回文自动机(PAM)

介绍：回文自动机是什么呢，就是一个用树形结构维护字符串的所有回文子串信息的算法。

这里主要整理算法实现的过程，不会细讲，~~太懒子~~。

1.预备处理：

下面的一个结点代表一个回文串。

$char\ a[]$:字符串

$len[i]$:结点i的回文长度。

$fail[i]$:结点i指向最长回文后缀的结点。

$num[i]$:结点i的回文后缀个数。

$tri[i][j]$:结点i的j儿子，j代表一个字符$ch=j+$‘a‘

cur :当前需要匹配的结点。

cnt :结点个数。

2.初始化：

构建两棵树，一棵储存偶长度回文串，一棵储存奇长度回文串。

偶根为结点0(空回文串)，奇根为结点1(没有意义,便于计算)。

$len[1]=-1,fail[0]=1,fail[1]=0,cnt=1$。

$len[1]=-1$是为便于计算奇回文串长度，当我们更新的时候若只有一个字符a，则它的最长回文长度就是$-1+2=1$。

$fail[0]=1$是为了当不存在aa的回文串时，让当前字符a和自己匹配，也就是我们保证如果一个字符匹配不到其他的回文串时，最终总能匹配到自己。

$fail[1]=0$没有意义，因为匹配到奇根就会停止了。

$cnt=1$因为当前我们已经建立1号结点.

3.不断插入字符求回文串。

操作：

1.首先找到能够对当前字符i进最大回文匹配的结点cur，此步用cur不断跳$fail$指针实现。

2.判断当前结点cur结点是否有儿子$a[i]-'a'$，即判断是否有边的转移。如果有的话我们直接将cur赋值为$tri[cur][a[i]-'a']$，否则我们需要建立一个新结点。

3.建立新结点并更新$fail[]$：首先我们更新$len[++cnt]=len[cur]+2$，即新结点能向cur的两遍扩展成回文串。

然后我们先要将cur跳一次$fail$指针，我们令$j=fail[cur]$即从它的父结点往上找到一个有儿子结点的结点，然后我们将当前结点的$fail[cnt]$指向它，即$fail[cnt]=tri[j][a[i]-'a']$.

然后我们建立新结点，即$tri[cur][a[i]-'a']=cnt$。

最后我们就可以更新回文后缀个数：$num[cnt]=num[fail[cnt]]+1$.

因为$fail[]$指向的是它的最大回文后缀，由于扩展一次，所以加1.

最后我们将这次的cur赋值为$tri[cur][a[i]-'a']$，成为下一次匹配的结点就完成了。

时间复杂度：$O(n)$

空间复杂度：$O(n)$ (定理：一个字符串本质不同的回文串个数不超过其长度。)

代码：

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=1e5+5,M=1e6+5,inf=0x3f3f3f3f,mod=1e9+7;
#define mst(a) memset(a,0,sizeof a)
#define lx x<<1
#define rx x<<1|1
#define reg register
#define PII pair<int,int>
#define fi first
#define se second
char a[N];
int tri[N][26],len[N],fail[N],cnt=1,cur;    //预备处理.
int main(){
    scanf("%s",a+1);
    int n=strlen(a+1);
    fail[0]=1,len[1]=-1;//初始化.
    for(int i=1;i<=n;i++){
        while(a[i-len[cur]-1]!=a[i]) cur=fail[cur]; //匹配.
        if(!tri[cur][a[i]-'a']){    //建立新结点.
             len[++cnt]=len[cur]+2; //更新长度.
             int j=fail[cur];  //跳一次fail,因为不能匹配本身为回文后缀.
             while(a[i-len[j]-1]!=a[i]) j=fail[j];  //求fail指针
             fail[cnt]=tri[j][a[i]-'a'];//求出fail指针.
             tri[cur][a[i]-'a']=cnt;//插入该结点.
             num[cnt]=num[fail[cnt]]+1;     //更新回文后缀个数.
        }
        cur=tri[cur][a[i]-'a'];//cur变为当前结点.
    }
    return 0;
}
```

例题1：模板题

此处需要注意的，下标从0开始的字符串时，进行匹配的时还需判断一下下标是否越界了。
写法1：

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=5e5+5,M=1e6+5,inf=0x3f3f3f3f,mod=1e9+7;
#define mst(a) memset(a,0,sizeof a)
#define lx x<<1
#define rx x<<1|1
#define reg register
```

```
#define PII pair<int,int>
#define fi first
#define se second
char a[N];
int len[N],fail[N],cnt=1,cur,num[N],ans,tri[N][26],u;
int get_fail(int x,int i){
    while(i-len[x]-1<0||a[i-len[x]-1]!=a[i])
        x=fail[x];
    return x;
}
int main(){
    scanf("%s",a);
    len[1]=-1,fail[0]=1;//初始化奇根，偶根.
    int n=strlen(a);
    for(int i=0;i<n;i++){ //求回文子串
        if(i>0) a[i]=(a[i]+ans-97)%26+97; //依题意.
         u=get_fail(cur,i); //找到最长匹配回文后缀的结点.
        if(!tri[u][a[i]-'a']){  //如果该边没有该儿子结点,则新建一个儿子结点.
            len[++cnt]=len[u]+2;//先更新长度,使后面找fail指针更快.
            fail[cnt]=tri[get_fail(fail[u],i)][a[i]-'a']; //先更新fail指针，若先更
新结点会 指向自己.
            tri[u][a[i]-'a']=cnt; //建立结点.
            num[cnt]=num[fail[cnt]]+1;//更新回文后缀个数.
        }
        cur=tri[u][a[i]-'a'];//cur移动到当前结点.
        ans=num[cur];//输出答案.
        printf("%d ",ans);
    }
    return 0;
}
```

写法2:

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N=5e5+5,M=1e6+5,inf=0x3f3f3f3f,mod=1e9+7;
#define mst(a) memset(a,0,sizeof a)
#define lx x<<1
#define rx x<<1|1
#define reg register
#define PII pair<int,int>
#define fi first
#define se second
char a[N];
int ch[N][26],len[N],fail[N],cnt=1,last,ans,num[N];
//num[i] 结点i的回文后缀个数.
int main(){
    scanf("%s",a);
    int n=strlen(a);
    fail[0]=1,len[1]=-1;
    for(int i=0;i<n;i++){
        if(a[i]>0) a[i]=(a[i]+ans-97)%26+97;
        while(i-len[last]-1<0||a[i-len[last]-1]!=a[i]) last=fail[last];
        if(!ch[last][a[i]-'a']){
             len[++cnt]=len[last]+2;
             int j=fail[last];  // 此步是为了求 当前子串的fail指针.
```

```
            while(a[i-len[j]-1]!=a[i]) j=fail[j];
            fail[cnt]=ch[j][a[i]-'a'];//求出fail指针.
            ch[last][a[i]-'a']=cnt;//插入该结点.
            num[cnt]=num[fail[cnt]]+1; //更新num.
        }
        last=ch[last][a[i]-'a'];//last变为当前结点.
        ans=num[last];
        printf("%d ",ans);
    }
    return 0;
}
```