http://dnizna.javaeye.com/blog/657147

AC 自动机算法详解

首先简要介绍一下 AC 自动机：Aho-Corasick automation，该算法在1975年产生于贝尔实验室，是著名的多模匹配算法之一。一个常见的例子就是给出 n 个单词，再给出一段包含 m 个字符的文章，让你找出有多少个单词在文章里出现过。要搞懂 AC 自动机，先得有模式树（字典树）Trie 和 KMP 模式匹配算法的基础知识。AC 自动机算法分为3步：构造一棵 Trie 树，构造失败指针和模式匹配过程。

如果你对 KMP 算法和了解的话，应该知道 KMP 算法中的 next 函数（shift 函数或者 fail 函数）是干什么用的。KMP 中我们用两个指针 i 和 j 分别表示，A[i-j+ 1..i]与 B[1..j]完全相等。也就是说，i 是不断增加的，随着 i 的增加 j 相应地变化，且 j 满足以 A[i]结尾的长度为 j 的字符串正好匹配 B 串的前 j 个字符，当 A[i+1]≠B[j+1]，KMP 的策略是调整 j 的位置（减小 j 值）使得 A[i-j+1..i]与 B[1..j]保持匹配且新的 B[j+1]恰好与 A[i+1]匹配，而 next 函数恰恰记录了这个 j 应该调整到的位置。同样 AC 自动机的失败指针具有同样的功能，也就是说当我们的模式串在 Tire 上进行匹配时，如果与当前节点的关键字不能继续匹配的时候，就应该去当前节点的失败指针所指向的节点继续进行匹配。

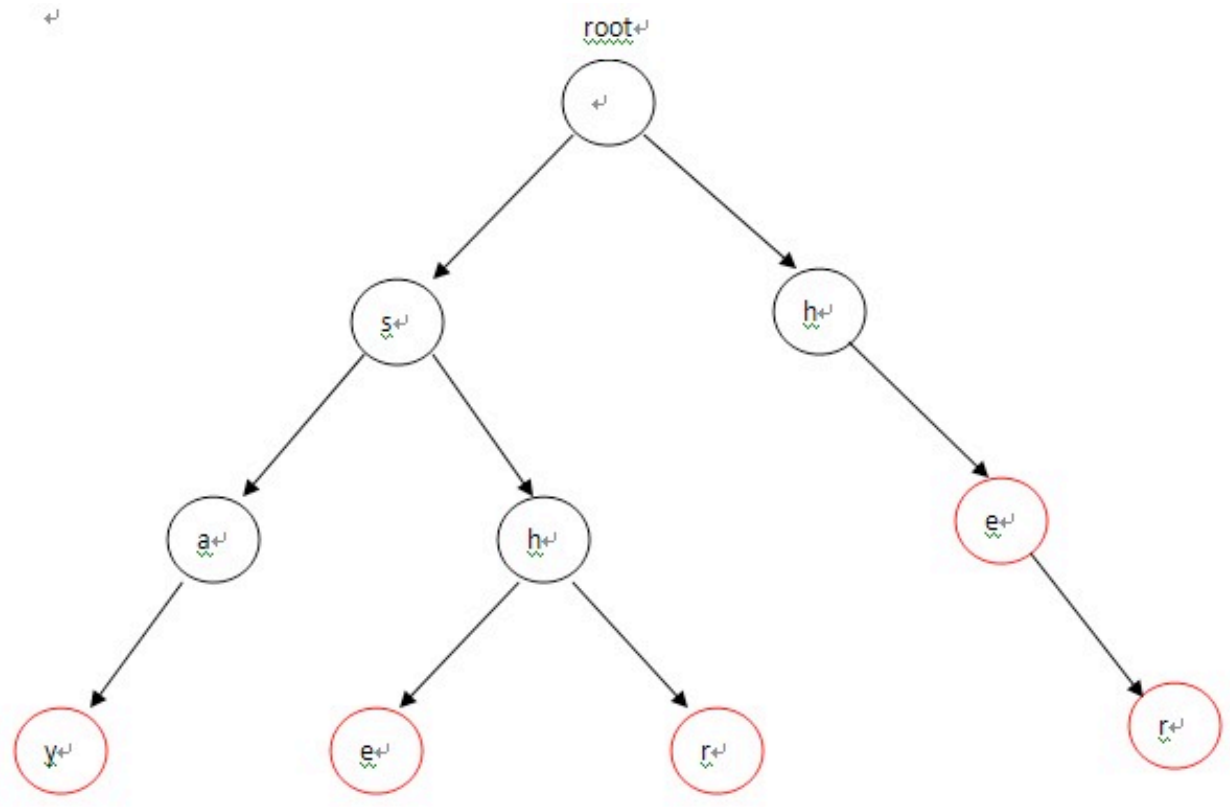
看下面这个例子：给定5个单词：say she shr he her，然后给定一个字符串 yasherhs。问一共有多少单词在这个字符串中出现过。我们先规定一下 AC 自动机所需要的一些数据结构，方便接下去的编程。

```
1 const int kind = 26;
2 struct node{
3     node *fail;         //失败指针
4     node *next[kind]; //Tire 每个节点的个子节点（最多个字母）
5     int count;          //是否为该单词的最后一个节点
6     node(){             //构造函数初始化
7         fail=NULL;
8         count=0;
9         memset(next,NULL,sizeof(next));
```

```
10      }
11 }*q[500001];            //队列，方便用于 bfs 构造失败指针
12 char keyword[51];       //输入的单词
13 char str[1000001];      //模式串
14 int head,tail;          //队列的头尾指针
```

有了这些数据结构之后，就可以开始编程了：

首先，将这5个单词构造成一棵 Tire，如图-1所示。




```
1 void insert(char *str,node *root){
2     node *p=root;
3     int i=0,index;
4     while(str[i]){
5         index=str[i]-'a';
6         if(p->next[index]==NULL) p->next[index]=new node();
```

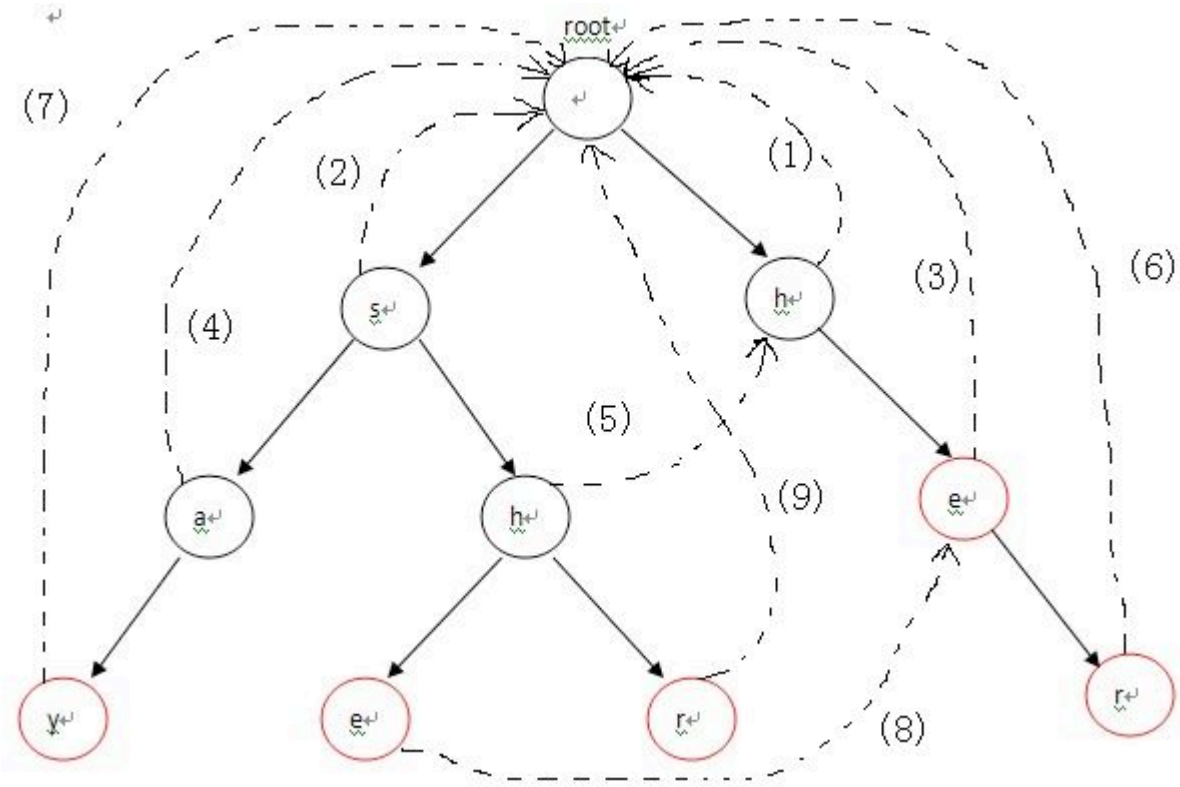
```
        p=p->next[index];
        i++;
    }
    p->count++;     //在单词的最后一个节点 count+1，代表一个单词
}
```

在构造完这棵 Tire 之后，接下去的工作就是构造下失败指针。构造失败指针的过程概括起来就一句话：设这个节点上的字母为 C，沿着他父亲的失败指针走，直到走到一个节点，他的儿子中也有字母为 C 的节点。然后把当前节点的失败指针指向那个字母也为 C 的儿子。如果一直走到了 root 都没找到，那就把失败指针指向 root。具体操作起来只需要：先把 root 加入队列(root 的失败指针指向自己或者 NULL)，这以后我们每处理一个点，就把它的所有儿子加入队列，队列为空。

```
void build_ac_automation(node *root){
    int i;
    root->fail=NULL; //root 的失败指针指向自己或者 NULL
    q[head++]=root; //把 root 加入队列
    while(head!=tail){
        node *temp=q[tail++];
        node *p=NULL;
        for(i=0;i<26;i++){
            if(temp->next[i]!=NULL){
                if(temp==root) temp->next[i]->fail=root;
                else{
                    p=temp->fail; //p 指向 h 节点所指的节点，也就是 root
                    while(p!=NULL){
                        if(p->next[i]!=NULL){
                            temp->next[i]->fail=p->next[i];
                            break;
                        }
                        p=p->fail;
                    }
```

```
20                      if(p==NULL) temp->next[i]->fail=root;
21                  }
22                  q[head++]=temp->next[i];
23              }
24          }
25      }
26  }
```

从代码观察下构造失败指针的流程：对照图-2来看，首先 root 的 fail 指针指向 NULL，然后 root 入队，进入循环。第1次循环的时候，我们需要处理2个节点：root->next[‘h’-‘a’](节点 h)和 root->next[‘s’-‘a’](节点 s)。把这2个节点的失败指针指向 root，并且先后进入队列，失败指针的指向对应图-2中的(1)，(2)两条虚线；第2次进入循环后，从队列中先弹出 h，接下来 p 指向 h 节点的 fail 指针指向的节点，也就是 root；进入第13行的循环后，p=p->fail 也就是 p=NULL，这时退出循环，并把节点 e 的 fail 指针指向 root，对应图-2中的(3)，然后节点 e 进入队列；第3次循环时，弹出的第一个节点 a 的操作与上一步操作的节点 e 相同，把 a 的 fail 指针指向 root，对应图-2中的(4)，并入队；第4次进入循环时，弹出节点 h(图中左边那个)，这时操作略有不同。在程序运行到14行时，由于 p->next[i]!=NULL(root 有 h 这个儿子节点，图中右边那个)，这样便把左边那个 h 节点的失败指针指向右边那个 root 的儿子节点 h，对应图-2中的(5)，然后 h 入队。以此类推：在循环结束后，所有的失败指针就是图-2中的这种形式。



最后，我们便可以在 AC 自动机上查找模式串中出现过哪些单词了。匹配过程分两种情况：(1)当前字符匹配，表示从当前节点沿着树边有一条路径可以到达目标字符，此时只需沿该路径走向下一个节点继续匹配即可，目标字符串指针移向下个字符继续匹配；(2)当前字符不匹配，则去当前节点失败指针所指向的字符继续匹配，匹配过程随着指针指向 root 结束。重复这2个过程中的任意一个，直到模式串走到结尾为止。

```
1 int query(node *root){
2     int i=0,cnt=0,index,len=strlen(str);
3     node *p=root;
4     while(str[i]){
5         index=str[i]-'a';  //index=0~26
6         while(p->next[index]==NULL && p!=root)
                  p=p->fail; //p 指向 p 的失败指针所指向的节点
7         p=p->next[index]; //p 指向 index 节点
8         p=(p==NULL)?root:p;
9         node *temp=p;
```

```
10          while(temp!=root && temp->count!=-1){
11              cnt+=temp->count; //节点信息为1时，cnt+1，
12              temp->count=-1; //将 count 信息置为-1，表示已经出现过了
13              temp=temp->fail; //temp 指向 e 的失败指针说指向的节点继续查找
14          }
15          i++;
16      }
17      return cnt;
18  }
```

对照图-2，看一下模式匹配这个详细的流程，其中模式串为 yasherhs。对于 i=0,1。Trie 中没有对应的路径，故不做任何操作；i=2,3,4时，指针 p 走到左下节点 e。因为节点 e 的 count 信息为 1，所以 cnt+1，并且将节点 e 的 count 值设置为-1，表示改单词已经出现过了，防止重复计数，最后 temp 指向 e 节点的失败指针所指向的节点继续查找，以此类推，最后 temp 指向 root，退出 while 循环，这个过程中 count 增加了2。表示找到了2个单词 she 和 he。当 i=5时，程序进入第5行，p 指向其失败指针的节点，也就是右边那个 e 节点，随后在第6行指向 r 节点，r 节点的 count 值为1，从而 count+1，循环直到 temp 指向 root 为止。最后 i=6,7时，找不到任何匹配，匹配过程结束。

到此为止 AC 自动机算法的详细过程已经全部介绍结束，看一道例题：http://acm.hdu.edu.cn/showproblem.php?pid=2222

Problem Description

In the modern time, Search engine came into the life of everybody like Google, Baidu, etc.

Wiskey also wants to bring this feature to his image retrieval system.

Every image have a long description, when users type some keywords to find the image, the system will match the keywords with description of image and show the image which the most keywords be matched.

To simplify the problem, giving you a description of image, and some keywords, you should tell me how many keywords will be match.

Input

First line will contain one integer means how many cases will follow by.

Each case will contain two integers N means the number of keywords and N keywords follow. (N <= 10000)

Each keyword will only contains characters 'a'-'z', and the length will be not longer than 50.

The last line is the description, and the length will be not longer than 1000000.

Output

Print how many keywords are contained in the description.

Sample Input

1

5

she

he

say

shr

her

yasherhs

Sample Output

3

```cpp
#include <iostream>
using namespace std;

const int kind = 26;
struct node{
    node *fail;         //失败指针
    node *next[kind]; //Tire 每个节点的26个子节点（最多26个字母）
    int count;          //是否为该单词的最后一个节点
    node(){             //构造函数初始化
        fail=NULL;
        count=0;
        memset(next,NULL,sizeof(next));
    }
}*q[500001];            //队列，方便用于 bfs 构造失败指针
char keyword[51];       //输入的单词
char str[1000001];     //模式串
int head,tail;          //队列的头尾指针

void insert(char *str,node *root){
    node *p=root;
    int i=0,index;
    while(str[i]){
        index=str[i]-'a';
        if(p->next[index]==NULL) p->next[index]=new node();
        p=p->next[index];
        i++;
    }
    p->count++;
```

```
29 }
30 void build_ac_automation(node *root){
31     int i;
32     root->fail=NULL;
33     q[head++]=root;
34     while(head!=tail){
35         node *temp=q[tail++];
36         node *p=NULL;
37         for(i=0;i<26;i++){
38             if(temp->next[i]!=NULL){
39                 if(temp==root) temp->next[i]->fail=root;
40                 else{
41                     p=temp->fail;
42                     while(p!=NULL){
43                         if(p->next[i]!=NULL){
44                             temp->next[i]->fail=p->next[i];
45                             break;
46                         }
47                         p=p->fail;
48                     }
49                     if(p==NULL) temp->next[i]->fail=root;
50                 }
51                 q[head++]=temp->next[i];
52             }
53         }
54     }
55 }
56 int query(node *root){
57     int i=0,cnt=0,index,len=strlen(str);
```

```
58      node *p=root;
59      while(str[i]){
60          index=str[i]-'a';
61          while(p->next[index]==NULL && p!=root) p=p->fail;
62          p=p->next[index];
63          p=(p==NULL)?root:p;
64          node *temp=p;
65          while(temp!=root && temp->count!=-1){
66              cnt+=temp->count;
67              temp->count=-1;
68              temp=temp->fail;
69          }
70          i++;
71      }
72      return cnt;
73  }
74  int main(){
75      int n,t;
76      scanf("%d",&t);
77      while(t--){
78          head=tail=0;
79          node *root=new node();
80          scanf("%d",&n);
81          getchar();
82          while(n--){
83              gets(keyword);
84              insert(keyword,root);
85          }
86          build_ac_automation(root);
```

```
87          scanf("%s",str);
88          printf("%d\n",query(root));
89      }
90      return 0;
91  }
```