KMP系列题目。

1.KMP最常用的用法：查找一个字符串在另一个字符串中的位置。复杂度O（m+n)

```
#include<bits/stdc++.h>
using namespace std;
string a,b;
const int N=1e6+5;
int f[N];
void fun(string a){
    int l=a.size(),j=0;
    for(int i=1;i<l;i++)
    {
        while(j&&a[i]!=a[j]) j=f[j-1];
        if(a[i]==a[j]) j++;
        f[i]=j;
    }
}
void kmp(string a,string b){//a与b进行kmp匹配
     fun(b);
     int l=a.size(),lb=b.size(),j=0;
     for(int i=0;i<l;i++)
     {
         while(j&&a[i]!=b[j]) j=f[j-1];
         if(a[i]==b[j]) j++;
         if(j==lb)
         {
             cout<<i-lb+2<<endl;//输出子串在字符串中的位置. (这里答案下标要求为1开始所以是+2)
                j=f[j-1];//回退 重新匹配
         }
    }
    for(int i=0;i<lb;i++)
        printf(i==lb-1?"%d\n":"%d ",f[i]);
}
int main(){
    cin>>a>>b;
    kmp(a,b);
    return 0;
}
```

用法2：回文串的查找(用于从字符串头开始或着到尾结束的字符串十分方便)

不适用于中间是回文串的字符串，比如：ABCBD。

例题：CF1325D. Prefix-Suffix Palindrome (Hard version)

题意：给定字符串，求由字串前缀和后缀组成的最大回文字符串，且长度不大于字符串本身.

思路1：对于easy version ： 可以选择暴力。双向指针找到最大相同前后缀strL，strR。在对中间的一段字符串进行暴力枚举找到最长的回文串mid。最后的答案为 strL+mid+strR 下面上代码。

```
xxxxxxxxxx
#include<bits/stdc++.h>
using namespace std;
int t,l,r;
string s,A;
bool C(string s)
{
    for(int i=0;i<s.size()/2;i++)
        if(s[i]!=s[s.size()-1-i])
            return false;
    return true;
}
int main()
{
    cin>>t;
    while(t--&&cin>>s)
    {
        if(C(s))
        {
            cout<<s<<'\n';
            continue;
        }
        l=0,r=s.size()-1;
        while(l<r&&s[l]==s[r])
            l++,r--;
        A="";
        for(int i=1;i<=r-l;i++) //找到中间的回文串 因为之前已经判断出不可能全部为回文串所以 i最大为r-l 而不是r-l+1
            if(C(s.substr(l,i)))
```

```
                A=s.substr(l,i);
            else if(C(s.substr(r-i+1,i)))
                A=s.substr(r-i+1,i);
        for(int i=0;i<l;i++)
            cout<<s[i];
        cout<<A;
        for(int i=r+1;i<s.size();i++)
            cout<<s[i];
        cout<<'\n';
    }
    return 0;
}
```

思路2：对于hard version 由于数据非常大 。所以对于mid的求法需要优化。这里用到KMP，方法为将 中间的字符串mid翻转过来 然后变成 mid+#+rev 利用KMP求出与前缀相连的最大回文串。 再求rev+#+mid 求出于后缀相连的最大回文串 举个例子，假设mid=ABACDD ,prv1=ABA，prv=DD, mid+#+rev=ABACDD#DDCABA 最大前后缀为3所以最大回文串长度为3ABA， rev+#+mid=DDCABA#ABACDD 最大前后缀为2所以最大回文串长度为2DD 下面上代码. 类似于将字符串的第一个与最后一个匹配，第二个与倒数第二个匹配……

```
x
#include <bits/stdc++.h>
using namespace std;
int pi[2000001]; //最长公共前后缀表（pi[i]表示字符串s从0到i的最长公共前后缀      那一部分一定是回文的
string preF(string s) {   //求最长公共前后缀.      举个例子 dfdce 变形为 dfdce#ecdfd 最长公共前后缀为dfd
    int n = (int)s.size(), j = 0;//显然pi[0]=0                                          //ecdfd#dfdce 最长公共前后缀为1
    for (int i = 1; i < n; ++i) {//求pi[1]到pi[n-1]  //i为后缀的最后一个下标 从字符串的第2个字母开始
        while (j && s[i] != s[j]) {   //这里的j既是当前最大的匹配长度，又是当前 前缀的最后一个下标.
            j = pi[j - 1]; //j退回到前一个字符的最大匹配长度
        }
        if (s[i] == s[j]) {
            ++j;
        }
        pi[i] = j;
    }
    /*for(int i=0;i<n;i++)
        printf("pi[%d]=%d\n",i,pi[i]);
    //printf("j=%d\n",j);
    */
    return s.substr(0, j);
}
int main() {
    ios_base::sync_with_stdio(0);
      cin.tie(0);
    int t;
```

```
    cin >> t;
    while (t--) {
        string s;
        cin >> s;
        int n = s.size();
        int j = n - 1, i = 0;
        while (i < j && s[i] == s[j]) {
            ++i;
            --j;
        }
        string mid = s.substr(i, j - i + 1); //中间的字符串.
        string rev = mid;
        reverse(rev.begin(), rev.end());//翻转过来
        string ans1 = preF(mid + '#' + rev);
        string ans2 = preF(rev + '#' + mid);
        //cout<<"mid="<<mid<<endl;
        //cout<<"rev="<<rev<<endl;
        //cout<<"mid#rev="<<mid<<"#"<<rev<<endl;
        //cout<<"ans1="<<ans1<<endl;
        ///cout<<"ans2="<<ans2<<endl;
        cout << s.substr(0, i) << (ans1.size() > ans2.size() ? ans1 : ans2) << s.substr(j + 1) << '\n';
    }
}
```

用法3：求字符串的最小循环节和循环周期.

结论：最小循环长度为 n-f[n] f[i]为最大前后缀表。注意这里的循环不是一定满足严格循环abcabcabc式的，bcabcabc也看作是循环的，最小循环节长度也为3（abc) 这里的结论不作证明。下面上代码。

从0开始的字符串

```
xxxxxxxxxx
#include<bits/stdc++.h>
using namespace std;
const int N=1e6+5;
int f[N],n,j;
int main(){
    string a;
    cin>>n>>a;
    for(int i=1;i<n;i++)
    {
```

```
        while(j&&a[j]!=a[i]) j=f[j-1];
        if(a[j]==a[i]) j++;
        f[i]=j;
    }
    cout<<n-f[n-1]<<endl;
    printf("循环周期为%d\n",n/(n-f[n-1]));
    return 0;
}
```

从1开始的字符串

```
xxxxxxxxxx
#include<bits/stdc++.h>
using namespace std;
const int N=1e6+5;
int f[N],j,n;
char a[N];
int main(){
    f[1]=0;
    cin>>n>>(a+1);
    for(int i=2;i<=n;i++)
    {
        while(j&&a[i]!=a[j+1]) j=f[j];
        if(a[i]==a[j+1]) j++;
        f[i]=j;
    }
    cout<<n-f[n]<<endl;
    printf("循环周期为%d\n",n/(n-f[n]));
    return 0;
}
```