一种 Trie 结构*

黄竞伟　　　　戴大为

(武汉水利电力大学,武汉 430072)　　(武汉大学,武汉 430072)

摘要　本文描述了一种 Trie 结构,给出了这种 Trie 结构的插入,查找算法.查找算法的时间复杂度为 $O(\log_n K)$.与以前的工作相比,这是一个改进.本文也给出了将 Trie 结构存放在一维数组后的查找算法.

关键词　Trie 结构*,查找,压缩*.

在计算机科学的许多领域里,我们经常遇到下列表查找问题:给定一个有 N 个关键字的关键字空间 S 以及一个初始状态为空的表 T,对于 S 中的关键字 K,我们希望在表 T 上执行下面二种操作:

Insert(K):将关键字 K 以及相关信息(若有的话)添加到表 T 中.

Search(K):在表 T 中查找关键字 K,若 K 在表 T 中,则获取与 K 相关的信息.

本文将考虑这个问题的静态情形,即所有的插入操作在任一查找操作开始之前即已结束的情形.不失普遍性,我们可以假定关键字空间 $S=\{0,1,\cdots,N-1\}$.

Tarjan 在[1]中描述了一种利用 Trie 结构解决上述问题的方法.本文的 Trie 结构与 Tarjan 的 Trie 结构有下列不同之处:其一是增加了一个数据域.应注意的是,有了这个数据域,结点中可以不必存储关键字,所以并没有增加额外的存储花费,因而这个"增加"只是概念上的.其二是本文中的 Trie 结构是一个堆.对于插入、查找关键字 K,我们的插入算法的时间复杂度和 Tarjan 的一样,都为 $O(\log_n N)$,但我们的查找算法的时间复杂度为 $O(\log_n K)$,从而改进了 Tarjan 的查找算法的时间复杂度 $O(\log_n N)$.

1　Trie 结构及其插入、查找算法

设 $F=\{a_1,a_2,\cdots,a_n\}$ 为关键字空间的一个子集合,表示 F 的一个静态 Trie 结构是一棵 m 叉树,m 是一个预先取定的正整数,在本文中,我们取 $m=n$.树的每个结点有二个数据域,一个域用来存储 F 中的关键字,另一个域用来存储该关键字被 n^t 除所得的商,其中 t 为根结点到该结点的路径长.此商用于 Trie 结构插入算法.还有一个具有 n 个分量的指针数组指向它的各个子结点.于是,可以给出如下的形式说明:

---

* 本文 1991-09-30 收到,1992-03-11 定稿

本文受国家自然科学资金资助.作者黄竞伟,38 岁,副教授,主要研究领域为数据结构,算法设计与分析.戴大为,57 岁,教授,主要研究领域为算法设计与分析,计算语言学.

本文通讯联系人:黄竞伟,武汉 430072,武汉水利电力大学

```
Type trie = ^ trienode;
      trienode = record
                  Key  :integer;
                  Quot :integer;
                  link :array[1…n]of trie;
                  end;
```

例如，表示集合 F={121,120,102,211,210,212}的一个静态 Trie 结构如图 1 所示.

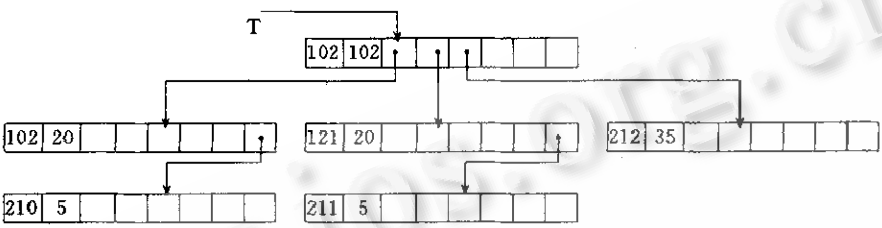

图1

它由下列插入算法生成.这一插入算法的关键是，它保证生成的 Trie 树始终是一个堆.

算法 1：Trie 结构插入算法

```
Procedure trieinsert( Var T:trie ; Var K:integer );
{将关键字 K 插入 Trie 结构 T 中}
begin
  if T=nil then
  begin
    new(T); T^.key:=K; T^.quot:=K; for i:=1 to n do T^.link[i]:=nil ;
  end
  else begin
        p:=T ; q:=nil ; x:=K ;
        while p<>nil do
          begin
           if K< p^.key then
            begin
             y:=p^.key; z:=p^.quot; p^.key:=K; p^.quot:=x; K:=y; x:=z;
            end
           r:=x mod n ; x:=x div n ; q:=p ; p:=p^.link[r+1];
          end;
          new(s) ; s^.key:=K ; s^.quot:=x ;
          for i:=1 to n do s^.link[i]:=nil; q^.link[r+1]:=s;
        end
end;
```

在算法1中，为简单起见，当插入关键字 K 时，我们假定 K 不在 Trie 结构中.

定理1. 设 K 为利用算法1建立起来的 Trie 结构中任一关键字，那么从根结点到存储关键字 K 的路径长度不超过$[\log_n K]+1$.

证明：设从根结点到存储关键字 K 的结点的路径上各结点的关键字依次是 $a_1, a_2, \cdots, a_s$，由算法1知

$$a_1 < a_2 < \cdots < a_s < K$$

设　$K = \lambda_t n^t + \cdots + \lambda_1 n + \lambda_0, \lambda_t \neq 0$

$a_i = \lambda_{v_i}^{(i)} n^{v_i} + \cdots + \lambda_1^{(i)} n + \lambda_0^{(i)}$,

$\lambda_{v_i}^{(i)} \neq 0$, $i = 1, 2, \cdots, s$.

图2

由于 $a_1, a_2, \cdots, a_s, K$ 在同一路径上，由算法1知：

$\lambda_0 = \lambda_0^{(s)} = \cdots = \lambda_0^{(2)} = \lambda_0^{(1)}$

$\lambda_1 = \lambda_1^{(s)} = \cdots = \lambda_1^{(2)}$

……

$\lambda_{s-1} = \lambda_{s-1}^{(s)}$

因为 $\lambda_0^{(1)} = \lambda_0^{(2)}$，且 $a_2 > a_1$，故有 $v_2 \geqslant 1$，又 $\lambda_0^{(3)} = \lambda_0^{(2)}, \lambda_1^{(3)} = \lambda_1^{(2)}$，且 $a_3 > a_2$，故有 $v_3 \geqslant 2$，同理可得 $v_s \geqslant s-1$，由 $\lambda_0 = \lambda^{(s)}0, \cdots, \lambda_{s-1} = \lambda_{s-1}^{(s)}$，且 $k > a_s$ 便得 $t \geqslant s$，即有 $s \leqslant t = [\log_n K]$，故从根结点到存储关键字 K 的结点的路径的长度不超过 $s+1 \leqslant [\log_n K]+1$.

定理2. 算法1的时间复杂度为 $O(n+\log_n N)$.

为了在上述 Trie 结构中查找关键字 K，我们首先考察 Trie 结构的根结点，若该结点存储的关键字不是 K，则用 n 去除 K 得商 k_1，余数为 r_1，然后沿着根结点的第 r_1+1 个指针分量到达一个新的结点，我们再去考察这个结点存储的关键字，若仍不是 K，则再用 n 去除 k_1 得商 k_2，余数为 r_2，在沿着这个新结点的第 r_2+1 个指针分量继续考查下一个结点继续这个过程，将达到一个结点，它存储的关键字不是等于 K 就是大于 K，或虽小于 K 但下一个待遵循的指针为空. 在第一种情形，查找成功，不然查找失败.

算法2：Trie 结构查找算法

```
procedure triesearch(T:trie ; K:integer; Var p:trie);
 {在 trie 结构 T 中查找关键字 k，若查找成功，则 p 指向存储关键字 K 的结点，若查找失败，则指针 p 为空指针}
 begin
  if T =nil then p:=nil
     else begin
          p:=T ; x:=K ; done:=false ;
          while ( p <> nil ) and ( not done ) do
           if K > p^.key then
                  begin r:=x mod n; x:=x div n; p:=p^.link[r+1] end
            else begin done:=true; if K <> p^.key then p:=nil end;
           end;
 end;
```

定理3. 算法2的时间复杂度为 $O(\log_n K)$.

若在算法1中利用[2]中习题2.12解的思想，可避免初始化. 故有下面的：

定理4. 本文中的 Trie 结构插入算法有一个时间复杂度为 $O(\log_n N)$ 的实现，且在此实现上的查找算法的时间复杂度仍为 $O(\log_n K)$.

2 Trie 结构一维数组的实现

在本节中，我们首先将上述 Trie 结构用一个矩阵表示，然后再将此矩阵存放到一个一维数组中，用矩阵表示 Trie 结构时，矩阵的每一行即为一个结点，行号作为该结点的名字，因而这时指针变量存放的是行号. 列号为结点各个域的名字，比如，我们约定第一列至第 n 列为指针域，第 n+1列为 Key 域，第 n+2列为 quot 域，另外用变量 root 记 Trie 结构根结点的名字，为方便起见，我们将矩阵的第 n+1列，第 n+2列从矩阵中分裂出来分别记为数组 D 和 Quot. 引言中的 Trie 结构可用下列矩阵表示：

$$A:\begin{bmatrix}3&2&6&0&0&0\\0&0&0&0&0&4\\0&0&0&0&0&5\\0&0&0&0&0&0\\0&0&0&0&0&0\\0&0&0&0&0&0\end{bmatrix}\qquad D:\begin{bmatrix}102\\121\\120\\211\\210\\212\end{bmatrix}\qquad root=1$$

图3

在图3中，数组 Quot 没有画出. 将图1与图3作一对照就会发现若 Trie 结构中的某个结点的指针数组存放在矩阵 A 的第 i 行，则该结点的关键字存放在 D[i]中. 利用[1]中的表压缩算法将矩阵 A 压缩到一维数组 A 中，即对矩阵 A 的第 i 行找到 rd[i]($1\leqslant i\leqslant n$)，使得

若 $A[i,j]\neq 0$ ，$A[i',j']\neq 0$ ，且 $(i,j)\neq(i',j')$时有

$$rd[i]+j-1\neq rd[i']+j'-1$$

一般在将 A 压缩到数组 C 中之前，应使 A 满足"harmonic decay"性质.[1]当 A 不满足"harmonic decay"性质时，可对 A 的第 j 列找到 cd[j]($1\leqslant j\leqslant n$)，使得当 A 的第 j 列向下移动 cd[j]个位置后，所得的矩阵 B 满足"harmonic decay"性质，然后将矩阵 B 压缩到一维数组 C 中，这时我们有如下关系：若 $A[i,j]\neq 0$，则有 $C[rd[i+cd[j]]+j-1]=A[i,j]$.

其中数组 cd 是对矩阵 A 而言的，数组 rd 是对矩阵 B 而言的，Tarjan 在[1] 中证明了数组 rd 需要 O(nloglogn) 的存储空间. cd，C 需要 O(n)的存储空间.

算法3：压缩后的 Trie 结构查找算法

```
Procedure Triesearch1(C:array[1..2n] of integer; D:array[1..n] of integer;K:integer; Var pos:integer );
  {在压缩后的 Trie 结构中查找关键字 K，若查找成功，则变量 pos 给出 K 在数组 D 中的位置，若查找失败，则 pos
   为0}
  begin
    pos:=root ; X:=K ; done:=false ;
    repeat
      if K=D[pos] then done:=true
        else if X=0 then begin pos:=0; done:=true end
                 else begin
                      r:=X mod n; X:=X div n; pos:=C[rd[pos+cd[r+1]]+r];
                      if pos=0 then done:=true ;
```

```
            end;
    until done;
  end;
```

定理5. 算法3正确地查找关键字 K.

证明：首先注意若 Trie 结构中的某个结点的指针数组存放在矩阵 A 的第 i 行，则该结点中的关键字存放在 D[i]中. 我们用变量 pos 记录将要查找的结点在矩阵 A 中的行号，并称此行号为该结点的名字. 在算法3中，我们从 Trie 结构的根结点开始查找，即将变量 pos 赋初值 root. 一般我们将 K 和 D[pos]相比较，若 K 等于 D[pos]，则查找成功，且 pos 恰为 K 在数组 D 中的位置. 若 K≠D[pos]，若已进行了 t 次比较，则 $x=K \text{ div } n^{t-1}$，令 $r=x \bmod n$，则下次应查找第 pos 个结点的第 r+1个指针所指向的结点，而该结点的名字存放在 A[pos，r+1]中，故下次应查找的结点对应的 pos 值为 C[rd[pos+cd[r+1]]+r]，若这是一此成功的查找，这样继续下去最终会找到 K. 而失败的查找则有两种情况：一种是 C[rd[pos+cd[r+1]+r]=0，这表明第 pos 个结点的第 r+1个指针为空，故这时 K 不在 Trie 结构中. 还有一种情况是 C[rd[pos+cd[r+1]+r]]≠0，而 x=0，这时已查找[$\log_n K$]+1 步，由定理1知这时 K 也不在 Trie 结构中，故查找失败.

定理6. 算法3的时间复杂度为 $O(\log_n K)$.

3　结束语

在本文中，我们修改了 Tarjan 在[1]中描述的 Trie 结构，改进了 Tarjan 的查找算法的时间复杂度. 更为重要的是，我们的 Trie 结构是一个堆，我们称之为 Trie 堆，有关 Trie 堆进一步的工作我们将另文讨论；并且本文给出了将 Trie 结构存放到一维数组后的查找算法. 这些算法有许多应用，例如可用于存储，查找 LR 分析表[3]. 如何将动态 Trie 结构存放到一维数组中，即如何使存放到一维数组后的 Trie 结构能进行插入、删除操作，这值得我们作进一步的研究.

参考文献

1　Tarjan R E, Yao A C—C. Storing a sparse table. Comm. ACM, 1979, **22(11)**: 606—611.

2　Aho A V, Hopcroft J E, Ullman J D. The design and analysis of computer algorithm. Reading, Mass.: Addison—Wesley, 1974.

3　Aho A V, Ullman J D. Principles of compiler design. Reading, Mass: Addison—Wesley, 1977.

A KIND OF TRIE STRUCTURE

Huang Jingwei

(*Wuhan University Of Hydraulic and Electric Engineering*, *Wuhan* 430072)

Dai Dawei

(*Wuhan University*, *Wuhan* 430072)

Abstract　In this paper, a kind of Trie structure is described. The insertion and search algorithms on this kind of Trie structure are presented. The time complexity the search algorithm is $O(\log_n K)$. Comparing to prior work, it is a improvement. This paper also presents the search algorithm on this kind of Trie structure stored into an one dimension array.

Key words　Trie structure*, searching, compression*.