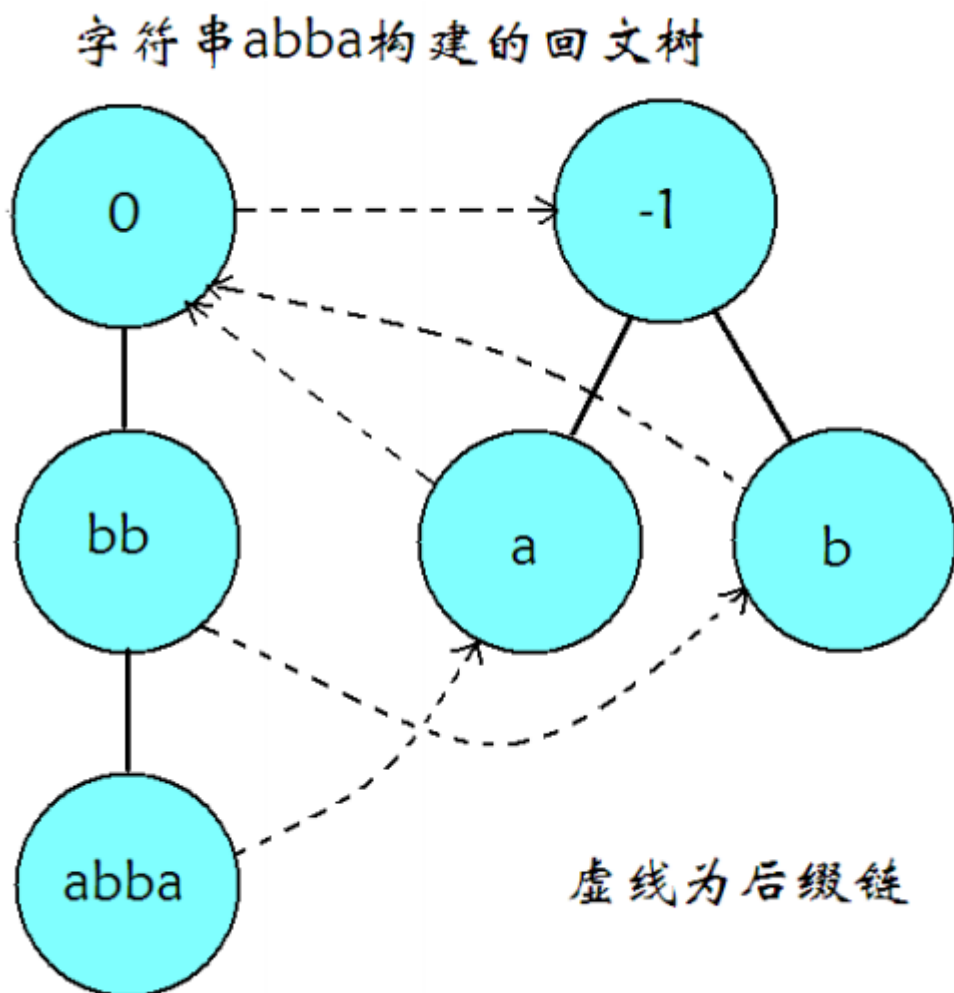


回文树

结构

回文树大概长这样：



功能

假设我们有一个串 S 下标从 0 开始，则回文树能做到如下几点：

1. 求串 S 前缀 $0 \sim i$ 内本质不同回文串的个数（两个串长度不同或者长度相同但至少有一个字符不同便是本质不同）。
2. 求串 S 内每一个本质不同回文串出现的次数。
3. 求串 S 内回文串的个数（其实就是 S_1 和 S_2 结合起来）。
4. 求以下标 i 结尾的回文串的个数。

模板

```
const int MAXN = 100005 ;
const int N = 26 ;

struct Palindromic_Tree {
    //cnt最后count一下之后是那个节点代表的回文串出现的次数
    int next[MAXN][N] ;//next指针[]next指针和字典树类似，指向的串为当前串两端加上同一个字符构成
    int fail[MAXN] ;//fail指针，失配后跳转到fail指针指向的节点
    int cnt[MAXN] ; //表示节点i表示的本质不同的串的个数（建树时求出的不是完全的，最后count()函数跑一遍以后才是正确的）
    int num[MAXN] ; //表示以节点i表示的最长回文串的最右端点为回文串结尾的回文串个数
    int len[MAXN] ;//len[i]表示节点i表示的回文串的长度（一个节点表示一个回文串）
    int S[MAXN] ;//存放添加的字符
    int last ;//指向新添加一个字母后所形成的最长回文串表示的节点。
    int n ;//表示添加的字符个数。
    int p ;//表示添加的节点个数。

    int newnode ( int l ) { //新建节点
        for ( int i = 0 ; i < N ; ++ i ) next[p][i] = 0 ;
        cnt[p] = 0 ;
        num[p] = 0 ;
        len[p] = l ;
        return p ++ ;
    }

    void init () { //初始化
        p = 0 ;
        newnode ( 0 ) ;
        newnode ( -1 ) ;
        last = 0 ;
        n = 0 ;
        S[n] = -1 ;//开头放一个字符集中没有的字符，减少特判
        fail[0] = 1 ;
    }

    int get_fail ( int x ) { //和KMP一样，失配后找一个尽量最长的
        while ( S[n - len[x] - 1] != S[n] ) x = fail[x] ;
        return x ;
    }

    void add ( int c ) {
        c -= 'a' ;
        S[++ n] = c ;
        int cur = get_fail ( last ) ;//通过上一个回文串找这个回文串的匹配位置
        if ( !next[cur][c] ) { //如果这个回文串没有出现过，说明出现了一个新的本质不同的回文串
            int now = newnode ( len[cur] + 2 ) ; //新建节点
```

```

        fail[now] = next[get_fail ( fail[cur] )][c] ;//和AC自动机一样建
立fail指针，以便失配后跳转
        next[cur][c] = now ;
        num[now] = num[fail[now]] + 1 ;
    }
    last = next[cur][c] ;
    cnt[last] ++ ;
}

void count () {
    for ( int i = p - 1 ; i >= 0 ; -- i ) cnt[fail[i]] += cnt[i] ;
    //父亲累加儿子的cnt[]因为如果fail[v]=u[]则u一定是v的子回文串！
}
};

```

例题

P5496 【模板】回文自动机 (PAM)

题意：给定一个字符串 s ，保证每个字符为小写字母。对于 s 的每个位置，请求出以该位置结尾的回文子串个数。

题解：理解好模板中的 $last$ 和 $num[]$ 的意义即可做。

评价：实现了功能 4

代码

```

#include<bits/stdc++.h>
using namespace std;
const int N=5e5+5;
struct Palindromic_Tree{
    int nxt[N][26],fail[N],cnt[N],num[N],len[N],S[N],last,n,p;
    int newnode(int l){
        memset(nxt[p],0,sizeof(nxt[p]));
        cnt[p]=num[p]=0;
        len[p]=l;
        return p++;
    }
    void init(){
        p=0;
        newnode(0);
        newnode(-1);
        last=0;
        n=0;
        S[n]=-1;
        fail[0]=1;
    }
    int get_fail(int x){

```

```
while(S[n-len[x]-1]!=S[n]) x=fail[x];
return x;
}
void add(int c){
    c-='a';
    S[++n]=c;
    int cur=get_fail(last);
    if(!nxt[cur][c]){
        int now=newnode(len[cur]+2);
        fail[now]=nxt[get_fail(fail[cur])][c];
        nxt[cur][c]=now;
        num[now]=num[fail[now]]+1;
    }
    last=nxt[cur][c];
    cnt[last]++;
}
void count(){
    for(int i=p-1;i>=0;i--) cnt[fail[i]]+=cnt[i];
}
}P;
char s[N];
int main(){
    scanf("%s",s);
    int len=strlen(s);
    P.init();
    P.add(s[0]);
    int k=P.num[P.last];
    printf("%d",k);
    for(int i=1;i<len;i++){
        P.add((s[i]-97+k)%26+97);
        k=P.num[P.last];
        printf(" %d",k);
    }
    return 0;
}
```

P3649 [APIO2014]回文串

题意：给你一个由小写拉丁字母组成的字符串 ss 我们定义 ss 的一个子串的存在值为这个子串在 ss 中出现的次数乘以这个子串的长度。对于给你的这个字符串 ss 求所有回文子串中的最大存在值。

题解：利用模板中的 $len[]$ 和 $cnt[]$ 即可。

评价：利用了 功能 2

代码


```
#include<bits/stdc++.h>
using namespace std;
```

```

const int N=3e5+5;
struct Palindromic_Tree{
    int nxt[N][26],fail[N],cnt[N],num[N],len[N],S[N],last,n,p;
    int newnode(int l){
        memset(nxt[p],0,sizeof(nxt[p]));
        cnt[p]=num[p]=0;
        len[p]=l;
        return p++;
    }
    void init(){
        p=0;
        newnode(0);
        newnode(-1);
        last=0;
        n=0;
        S[n]=-1;
        fail[0]=1;
    }
    int get_fail(int x){
        while(S[n-len[x]-1]!=S[n]) x=fail[x];
        return x;
    }
    void add(int c){
        c-='a';
        S[++n]=c;
        int cur=get_fail(last);
        if(!nxt[cur][c]){
            int now=newnode(len[cur]+2);
            fail[now]=nxt[get_fail(fail[cur])][c];
            nxt[cur][c]=now;
            num[now]=num[fail[now]]+1;
        }
        last=nxt[cur][c];
        cnt[last]++;
    }
    void count(){
        for(int i=p-1;i>=0;i--) cnt[fail[i]]+=cnt[i];
    }
}P;
char s[N];
int main(){
    P.init();
    scanf("%s",s);
    int len=strlen(s);
    for(int i=0;i<len;i++) P.add(s[i]);
    P.count();
    long long maxx=0;
    for(int i=0;i<P.p;i++){
        maxx=max(maxx,1ll*P.len[i]*P.cnt[i]);
    }
    printf("%lld",maxx);
}

```

```
return 0;  
}
```

From: <https://wiki.cvbbacm.com/> - CVBB ACM Team
Permanent link: https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:lgwza:%E5%9B%9E%E6%96%87%E6%A0%91&rev=1601726480 
Last update: **2020/10/03 20:01**