1]$(或 $S_{j+2}^1[i+1]=S_j[i-1]$),即 $S_{j+1}^1=S_j\gg1$(或 $S_{j+2}^1=S_j\gg2$).

(2)到 t_j,与 $p_1\cdots p_{i-1}$ 有允许 1 个替代错的匹配且 $t_{j+1}=p_i$(或 $t_{j+1}t_{j+2}=p_ip_{i+1}$),则 $S_{j+1}^1[i]=S_j^1[i-1]|MT_i(t_{j+1})$(或 $S_{j+2}^1[i+1]=S_j^1[i-1]|MT_i(t_{j+1})|MT_{i+1}(t_{j+2})$),即 $S_{j+1}^1=S_j^1\gg1\ |MT(t_{j+1})$(或 $S_{j+2}^1=(S_j^1\gg1|MT(t_{j+1}))\gg1|MT(t_{j+2})$).

说明:在替代匹配的情况下,有初值 $S_0^1=1\cdots1$(m 个 1).

2.4 允许 k 个(插入/删除/替代)错误的匹配

记 S 为 S^0,设 $S^1,\cdots,S^k,S_j^d(1\leqslant d\leqslant k)$表示处理完 t_j 后所有允许最多有 d 个错误的匹配情况,即:如果 $p_1\cdots p_i$ 与 t_j 向前存在最多有 d 个错误的匹配,则 $S_j^d[i]=0$,否则 $S_j^d[i]=1$,显然:$S_j^d[m]=0$ 表示存在一个最多 d 个错误的匹配,从 S_j^d 到 S_{j+1}^d(或 S_{j+2}^d)转换方法如下:(分 4 种情况讨论)

(1)到 t_j,与 $p_1\cdots p_{i-1}$ 有最多 d 个错误的匹配且 $t_{j+1}=p_i$(或 $t_{j+1}t_{j+2}=p_ip_{i+1}$),则 $S_{j+1}^d[i]=S_j^d[i-1]|MT_i(t_{j+1})$(或 $S_{j+2}^d[i+1]=S_j^d[i-1]|MT_i(t_{j+1})|MT_{i+1}(t_{j+2})$),即 $S_{j+1}^d=S_j^d\gg1|MT(t_{j+1})$(或 $S_{j+2}^d=(S_j^d\gg1|MT(t_{j+1}))\gg1|MT(t_{j+2})$).

(2)到 t_j,与 $p_1\cdots p_{i-1}$ 有最多 $d-1$ 个错误的匹配,这相当于用 t_{j+1} 替代 p_i(或 $t_{j+1}t_{j+2}$ 替代 p_ip_{i+1}),则 $S_{j+1}^d[i]=S_j^{d-1}[i-1]$(或 $S_{j+2}^d[i+1]=S_j^{d-1}[i-1]$),即 $S_{j+1}^d=S_j^{d-1}\gg1$(或 $S_{j+2}^d=S_j^{d-1}\gg2$).

(3)到 t_j,与 $p_1\cdots p_i$ 有最多 $d-1$ 个错误的匹配,这相当于插入 t_{j+1}(或 $t_{j+1}t_{j+2}$),则 $S_{j+1}^d[i]=S_j^{d-1}[i]$(或 $S_{j+2}^d[i]=S_j^{d-1}[i]$),即 $S_{j+1}^d=S_j^{d-1}$(或 $S_{j+2}^d=S_j^{d-1}$).

(4)到 t_{j+1}(或 t_{j+2}),与 $p_1\cdots p_{i-1}$ 有最多 $d-1$ 个错误的匹配,这相当于删除 p_i(或 p_ip_{i+1}),则 $S_{j+1}^d[i]=S_{j+1}^{d-1}[i-1]$(或 $S_{j+2}^d[i+1]=S_{j+2}^{d-1}[i-1]$),即 $S_{j+1}^d=S_{j+1}^{d-1}\gg1$(或 $S_{j+2}^d=S_{j+2}^{d-1}\gg2$).

归纳之:$S_{j+1}^d=(S_j^d\gg1|MT(t_{j+1}))\&S_j^{d-1}\gg1\&S_j^{d-1}\&S_{j+1}^{d-1}\gg1$

(或 $S_{j+2}^d=((S_j^d\gg1|MT(t_{j+1}))\gg1|MT(t_{j+2}))\&S_j^{d-1}\gg2\&S_j^{d-1}\&S_{j+2}^{d-1}\gg2$).

初值:$S_0^d=0\cdots01\cdots1$ ($0\leqslant d\leqslant k$,d 个 0,$m-d$ 个 1).

3 实现算法及复杂性分析

允许错误的字符(汉字)串快速检索算法.

```
#define WORDLGTH 32
#define MAXPAT 20
#define EOS 0
#define MAXSYMBOL 256
```

```
#define MAXERROR 10
strsearch(unsigned char * text,unsigned char * pattern,int k)
{
 unsigned long searchstate[MAXERROR+1],mt[MAXPAT+1],matches,mask;
 char inter_mt[MAXSYMBOL];
 int d,patlgth;
 patlgth=strlen(pattern);
 mask=~(~0<<patlgth);
 /* ---- next forming char—pattern match vector MT ------ */
 for(d=0;d<MAXSYMBOL;d++) inter_mt[d]=0;
 for (d=1; *pattern! =EOS;pattern++)
    if (! inter_mt[ *pattern]){inter_mt[ *pattern]=d;d++;};
 for(d=0;d<=MAXPAT;d++) mt[d]=~0&mask;
 pattern-=patlgth;
 for (d=(1<<(patlgth-1)); *pattern! =EOS;d>>=1,pattern++)
    mt[inter_mt[ *pattern]]&=~d;
 /* ----next forming initial states 0…01…1
            (d zeors and (patlgth-d) ones)              ------ */
 searchstate[0]=~0&mask;
 for(d=0;d<k;d++) searchstate[d+1]=searchstate[d]&(~(1<<(patlgth-d)));
 matches=0;
 while( *text!=EOS){
    unsigned long temp1,temp2;
    temp1=searchstate[0];
    searchstate[0]=searchstate[0]>>1|mt[inter_mt[ *text]];
    if ( *text<0xa0)          /* is a char */
      for (d=1;d<=k;d++){
         temp2=searchstate[d];
         searchstate[d]=(searchstate[d]>>1|mt[inter_mt[ *text]])/
                        &(temp1>>1)&temp1&(searchstate[d-1]>>1);
         temp1=temp2;}
      else{ /* is a chinese character */
        unsigned char tempchar= *text;
        text++;searchstate[0]=searchstate[0]>>1|mt[inter_mt[ *text]];
        for (d=1;d<=k;d++){
           temp2=searchstate[d];
           searchstate[d]=(searchstate[d]>>1|mt[inter_mt[tempchar]])>>1/
                          |mt[inter_mt[ *text]]&(temp1>>2)&temp1&searchstate[d-1]>>2;
           temp1=temp2;};
             };
        text++;
        if (!(searchstate[k]&1)) matches++;
        /* insert processing code here if needed */
      }
 }
```

复杂性分析. 算法预先根据 Pat 及 Σ 生成 MT,其处理时间为 $O(m+|\Sigma|)$; 从头到尾对 $Text$ 扫描需 n 次,每次需 k 次计算,因此串检索的总时间为 $O(kn)$,仅与文本长度有关,而与模式及文本内容无关,也即最好、最坏及平均检索时间均为 $O(kn)$. 为了存放 MT,需 m 个字的附加空间,另外引入字符数组 $inter\_mt$,大小为256,将 Pat 中的 $p_1,\cdots,p_m$ 影射到1,…,m,而将非 Pat 中的所有字符影射到0. $inter\_mt$ 占的空间为256/4=64,因此总的附加空

间为64+m 个字.

4 总 结

上面给出了一种实用的含汉字的字符串相似检索的快速技术及实现算法,该技术具有简明性(原理和算法很简单)、高效性(对文本一次性扫描且用简单算术运算代替复杂的比较运算)、无关性(与模式及文本内容无关,时间复杂性仅与文本长度有关)、通用性(可适合于任何类型字符/汉字串的检索).用 C 语言写出的程序已在 AST 386微机上调试通过.

参考文献

1 Baeza_yates R, Gonnet G H. A new approach to text searching. Comm. ACM, 1992,35(10):74—82.

2 Levenshtein V I. Binary codes capable of correcting deletions, insertions and reversals. Sov. Phys. Dokl, 1966: 707—710.

3 Chang W I, Lawler E L. Approximate string matching in sublinear expected time. FOCS, 1990:116—124.

4 Galil Z, Park K. An improved algorithm for approximate string matching. SIAM J. Comput., 1990,19(12):989—999.

5 Landau G M, Vishkin U. Fast string matching with k differences. J. Comput. Syst. Sci., 1989,37:63—78.

6 Ukkonen E. Finding approximate patterns in strings. J. Algor., 1985,6:132—137.

7 Wu S, Manber U. Fast text searching allowing errors. Comm. ACM, 1992,35(10):83—91.

A FAST APPROACH TO (CHAR/CHINESE CHARACTER) TEXT SEARCHING ALLOWING ERRORS

Zou Xukai and Wang Suqin

(*Department of Computer Science, Zhengzhou University, Zhengzhou* 450052)

Abstract　There exists the problem of approximate text searching in many fields of computer application. This paper offers an approach which, by using searching state vector and char—pattern matching vector, changes text_pattern matching from comparison to simple integer bit operation and solves effectively the problem of char/Chinese character string approximate matching. The implementation algorithm and the analysis of the algorithm are also provided.

Key words　Text, pattern, approximate matching, searching state vector, char—pattern matching vector, edit distance.