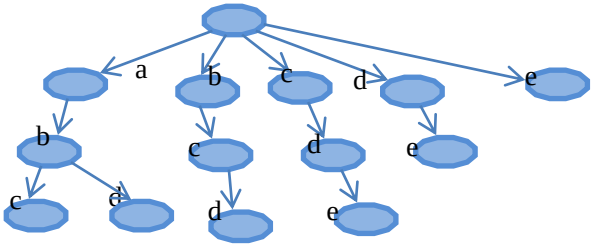
字母树

我们经常会遇到这类题目，要求给字符串一个或几个附加值，对其进行处理，查询相应字符串对应的某个值；或者在其他的题目如图论、DP、搜索等中加入字符串的相关处理。本文将简单介绍一种构造简单的、编程复杂度低、时间效率极高的高级数据结构——字母树，可以高效的进行字符串的一些处理。从字母树的构造、基本操作、应用等角度对其进行说明。

【字母树的初步了解】

字母树，又叫字典树、tire 树。顾名思义，它与字母和单词（字符串）密切相关的一种数据结构。如下图可以称之为一个字母树：



它由数字表示节点（为标出），字母表示边。如 a[7]表示第 7 个节点。

每一个结点都通过一条字母边连接到下一个结点。

从根到每一个节点所经过的路径组成的是一个字符串。这样只要对每一个节点进行某种标记，就可以判断从根到该节点是否满足一个合法字符串（即题目中给的）。

它充分利用了不同字符之间的公共前缀，如树中的“abc”和“abd”。

由上一条性质可以知道，字母树的末尾字母的后继结点从某种意义上可以代表一个字符串（上面提到的由路径组成的字符串）。

每一个节点会有至多 26 个儿子，与每一儿子相连的边是从“a”到“z”的字母，当然有的不足 26 个儿子。

【字母树的构造】

假设我们读入 n 个字符串，并将其插入到字母树中，这样字母树就构造成功了。字母树的构造和插入其实是相同的。

首先需要知道空字母树只是一个节点，没有其他域。

然后将字母树构造所需的基本的定义的变量进行说明，这是一个比较通用的代码：

```
type
  tire=record
  data:longint;
  next:array['a'..'z'] of longint;
  v:boolean;
end;
Var
  A:array[0..2000] of tire;
```

```
  len:longint;
```

这里的 tire 是一个记录类型的，a 是一个 tire 组成的数组。

对于每一个 a[i]，其中的 data 表示当前节点的某一个价值（其实是一个字符串对应的价值），初始为 0；v 是一个 boolan 型的，表示到当前节点是否是合法字符串（从根到当前节点），即上文所说的标记，初始为 false；Next 表示当前节点的儿子，其中使用字母表示边，初始值全部为 0。如 a[7].next[a]=9,表示第七个节点通过一条边 a 到达第九个节点;a[7].next[b]=0，说明第七个节点没有后继边 b。

Len 表示当前已经有几个节点了，初始 len=1。

这里给出插入的通用代码，然后进行说明：

```
procedure insert(s:string; k:longint);
var
  now,l,i:longint;
begin
  l:=length(s); now:=1;
  for i:=1 to l do
    if a[now].next[s[i]]<>0 then now:=a[now].next[s[i]] else
      begin
        inc(len);
        a[now].next[s[i]]:=len;
        now:=len;
      end;
  a[now].v:=true;
  a[now].data:=k;
end;
```

我们以插入“abc”为例：

1.初始时字母树为空，len=1；

2.执行 inert(abc,5)；

3.L=3； now=1；

4.实行 for 循环，从 1 到 3；

5.因为 a[1].next[a]=0,所以实行 begin 内容，开辟另一个新的节点 2，使得 a[1].next[a]=2,这是 now=len=2；

6.因为 a[2].next[b]=0,所以实行 begin 内容，开辟另一个新的节点 3，使得 a[2].next[b]=3,这是 now=len=3；

7.因为 a[3].next[c]=0,所以实行 begin 内容，开辟另一个新的节点 4，使得 a[3].next[c]=4,这是 now=len=4；

8.这是 for 循环执行完毕，将 a[4].v=true，a[now].data=5；

9.我们已经知道，从根节点可以一直延续到节点 4，而且节点 4 的 v 值为 true，所以由一路上经过的字母组成的字符串是一个合法字符串；

10.对于其他的字符串也是这样进行插入，注意一点，如果 a[now].next[s[i]]<>0,说明当前节点的的 s[i]边已经构造成功了，直接过渡到节点 a[now].next[s[i]]，不必重新开辟节点。这就是如何实现利用公共前缀。

为了更加详细的明白插入（即构造过程），请自己进行模拟和附录中的“*字母树的插入*”进行动态体验。

【字母树的查找和删除】

查找和删除时字母树的最基本的操作，如果我们了解了插入过程的进行，对于查找和删除时很好理解的。

查找的基本代码：

```
function find(s:string):longint;
var
  now,l,i:longint;
begin
  l:=length(s); now:=1;
  for i:=1 to l do
    if a[now].next[s[i]]=0 then exit(-1) else
      now:=a[now].next[s[i]];
  if a[now].v then exit(a[now].data);
  exit(-1);
end;
```

这里的查找其实在前面已经提到过，即从根节点（1 节点）一直沿着所需查找的字符串的每一个字母（即节点的后继边）过渡到下一个节点，一直到 for 循环的结束（或者由于当前节点没有所对应的边而退出），如果查到就返回该字符串的 data 值，反之返回-1。

删除的基本代码：

```
function delete(s:string):boolean;
var
  now,l,i:longint;
begin
  l:=length(s); now:=1;
  for i:=1 to l do
    if a[now].next[s[i]]=0 then exit(false) else
      now:=a[now].next[s[i]];
  if not a[now].v then exit(false);
  a[now].v:=false;
  a[now].data:=0;
  exit(true);
end;
```

这里的删除也需要进行查找，如果不能成功查找就返回 false；查找成功后，将对应的末尾节点的 v 值赋为 false，data 赋为 0，这样就不会在查找过程中查找

到。

【字母树时间效率的简单介绍】

通过了解前面的基本操作，我们可以发现，字母树的每一次基本操作的操作次数不会超过字母树的层数，而字母树的层数就是字符串的长度，所以字母树的操作的时间效率可以看成 O(L)。

如果有 n 此操作，m 个字符串，那么字符串的效率就是 O(max(ml,nl))。

对于 100000 个字符串，字符串长度不超过 20 的数据量，字母树完全可以承受。

【字母树的改进】

字母树并不是完美的。

我们知道，对于每一个节点，都需要为它准备 26 个子节点，有 a 数组的定义方式也能知道，如果数组开的很大，空间会受不了，这也是字母树的最大缺点，必须用空间换取时间。实际上，如果对于上述的 a 数组开到 1000000，只能承受不超过 200000 个长度不超过 10 的字符串，空间消耗大约 100mb。

但是这种缺点是可以在一定程度上进行弥补的，这就是用链表代替数组进行改进。

链表能实现动态分配内存，尽量把内存给最需要的，能减少很多的浪费，而且用链表不会出现全部内存溢出的情况。用链表写实际上降低了编程复杂度，但是需要注意开辟空间等。

这是用链表实现的比较通用的定义方式：

```
type
   tire=^rec;
   rec=record
   data:longint;
   next:array['a'..'z'] of tire;
   v:boolean;
end;
var
   a,temp:tire;
```

这里定义的 a 是我们实际中操作的字母树，只需定义一个。

Temp 是在查找过程中的中间变量，我们往往需要执行：

```
Temp:=a;
进行操作;
A:=temp;
```

对于每一个节点，它的 next 是指向另一个小的字母树，可以称之为这个节点的子树。

用链表实现的包括插入和查找的完整代码（删除操作请自己实现）参看附录“*相关程序*”。

希望读者能够自己真正理解用链表实现的代码。

【字母树处理对象的扩展】

通过对字母树的基本了解，我们很容易知道，字母树不仅仅局限于 26 个小写字母。

将 next 数组中的['a'..'z']改成['0'..'9']可以变成“数字树”。甚至直接将 next 数组变成[0..30]，表示可以出现 31 个字符，这种方式在实际中更常用到。

【字母树的应用】

说到底，字母树还是用来使用的。这里列举能使用到字母树的几类题目。

1.直接考察字母树的插入和查找。这类题目是比较简单的应用，一般很容易看出读者意图；

2.与单词的插入非常类似的一类题目，查询相同的字符串出现的次数，这时候需要附加一个数值 sum，表示出现的次数，没插入一次实行 inc(sum)，查询的时候可以返回 sum 值。

3.以字符串形式作为边或者以字母作为顶点的图论中，用字母树实现字符串的储存和进行搜索，字母树只是一种储存手段；

4.在搜索中储存字符串并进行搜索操作；

5.在 DP 中作为快速进行方程转移的手段；

6.在字符串排序中的应用；

具体的题目分析可以参看附录中“*字母树的应用*”。

总的来说，字母树有两大应用，第一是直接对其进行考察，第二就是作为辅助工具。在第二类应用中，有着非常广阔的“前景”，几乎所有类型的题目都可以把字符串作为基本操作对象，而字母树又是处理字符串的厉害工具。

【总结】

本文只是对字母树的最基本的介绍，可以作为字母树的入门教程。关于字母树的课件在网上不容易找到，自己感觉介绍的内容不太适合 noip 选手看，便写了这篇文章。

之前写过一篇，但是由于意外非常不幸的丢失了，只能重新写一个，结合了两个幻灯片，自己感觉比较清晰。

Noip 级别以上的同学不要局限与本文，应该查找更多的资料，比如“*国家集训队 2009 论文集浅析字母树在信息学竞*”和“*《Trie 图的构建、活用与改进》*”，进行更深入的研究。对于更加深入的知识，本文不做探讨。