思考与探讨

现在如果让读者自己实现一个代码编辑器中的**智能提示**功能，是不是有思路了呢？将需要提示的所有词汇做成一个字典树，每当用户输入了一个字符，则在字典树中将该字符后面子孙中的所有可能词汇列出来**候选**，是不是很酷？**有很多用数组实现的字典树，难度要比链式存储结构难好多！**

7.7 AC 自动机（Aho-Corasick）★★

字典树的核心就是空间换时间，空间消耗大（指针数组），但是插入和查询有着很优秀的时间复杂度，利用字符串的**公共前缀**来避免多余的字符串比较，降低查询时间。

字典树的平均高度 h 为单词平均长度 len，所以字典树的查询复杂度为 $\mathcal{O}(h) = \mathcal{O}(len)$。

字典树有一个很大的缺点就是每次只能查找一个单词是否在字典树中，一旦有一个节点比较失败了，就只能重新回到整棵树的根，重新开始比较。这是不是很像 KMP 之前的字符串匹配？观察图 7.20。

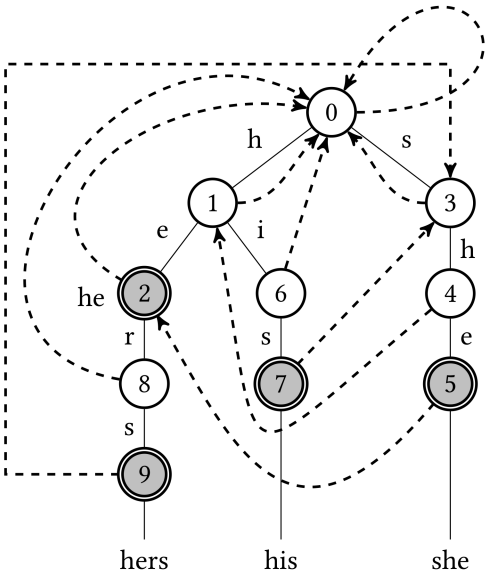

图 7.20：带失配指针的字典树——AC 自动机

（1）假设字典树匹配到节点 4 时失配（下一个节点不是 `'e'`），这个时候程序并不会回到节点 0 重新开始，而是跳到节点 1，因为节点 1 上面有一个 `'h'`，也就是说 $[0, 3, 4]$ 包含 $[0, 1]$，所以可以从节点 1 继续向下查找。

（2）当节点 7 匹配结束之后，直接跳到节点 3，因为 $[0, 1, 6, 7]$ 包含 $[0, 3]$，所以在找到字符串“his”之后，依然可以继续查找 `'s'` 开头的“she”存不存在。

图中的虚线叫作**失配指针**，每当比较失败或者找到一个单词之后，就沿着失配指针跳跃到另一个位置进行比较。看起来很复杂，其实就是一个树上的 KMP。问题的关键是如何构造失配指针，而且因为需要存储失配指针，所以字典树节点中还要再增加一个数据域用于存放失配指针。

next 数组就是失配指针数组呀！

程序 7.14: AC 自动机

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <stdbool.h>
5  #define ARRAY_SIZE(a) sizeof(a)/sizeof(a[0])
6  #define ALPHABET_SIZE (26)
7  #define CHAR_TO_INDEX(c) ((int)c - (int)'a')
8  struct TrieNode {
9      struct TrieNode *children[ALPHABET_SIZE];
10     bool isEndOfWord;
11     TrieNode *fail;          //失配指针
12 };
13 struct TrieNode *getNode(void) {
14     struct TrieNode *pNode = NULL;
15     pNode = (struct TrieNode *)malloc(sizeof(struct TrieNode));
16     if (pNode) {
17         int i;
18         pNode->isEndOfWord = false;
19         pNode->fail = NULL; //失配指针
20         for (i = 0; i < ALPHABET_SIZE; i++) pNode->children[i] = NULL;
21     }
22     return pNode;
23 }
24 void insert(struct TrieNode *root, const char *key) {
25     int level;
26     int length = strlen(key);
27     int index;
28     struct TrieNode *p = root;
29     for (level = 0; level < length; level++) {
30         index = CHAR_TO_INDEX(key[level]);
31         if (!p->children[index]) p->children[index] = getNode();
32
33         p = p->children[index];
34     }
35     p->isEndOfWord = true;
36 }
37
38 void build_AC(TrieNode *root){//构造失配指针
39     TrieNode *q[100000];//构造失配指针的bfs队列
```

```
40      int head=0;
41      int  tail=1;
42      q[head]=root;
43      TrieNode *temp,*p;
44      while(head<tail){ //bfs构造字典树的失配指针,需要由近到远一层层地来构造
45          temp=q[head++];
46          for(int i=0; i<26; i++){
47              if(temp->children[i]){//如果某个分支存在
48                  //root下的第一层节点的失配指针都指向root
49                  if(temp==root)
50                      temp->children[i]->fail=root;
51                  else{
52                      //依次回溯该节点的父节点的失配指针
53                      //直到某节点的next[i]与该节点相同，则
54                      //把该节点的失配指针指向该next[i]节点
55                      //若回溯到 root 都没有找到,则该节点的失配指针 指向 root
56                      p=temp->fail;//加入temp为节点4,失配指针指向1,p==1
57                      while(p){ //当p不为空（不指向root）
58                          if(p->children[i]){
59                    //4中指定分支的失配指针指向1的特定分支，5节点指向了2节点
60                              temp->children[i]->fail=p->children[i];
61                              break;
62                          }
63                          p=p->fail;//找不到相等的，就继续退
64                      }     //退到头了，就执行root
65                      if(!p)temp->children[i]->fail=root;
66                  }
67                  //每处理一个节点,就把它的所有孩子节点加入队列
68                  q[tail++]=temp->children[i];
69              }
70          }
71      }
72  }
73  int query(char *str,TrieNode *root){
74      int ans=0,k,len=strlen(str);
75      TrieNode *p=root;
76      //i为主串指针，p为匹配串指针
77      for(int i=0; i<len; i++){ //注意i没有回溯,拿出每一个str[i]
78          k=CHAR_TO_INDEX(str[i]);
79          //匹配失败，则回溯，直到回到根
80          while(!p->children[k]&&p!=root)
81              p=p->fail;
82          p=p->children[k];//沿着k向下走
83          if(!p)p=root;//指针仍为空，则没有找到与之匹配的字符
84          //指针重新回到根节点root,下次从root开始搜索字典树
85          TrieNode *temp=p;//匹配该节点后,沿其失配指针回溯,判断其他节点是否匹配
86          while(temp!=root&&temp->isEndOfWord){//假设匹配了5节点
87              ans+=1; //匹配数增加
```

```
88                  //temp->isEndOfWord=-1;//标记已访问
89                  printf("\n--->%2d %c\n",i,str[i]);
90                  temp=temp->fail;//回溯失配指针，继续寻找下一个满足条件的节点，例如
                        节点2
91              }
92          }
93          return ans;
94      }
95      int main() {
96          //Input keys (use only 'a' through 'z' and lower case)
97          char keys[][8] = {"the", "a", "there", "answer", "any",
98              "by", "bye", "the"};
99          struct TrieNode *root = getNode();
100         int i;
101         for (i = 0; i < ARRAY_SIZE(keys); i++) insert(root, keys[i]);
102
103         build_AC(root);
104         //答案6，分别为the,there,a,answer,a,any。两个the，字典树直接去重了
105         char *str="thereanswerany";
106         printf("%d\n",query(str,root));
107         return 0;
108     }
```

AC 自动机是 KMP 算法在树中的推广，可以快速地在一篇文章中找到一批需要寻找的词汇。相对于字典树，AC 自动机的难点在于失配指针的构造。还记得本书常使用的一个方法吗? **实例化**，举一个特殊的例子，就可以看清算法的精髓。仔细阅读程序 7.14 的注释，配合图 7.20，就可以明白为什么使用队列及失配指针是如何计算的。

计算节点 4 中的分支数组下标 `'e'` 所指节点的失配指针的时候，要参考节点 4 的失配指针，其指向节点 1。4 上面是从 `'h'` 来的，1 上面也是从 `'h'` 来的，也就是说如果节点 4 无法向下走了，可以去节点 1 向下走走看。如果这个时候如图 7.20 所示，`p->children[i]` 为真，即节点 1 对应的分支节点 `'e'` 非空，也就是两个 `'e'` 相等了，那么直接给失配指针赋值就好了，`temp->children[i]->fail = p->children[i];`，节点 5 的失配指针就是节点 2 了。也就是说，如果后面匹配过程从节点 5 无法向下走，那么就跳到节点 2 继续向下看看。

还有一个需要思考的地方是 `TrieNode` 结构体中的 `isEndOfWord` 变量，这里依然使用布尔类型。很多书籍将此成员定义成整型。因为 AC 自动机与字典树不同，AC 自动机一次寻找所有的单词，如果单词有重复，那么采用布尔类型就无法计算次数了。例如主函数中 `keys` 包含两个“the”，如果重复也算一个，那就应该返回 7，本书的算法返回 6，在生成字典树的时候直接去重了。如果 `isEndOfWord` 是整型，要统计重复单词，算法会复杂一些。读者感兴趣可以自行解决。

思考与探讨

为什么 IT 大厂、高校考研录取都喜欢参加算法竞赛的学生？只有算法学好了，有些问题才知道怎么解决，才能够创造性地解决新问题。

7.8 课后练习

单项选择题：

1. 在一棵度为 4 的树 T 中，若有 20 个度为 4 的节点，10 个度为 3 的节点，1 个度为 2 的节点，10 个度为 1 的节点，则树 T 的叶子节点个数是（　　）。

A. 41　　B. 82　　C. 113　　D. 122

答：B

设树中度为 i（$i=0,1,2,3,4$）的节点数分别为 n_i，节点总数为 n，分支总数为 k，则 $n=k+1$，而分支数又等于树中各节点的度之和，即 $n=n_0+n_1+n_2+n_3+n_4=n_0+10+1+10+20=n_0+41$，$k=0\times n_0+1\times n_1+2\times n_2+3\times n_3+4\times n_4=10+2+30+80=122$。整理可得 $n_0+41=122$，解得 $n_0=82$ 即树 T 的叶节点的个数是 82。

搞懂树的节点数与分支数的关系很重要。

2. 树最适合用来表示（　　）的数据。

A. 有序　　B. 无序

C. 任意元素之间具有多种联系　　D. 元素之间具有分支层次关系

答：D

树是一种分层结构，它特别适合组织那些具有分支层次关系的数据。

3. 具有 10 个叶子节点的二叉树中有（　　）个度为 2 的节点。

A. 8　　B. 9　　C. 10　　D. 11

答：B

由二叉树的性质 $n_0=n_2+1$，得 $n_2=n_0-1=10-1=9$。

4. 一个具有 1025 个节点的二叉树的高为（　　）。

A. 11　　B. 10　　C. 11 ～ 1025　　D. 10 ～ 1024

答：C

当二叉树为单支树时具有最大高度，即每层上只有一个节点，最大高度为 1025。而当树为完全二叉树时，其高度最小，最小高度为 $\lfloor\log_2 n\rfloor+1=11$。

一看题目就可以排除选项 A、B，因为答案必须是一个区间。1025 个点，当然最多有 1025 层，所以直接选 C。

5. 设二叉树只有度为 0 和 2 的节点，其节点个数为 15，则该二叉树的最大深度为（　　）。

A. 4　　B. 5　　C. 8　　D. 9

答：C

第一层有一个节点，其余 $h-1$ 层上各有两个节点，总节点数 $=1+2(h-1)=15$，$h=8$。画草图观察。

6. 若一棵完全二叉树有 768 个节点，则该二叉树中叶子节点的个数是（　　）。

A. 257　　B. 258　　C. 384　　D. 385

答：C