回文自动机（姬）

算法思想

回文自动机是一种可以存储一个串中所有回文子串的数据结构。

与其他自动机类似，回文自动机也有转移边和失配指针 $fail$ 组成，每个结点都代表一个回文子串。

对于奇数和偶数的回文串，我们分别建立一棵树。一个节点的 $fail$ 指针指向的是这个节点所代表的回文串的最长回文后缀所对应的节点。转移边代表在原结点代表的回文串前后各加一个相同的字符。还需要对每个结点维护回文子串的长度 len

构造自动机

两个初始状态分别代表 $-1,0$ 的回文串，分别叫做奇根和偶根。偶根的 $fail$ 指针指向奇根，我们不用管奇根的 $fail$ 指针，因为奇根不可能失配（因为最短的奇数回文串是单个字符串，这显然存在，不可能失配）。

插入字符

现在假设已经构造完前 $p-1$ 个字符的回文自动机后，现在插入位置为 p 的字符。

从上一个字符结尾的最长回文子串对应的结点开始，一直沿着 $fail$ 走，直到找到一个结点满足 $s_{p}=s_{p-len-1}$ 就说明这个位置和这个回文串之前的一个字符一样，所以可以构成新的回文串。如果到最后都没找到 len 为 -1，正好是 $s_{p}=s_{p}$ 说明没有这个结点，需要特判新建。

我们还需要求出新建的结点的 $fail$ 指针，具体方法和上面的过程类似，不断跳转 $fail$ 指针，从不加新字符的那个回文串开始，就可以找到另一个带着两个新字符的回文串，然后把 $fail$ 指针指到这个字符串即可。如果 $fail$ 没匹配到，那么将它连向长度为 0 的那个结点，显然可以（因为这是所有节点的后缀）。

对于一个字符串 s 它的本质不同的回文子串个数最多只有 $|s|$ 个（数学归纳法）。所以转移状态数也是 $O(|s|)$ 的。

算法实现

```
//len数组长度 fail数组存失配以后跳转到最长后缀回文串的结点
//cnt数组存回文字符串在整个字符串中出现多少次（需要倒着求和才对）
//num[i]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。
//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。
struct PAM {
    char s[maxn];
    int
len[maxn],n,num[maxn],fail[maxn],last,cur,pos,trie[maxn][26],tot,cnt[maxn];
    void init() {
        memset(s,0,sizeof(s));
```

```
        memset(len,0,sizeof(len));
        memset(num,0,sizeof(num));
        memset(fail,0,sizeof(fail));
        memset(trie,0,sizeof(trie));
        memset(cnt,0,sizeof(cnt));
        n=0; last=0; cur=0;
        pos=0; tot=1;
        fail[0]=1; len[1]=-1;
    }
    int getfail(int x,int i) {
        while(i-len[x]-1<0||s[i-len[x]-1]!=s[i]) x=fail[x];
        return x;
    }
    void solve() {
        for(int i=0; i<=n-1; i++) {
            pos=getfail(cur,i);
            if(!trie[pos][s[i]-'a']) {
                fail[++tot]=trie[getfail(fail[pos],i)][s[i]-'a'];
                trie[pos][s[i]-'a']=tot;
                len[tot]=len[pos]+2;
                num[tot]=num[fail[tot]]+1;
            }
            cur=trie[pos][s[i]-'a'];
            cnt[cur]++;
        }
        for(int i=tot;i>=0;i--){
            cnt[fail[i]]+=cnt[i];
        }
        for(int i=tot;i>=0;i--){
            if(1ll*cnt[i]*len[i]>maxv) maxv=1ll*cnt[i]*len[i];
        }
    }
} p;
```

代码练习

1.询问本质不同回文子串的个数

状态数 -2 （上面代码 $tot-1$ 即可），因为要排除奇根和欧根。

2.回文子串出现次数

上面的 cnt 数组，类似于 AC 自动机的做法，我们 $fail$ 指向的那个节点的字符串一定在现在这个结点的字符串中出现，又因为编号一定是 $fail$ 更小，所以倒着求一遍就好了。

3.最小回文划分

给定一个字符串 $s(1≤|s|≤10^{5})$ 求最小的 k 使得存在 $s_{1},...,s_{k}$ 满足均为回文串，且依次连接后得到的字符串是 s

考虑 dp 记 $dp[i]$ 表示 s 长度为 i 的前缀的最小划分数，考虑只需要枚举第 i 个字符结尾的所有回文串，有 $dp[i]=1+min_{s[j+1]...i ok}(dp[j])$ 一个字符串最多有 $O(n^{2})$ 个回文子串，所以时间复杂度是 $O(n^{2})$ 不能通过。考虑优化。

我们有 s 和 t 两个字符串，周期定义为一直循环出现即可，不一定正好出现完整个 $border$ 定义为前缀和后缀一样且不是原串。周期和 $border$ 的关系 t 是 s 的 $border$ 当且仅当 $|s|-|t|$ 是 s 的周期。

我们还有 t 是回文串 s 的后缀，则 t 是 s 的 $border$ 当且仅当 t 是回文串。画个图，显然。

我们还有 t 是回文串 s 的 $border(|s|≤2|t|)$ s 是回文串当且仅当 t 是回文串。如果 s 是回文串，显然根据上面的定理 t 是回文串，如果 t 是回文串，画了一下挺抽象的当然也是显然的。

还有 t 是回文串 s 的 $border$ 则 $|s|-|t|$ 是 s 的周期 $|s|-|t|$ 为 s 的最小周期，当且仅当 t 是 s 的最长回文真后缀。

还有 x 是一个回文串 y 是 x 的最长回文真后缀 z 是 y 的最长回文真后缀。令 u,v 分别为满足 $x=uy,y=vz$ 的字符串，则有：

$1.|u|≥|v|$

证明 $|u|=|x|-|y|$ 是 x 的最小周期，同样也是 y 的周期，又因为 v 是 y 的最小周期，所以 $|u|≥|v|$

$2.$ 如果 $|u|=|v|$ $|u|=|z|$

$3.$ 如果 $|u|=|v|$ $u=v$

4.https://www.luogu.com.cn/problem/P4555

题意

给定一个字符串 S 求最长双回文子串 T 双回文串即可以拆成两个非空的回文串。

题解

显然 T 拆成两个小回文串，是有分割点的，我们枚举分割点，然后相当于求一个在这个位置截止的最长回文串和这个位置开始的最长回文串，其实我们可以倒着求一下，所以只求这个位置结束的最长回文串就可以了。如果只求一边比如 $aaaaa$ 它不会存第二个位置之后的最长回文串，也就是只求一边是算不全的，还是得算两边的。

```
#include <bits/stdc++.h>
using namespace std;
const int maxn=100100;
struct PAM {
    char s[maxn];
    int
len[maxn],n,num[maxn],fail[maxn],last,cur,pos,trie[maxn][26],tot,m[maxn];
    void init() {
        memset(s,0,sizeof(s));
```

```
        memset(len,0,sizeof(len));
        memset(num,0,sizeof(num));
        memset(fail,0,sizeof(fail));
        memset(trie,0,sizeof(trie));
        memset(m,0,sizeof(m));
        n=0;last=0;cur=0;pos=0;tot=1;
        fail[0]=1;len[1]=-1;
    }
    int getfail(int x,int i) {
        while(i-len[x]-1<0||s[i-len[x]-1]!=s[i])x=fail[x];
        return x;
    }
    void solve() {
        for(int i=0; i<=n-1; i++) {
            pos=getfail(cur,i);
            if(!trie[pos][s[i]-'a']) {
                fail[++tot]=trie[getfail(fail[pos],i)][s[i]-'a'];
                trie[pos][s[i]-'a']=tot;
                len[tot]=len[pos]+2;
                num[tot]=num[fail[tot]]+1;
            }
            cur=trie[pos][s[i]-'a'];
            m[i]=len[cur];
        }
    }
} p,q;

int main() {
    p.init();q.init();
    scanf("%s",p.s);
    p.n=strlen(p.s);
    q.n=p.n;
    for(int i=0;i<p.n;i++){
        q.s[i]=p.s[p.n-1-i];
    }
    p.solve();
    q.solve();
    int ans=0;
    for(int i=0;i<p.n-1;i++){
        if(p.m[i]+q.m[p.n-2-i]>ans){
            ans=p.m[i]+q.m[p.n-2-i];
        }
    }
    printf("%d",ans);
    return 0;
}
```

5.https://www.luogu.com.cn/problem/P4287

题意

给一个字符串 S 让求一个双倍回文串，这里的双倍回文串要求必须是 $RR^{T}RR^{T}$ 形式的。

题解

我们发现这样的子串长度一定是 4 的倍数，并且不超过自己一半长度的最长的失配串的长度一定正好是一半。于是我们重新搞一个 f 数组，记录不超过自己一半长度的最长失配串，如果在找 $fail$ 的时候，发现不到一半，直接赋值成 $fail$ 指针就好了，不然我们需要走 $fail$ 指针，直到长度不到一半为止。

```
void solve() {
        for(int i=0; i<=n-1; i++) {
            pos=getfail(cur,i);
            if(!trie[pos][s[i]-'a']) {
                fail[++tot]=trie[getfail(fail[pos],i)][s[i]-'a'];
                trie[pos][s[i]-'a']=tot;
                len[tot]=len[pos]+2;
                num[tot]=num[fail[tot]]+1;
                if(len[fail[tot]]<=len[tot]>>1) f[tot]=fail[tot];
                else{
                    int poss=f[pos];//仿照上面的
                    while(len[poss]+2>len[tot]/2||s[i]!=s[i-len[poss]-1])
poss=fail[poss];//仿照上面的函数len[poss]是没加上两边的串所以再加上两边的大于一半就得跑fail指针
                    f[tot]=trie[poss][s[i]-'a'];//仿照上面的
                }
            }
            cur=trie[pos][s[i]-'a'];
        }
    }

int main() {
    p.init();
    int nn;
    scanf("%d",&nn);
    scanf("%s",p.s);
    p.n=strlen(p.s);
    p.solve();
    for(int i=2;i<=p.tot;i++){
        if(p.len[i]%4==0&&p.len[p.f[i]]==p.len[i]>>1){//长度正好是一半
            p.Ans=max(p.Ans,p.len[i]);
        }
    }
    printf("%d\n",p.Ans);
    return 0;
}
```