简述

AC自动机可以说是KMP的进阶版，支持多个模式串在目标串中的查询。
复杂度为$O((N+M)\times L)$

思路

首先对模式串进行trie树的构建。
接着引入fail的概念：对于trie树的每一个结点都有一个fail值，fail对应了另一个树上的结点，从root到fail的字符串是从root到该结点的字符串的最大后缀。 类似于KMP中的Next数组。 匹配过程中，若对一个结点匹配，则对其fail，fail'sfail....均匹配；若对一个结点失配，则应来到其父结点的fail再进行匹配。

操作

建树

对所有的模式串进行trie树的构建。 这个就不贴代码了。

计算fail

运用队列

1. root入队。
2. 按顺序出队，遍历子结点，子结点的fail为父节点fail的与子结点字符相同的子结点，若子结点不存在则指定该位置为父结点fail的与子结点字符相同的子结点。
注：原来的版本是没有最后的半句的，而倒数第二个半句也有一个要添加的地方就是若父结点`fail`的与子结点字符相同的结点不存在则跳转至父结点`fail`的`fail`寻找这样的子结点，若再没有则继续以此类推。这里用最后的半句巧妙地预处理了这件事并且避开了程序递归的过程，而在常规的队列操作中实现了递归的效果。
3. 子结点入队。

需要注意的是，root子结点的fail应当是root，然而它们父亲(root)的fail(root)与它们字符相同的结点就是他们本身，不符合规则。因此设置“ 0 ”结点，使root结点的fail为0，0的所有子结点为root，从而能实现这个效果。

```
void Fail()
{
    for(int i=0;i<26;i++)
        trie[0][i]=1;
    queue<int>q;
    q.push(1);
```

```
        int head=-1,foh=0;
    for(int m=1;m;q.pop(),head=q.front(),m--,foh=fail[head])
        for(int i=0,s=trie[head][0];i<26;s=trie[head][++i])
        {
            if(!s) trie[head][i]=trie[foh][i];
            else
            {
                fail[s]=trie[foh][i];
                q.push(s);
                m++;
            }
        }
}
```

进行匹配

如果一个结点匹配成功，那么它的fail也可以匹配成功，以此类推。
这样是将目标串的当前关注位置与所有可能的模式串匹配。

主线正常按照tire树查找的方式进行。
这里放只需判断模式串存在与否的代码。

```
int AC()
{
    int ans=0,root=1,id=t[0]-97,k=trie[root][id];
    for(int i=0;t[i];i++,id=t[i]-97,k=trie[root][id])
    {
        while(k>1&&freq[k]!=-1)
        {
            ans+=freq[k];
      freq[k]=-1;
            k=fail[k];
        }
        root=trie[root][id];
    }
    return ans;
}
```

例题

说起来其实只做了几道入门题。~~天天打游戏的某人~~
所以放入门题吧。~~羞耻~~

洛谷 p3786 AC自动机加强版

判断哪些模式串在目标串中出现的次数最多。
这道题就需要统计出现的次数了，因此在每个点跳fail就不能像最基础的问题一样直接标记去过了就不去了，而是每一次都顺着fail路径一路统计下去。

```
#include<cstdio>
#include<cstring>
#include<queue>
using namespace std;
int n,vist[11000],tot;
char t[1000005],ss[155][75];
struct unit{int v,s[27],fail,flag;}T[11000];
void init()
{
    for(int i=1;i<=tot;i++)
    {
        T[i].fail=T[i].flag=0;
        memset(T[i].s,0,sizeof(T[i].s));
    }
    memset(vist,0,sizeof(vist));
    tot=1;
}
void Insert(int num)
{
    int root=1;
    for(int
i=0,id=ss[num][0]-97;ss[num][i];root=T[root].s[id],id=ss[num][++i]-97)
    {
        if(!T[root].s[id])
            T[root].s[id]=++tot;
    }
    T[root].flag=num;
}
void get_fail()
{
    for(int i=0;i<26;i++)
        T[0].s[i]=1;
    queue<int>q;
    q.push(1);
    T[1].fail=0;
    int head,m=1,fafail;
    while(m--)
    {
        head=q.front();
        q.pop();
        fafail=T[head].fail;
        for(int i=0;i<26;i++)
        {
```

```
            if(!T[head].s[i]) T[head].s[i]=T[fafail].s[i];
            else
            {
                T[T[head].s[i]].fail=T[fafail].s[i];
                m++;
                q.push(T[head].s[i]);
            }
        }
    }
}
void match()
{
    scanf("%s",t);
    for(int
i=0,root=1,id=t[0]-97,k=T[1].s[id];t[i];root=T[root].s[id],id=t[++i]-97,k=T
[root].s[id])
        while(k>1)
        {
            if(T[k].flag) vist[T[k].flag]++;
            k=T[k].fail;
        }
}
int main()
{
    while(scanf("%d",&n)&&n)
    {
        init();
        for(int i=1;i<=n;i++)
        {
            scanf("%s",ss[i]);
            Insert(i);
        }
        get_fail();
        match();
        int mx=0;
        for(int i=1;i<=n;i++)
            if(vist[i]>mx) mx=vist[i];
        printf("%d\n",mx);
        for(int i=1;i<=n;i++)
            if(vist[i]==mx)
                printf("%s\n",ss[i]);
    }
    return 0;
}
```

洛谷 p5357 AC自动机二次加强版

分别求出每个模式串在目标串中的出现次数。
这道题如果按照上面的做法会tle因为跳fail的操作存在大量重复。

优化的方式就是在本来要跳fail的地方只打一个标记而不跳fail，在匹配结束后进行fail关系的拓扑排序从而找到fail的“根源点”（入度为零）,再从这些点顺着fail边统计每一个点真正匹配成功的次数。
利用这种方法相当于只进行了一次跳fail。

```
#include<cstdio>
#include<queue>
using namespace std;
int n,tot=1,vist[200005],pos[200005];
char t[200005],s[2000005];
struct unit{int son[27],fail,flag,in,ans;}T[200005];
void Insert(int num)
{
    int root=1,id=t[0]-97;
    for(int i=0;t[i];root=T[root].son[id],id=t[++i]-97)
        if(!T[root].son[id])
            T[root].son[id]=++tot;
    T[root].flag=1;
    pos[num]=root;
}
void get_fail()
{
    for(int i=0;i<26;i++)
        T[0].son[i]=1;
    queue<int>q;
    q.push(1);
    T[1].fail=0;
    int m=1,head,fafail,v;
    while(m--)
    {
        head=q.front();
        fafail=T[head].fail;
        q.pop();
        for(int i=0;i<26;i++)
        {
            v=T[head].son[i];
            if(!v) T[head].son[i]=T[fafail].son[i];
            else
            {
                T[v].fail=T[fafail].son[i];
                T[T[v].fail].in++;
                m++;
                q.push(v);
            }
        }
    }
}
void match()
{
    scanf("%s",s);
    for(int
```

```
i=0,id=s[0]-97,root=T[1].son[id];s[i];id=s[++i]-97,root=T[root].son[id])
        T[root].ans++;
}
void topu()
{
    queue<int>q;
    int m=0,head,v;
    for(int i=1;i<=tot;i++)
        if(!(T[i].in))
        {
            m++;
            q.push(i);
        }
    while(m--)
    {
        head=q.front();
        q.pop();
        if(T[head].flag) vist[head]=T[head].ans;
        v=T[head].fail;
        T[v].ans+=T[head].ans;

        if(!(--T[v].in))
        {
            q.push(v);
            m++;
        }
    }
}
int main()
{
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
    {
        scanf("%s",t);
        Insert(i);
    }
    get_fail();
    match();
    topu();
    for(int i=1;i<=n;i++)
        printf("%d\n",vist[pos[i]]);
    return 0;
}
```

洛谷 p4052 [JSOI2007] 文本生成器

给出一些模式串和一个长度，求出给定长度下至少包含一个给出的模式串的文本串数量。

这道题可以通过计算不包含任何一个模式串的文本串数量间接求解。

主要是在Trie树上运用dp

在这里我想做题的人需要对文本串在Trie树上匹配的过程有一个直观而清晰的认识——匹配到当前的树上的位置代表一切其fail一串上的均可以匹配，但以该点为fail的位置不能被匹配。

运用dp dp[i][j]表示 长度为i 结点序号为j的位置不包含任何一个模式串的文本串数量。
状态转移方程为 dp[i+1][son[j][k]]+=dp[i][j] (!flag[son[j][k])
这里面需要注意的是，如果一个结点是一个模式串的末尾，那么以其为fail关系的子孙的结点都要标记flag 原因如上。

```
#include<cstdio>
#include<queue>
using namespace std;
int n,m,tot=1,dp[6005][105],mod=10007;
char s[105];
struct unit{int s[27],fail,flag;}T[6005];
void Insert()
{
    int root=1,id;
    for(int i=0;s[i];i++)
    {
        id=s[i]-65;
        if(!T[root].s[id])
            T[root].s[id]=++tot;
        root=T[root].s[id];
    }
    T[root].flag=1;
}
void get_fail()
{
    for(int i=0;i<26;i++)
        T[0].s[i]=1;
    T[1].fail=0;
    queue<int>q;
    q.push(1);
    int m=1,head;
    while(m--)
    {
        head=q.front();
        q.pop();
        for(int i=0,v,fafail;i<26;i++)
        {
            v=T[head].s[i];
            fafail=T[head].fail;
            if(!v) T[head].s[i]=T[fafail].s[i];
            else
            {
                T[v].fail=T[fafail].s[i];
                T[v].flag|=T[T[v].fail].flag;
                q.push(v);
                m++;
```

```
                }
            }
        }
    }
int main()
{
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++)
    {
        scanf("%s",s);
        Insert();
    }
    get_fail();
    dp[1][0]=1;
    for(int i=1;i<=m;i++)
        for(int j=1;j<=tot;j++)
            for(int k=0;k<26;k++)
                if(!T[T[j].s[k]].flag)
                    dp[T[j].s[k]][i]=(dp[T[j].s[k]][i]+dp[j][i-1])%mod;
    int ans=1;
    for(int i=26,j=m;j;j>>=1,i=(i*i)%mod)
        if(j&1)
            ans=(ans*i)%mod;
    for(int i=1;i<=tot;i++)
        ans=(ans-dp[i][m]+mod)%mod;
    printf("%d",ans);
    return 0;
}
```