文献系统定题检索的设计

黄正瑞　（暨南大学）

摘要:定题检索是文献检索系统的主要服务方式之一,本文介绍作者在设计文献检索系统 *SOCMI* 时,如何实现定题检索服务,并给出几个有效的算法。当然,其中的主要算法也适用于追溯检索。

定题检索是特定单位或正在从事某一特定课题研究的科研人员所欢迎的一种文献检索方式,因为他们在一个时期内的检索要求往往是固定的,可以根据他们的要求预先编写提问逻辑式,在文献检索系统中建立户头,以便定期按户头进行检索,并将结果及时打印分发给情报的用户。

本文介绍我们在设计文献检索系统 SOCMI 时,实现定题检索服务方法。当然,其中的主要算法也适用于追溯检索。

一、用户检索要求的存储及定题检索算法

为了实现定题检索,首先要设法在文献检索系统内保存用户的检索要求,即建立定题检索的户头。检索要求一般用逻辑表达式的形式表达,例如

(C 语言+PASCAL 语言)＊程序设计−题解

表示要检索有关 C 语言或 PASCAL 语言程序设计方面的文献资料,但其中不含题解一类的文献。表达式中的逻辑运算将一般使用+、＊和−,其中+为″或″,表示文献若满足其中一个条件则为命中文献;＊为″与″,表示只有同时满足两个条件,即同时含有两个检索词,才为命中文献;−为″与非″,如 A−B 即为 A AND NOT B,表示含有 A 而不含有 B 检索词的文献。

在文献检索系统中,我们设计定题检索服务文件 sdif,该文件以字符串形式存储用户的检索要求的逻辑表达式以及用户的其他信息,一个定题检索用户为一个记录。在 C 语言程序中,定题检索用户文件的记录结构如下:

```
struct   user{
   char   id[4];        /＊用户标识     ＊/
   char   name[9];      /＊用户姓名     ＊/
   char   addr[21];     /＊用户地址     ＊/
   char   tel[15];      /＊用户电话     ＊/
   char   exp[90];      /＊提问逻辑式   ＊/
}
```

我们在文献检索系统 SOCMI 中设计了如下的定题检索算法,该算法建立在倒排文档数据结构的基础上:

1.按用户标识从定题检索用户文件 sdif 中检出特定用户的检索要求以及其他涉及用户的信息。

2.顺序扫描提问逻辑式,每遇到一个提问项,则检索倒排文档,并各自作成检索结果的文献集合,进而将提问逻辑式转换成集合运算表达式。

3.将集合运算表达式转换成逆波兰表达式。

4.按逆波兰表达式的顺序进行集合运算,得结果文献集合。

5.按结果文献集合所提供的文献地址输出命中文献。

下面讨论上述求解过程的主要实现方法。

二、求文献集合并将提问逻辑式转换为集合运算式

求文献集合是对倒排文件进行的,方法是从左到右顺序扫描提问逻辑式,识别每一个提问项,每当找到一个提问项,就调用检索过程求出满足它的文献集合,并按序用一个字母作为集合号,用集合号置换提问逻辑式中的提问项,但注意文献集合中的元素不是文献本身,而是文献号或文献存放的地址。这样经过一遍扫描之后,提问逻辑式就变成了文献集合运算表达式。例如,提问逻辑式(C 语言+PASCAL 语言)＊程序设计−题解,经过上述处理之后变换为集合运算式:(A+B)＊(C−D)。

© 中国科学院软件研究所 http://www.c-s-a.org.cn

下面的C语言函数setexp()定义了上述处理过程。

```
char s[40];
void setexp(char command[]){
  register int i,j,k;char word[13],c=65;
  j=k=0;setx=0;
  for (i=0;i< strlen(command)+1;i++){
    switch (command[i]){
      case'(':s[j++]=command[i];break;
      case'/0/':if (command[i-1]==')')break;
      case')':
      case'+':
      case'-':
      case'*':if(k>){
              word[k]='/0';find(word);
              s[j++]=(char)c++;k=0;}
            s[j++]=command[i];
            break;
      default:word[k++]=command[i];
    }  }s[j]='/0';
}
```

函数setexp的输入是存于command中的提问逻辑式,转换后的集合运算表达式存在字符变量S中。函数setexp逐个字符地扫描command,并按以下规则处理:

1.若当前字符为(,则送入S;

2。若当前字符为)、+、-、*,如果存放提问项的字符数组WORD非空,则先调用查询函数FIND求出相应的文献集合,然后将用大写字母表示的文献集合号和当前字符送入S中;否则只需将当前字符送入S即可,这种情况出现在右括号之后紧跟运算符的场合。

3.若当前字符既不是括号又不是运算符,则它必是提问项的内容,应将它送入存放提问项的字符数组WORD中。

FIND是求文献集合的检索函数,它通过倒排文档进行检索,所以它依赖于倒排索引的数据结构。为了节省内存开销,我们将检索结果暂存在一个磁盘文件上,并设一个数组分别记下各个文献集合在文件中的开始地址和检中文献的个数。

三、集合运算表达式的逆波兰变换

集合表达式的求解如同算术表达式求解一样,先将集合运算表达式转换成逆波兰表示,例如将(A+B)*C-D转换为逆波表达式AB+CD-*。然后从左到右扫描逆波兰表达式查找算子并对它求值。为此需设置专门的数据结构、规定算子的优先级以及设计变换算法:

1.数据结构。为了将集合表达式转换为逆波兰式,需要设置以下三个工作区:

(1)算子栈。暂时存放不能送往逆波兰输出区的算子。在C语言程序中可以设计一个一维字符数组stack作为算子栈。

(2)逆波兰表示输出区。用来存放由集合运算式转换来的逆波兰表达式。在C语言程序中,可以定义一个二维字符数组作为输出区。例如设为poland[30] [2],其中poland[i] [0]存放特征,约定0表示文献集合,用1表示算子;poland[i] [1]存放集合或算子。

2.算子的优先级。文献检索系统应规定算子的优先级数,如+的优先级数为2,*为3,-为4,即-的优先级最高。另外,为了比较也规定了括号的优先级数为1。

3.变换算法规则。逆波兰变换过程需要从左到右逐个扫描集合运算式的字符,并按以下规则将文献集合号和算子送入逆波兰输出区poland:

(1)若当前字符是文献集合号,则无条件地将它送入逆波兰输出区poland[i] [1],并在元素poland[i] [0]置标志0;

(2)若当前字符是算子*、+、-,则当算子栈非空时比较它与算子栈顶元素的优先级,如果当前算子的优先级较高,表示在此之前的集合运算暂不能进行,因此必须将当前算子压入栈内;反之,如果栈顶元素的算子优先级高,则将栈顶的算子移入逆波兰输出区poland[i][1]中,并在poland[i][0]中置标志1,然后,再与新的栈顶算子进行比较。但如果算子栈为空,则当前字符无条件压入栈内。

(3)若当前字符是(,则表示其后还有内容,暂时还不能组成运算,所以应无条件地将它压入栈内。

(4)若当前字符是),则表示它及与其配对的(之间的所有运算都可以执行了,这时应从算子栈中按后进先出的次序将这对括号内的所有算子依次移入逆波兰输出区,而(本身也从栈内清除。

(5)若当前字符为集合式的结束标志,则将留在算子

© 中国科学院软件研究所 http://www.c-s-a.org.cn

栈中的所有算子依后进先出次序移入逆波兰区,并终止变换过程。

例如,对于集合运算式(A+B)＊C−D,按上述规则变换之后,逆波兰输出区如右图所示。注意算子栈只是一个临时工作区而已,变换过程一旦结束,它就失去存在的意义。而逆波兰输出区则不同,它的结果正是下一步集合运算的基础,所以它要定义为一个全局数组。

0	A
0	B
1	+
0	C
0	D
1	−
1	*

下面是用C语言定义的逆波兰变换函数change( )。

```
void change (char command[]){
 char stack[20],poland[30][2];
 int i,j,p,order[8]={1,1,3,2,0,4,0,0};
 i=top=0;
 for(j=0;j<strlen(command)+1;j++){
   switch(command[j]){
     case'(':stack[top++]=command[j];break;
     case')':while(stack[--top]!='('){
          poland[i][0]='1';
          poland[i++][1]=stack[top];
        }break;
   case'+':
   case'-':
   case'*':if (top==0) stack[top++]=command[j];
       else while (top>0){
         if(order [command[j]-40]>order[stack
                                  [top-1]-40]){
           stack[top++]=command[j];break;}
         else{ poland[i][0]='1';
           poland[i++][1]=stack[--top];
         }
     }break;
   case'^':top--;
         while(top>=0 && stack[top]!='('){
           poland[i][0]='1';
           poland[i++][1]=stack[top--];
         }break;
   default:poland[i][0]='0';
           poland[i++][1]=(char)(command[j]);break;
  }
 }
}
```

在程序中用一维字符组command从调用处接收文献集合运算式,定义stack为算子栈,整型数组order存放各种算子的优先级。因为(、)、＊、+、−的ASCII码分别为40、41、42、43和45,减去40则分别为0、1、2、3和5。我们正好利用这个特点将(、)、＊等算子的优先级分别存放在order的第0、1、2等元素处,即得order[6]={1,1,3,2,0,4}。这样,如果command[j]是一个算子,那么其优先级正好是order[command[j]−40],而栈顶元素的优先级则为order[stack[top−1]−40]。这种处理使程序十分简练。

四、集合运算

集合运算仍在逆波兰输出区上进行,其过程是:顺序扫描逆波兰输出区以寻找算子每当找到一个算子,就上溯找出两运算集合号,并据此从存放文献集合的临时文件中读出各自的文献号(或地址),按算子的含义对他们执行集合运算,运算结果产生的新文献集合仍存入临时文件中。但在逆波兰输出区的当前算子处存入结果集合号。并将标志由1改为0。此外,将参与运算的两个文献集合均作删除标志,则置poland[j][0]=″＊″。

集合运算的一个技术问题是工作区的设置策略,日本学者福岛曾建议实现对倒排文档的检索时,应该在内存中设置7个左右的工作区,同时存放参与运算的集合中的文献号。只要灵活调度,及时释放不必占用的工作区,这对绝大多数提问是完全能满足需要的。但是,对倒排文档的检索,每个工作区必须大到能够放下倒排索引中任一检索词所包含的所有文献号,这对一个文献和稍多的系统就是一个不容忽略的问题。因为,假设检索词中包含文献最多为10000个,每个文献号用4个字节存储,那么7个工作区就要占用内存270KB以上,这是一个不小的数字。因此,我们在设计时,将中间结果都存入到磁盘文件中,而只在内存中开辟两个动态分配空间的工作区。这两个工作区不仅存放参与运算的文献集合号,而且结果集合也存放在第一个工作区中。这样不仅应付自如,而且由于C语言能成批读写磁盘,对磁盘文件读写文献集合的速度的影响是很小的。

本文介绍的定题检索服务,以及上述各种算法均已在笔者设计的一个具有数十万文献记录的文献检索系统中实现,收到较好的效果。

© 中国科学院软件研究所 http://www.c-s-a.org.cn