hash

```
namespace HASH
{
#define ULL unsigned long long
ULL h[MAXN];
int x;
int lenth;
ULL p1, p2;
ULL xp1[MAXN], xp2[MAXN], xp[MAXN];
char s[MAXN];
inline void init(const char *str)
{
        memcpy(s, str, sizeof(s));
        lenth = strlen(s);
        x = 13331;
        p1 = 998244353;
        p2 = 1e9 + 7;
        #ifdef ENABLE_DOUBLE_HASH
        xp1[0] = xp2[0] = 1;
        for(int i = 1; i <= lenth; ++i)
                xp1[i] = xp1[i - 1] * x % p1;
        for(int i = 1; i <= lenth; ++i)
                xp2[i] = xp2[i - 1] * x % p2;
        #else
        xp[0] = 1;
        for(int i = 1; i <= lenth; ++i)
                xp[i] = xp[i - 1] * x;
        #endif
}
ULL hash()
{
        ULL res1 = 0, res2 = 0;
        h[lenth] = 0;
        for(int j = lenth - 1; j >= 0; --j)
        {
        #ifdef ENABLE_DOUBLE_HASH
                res1 = (res1 * x + s[j]) % p1;
                res2 = (res2 * x + s[j]) % p2;
                h[j] = (res1 << 32) | res2;
        #else
                res1 = res1 * x + s[j];
                h[j] = res1;
        #endif
        }
        return h[0];
}
ULL get_substring_hash(int left, int right)
{
        int len = right - left;
#ifdef ENABLE_DOUBLE_HASH
        unsigned int mask32 = ~(0u);
        ULL left1 = h[left] >> 32, right1 = h[right] >> 32;
        ULL left2 = h[left] & mask32, right2 = h[right] & mask32;
        return (((left1 - right1 * xp1[len] % p1 + p1) % p1) << 32) | (((left2 - right2 * xp2[len] % p2 + p2
#else
        return h[left] - h[right] * xp1[len];
#endif
```

```
}
}
```

Trie树

$MAXN = \Sigma s_i$

```
namespace TRIE
{
int trie[MAXN][65], cnt[MAXN];
int tot = 1;
char s[MAXN];
int n, m;
//获取每个字符对应数字
int getnum(char c)
{
        if('a' <= c && c <= 'z')
                return c - 'a' + 26;
        else if('0' <= c && c <= '9')
                return c - '0' + 52;
        else if('A' <= c && c <= 'Z')
                return c - 'A';
}
//插入一个字符串
void insert(char *s)
{
        int len = strlen(s);
        int p = 1;
        for(int k = 0; k < len; ++k)
        {
                int ch = getnum(s[k]);
                if(trie[p][ch] == 0)
                        trie[p][ch] = ++tot;
                p = trie[p][ch];
                ++cnt[p];
        }
}
//查询字符串前缀出现次数
int search(char *s)
{
        int len = strlen(s);
        int p = 1;
        for(int k = 0; k < len; ++k)
        {
                int ch = getnum(s[k]);
                p = trie[p][ch];
                if(p == 0)
                        return 0;
        }
        return cnt[p];
}
//初始化（不用memset防止MLE）
void init(void)
{
        for(int i = 0; i <= tot; ++i)
        {
                for(int j = 0; j < 65; ++j)
                {
                        trie[i][j] = 0;
                }
                cnt[i] = 0;
        }
        tot = 1;
}
}
```

AC自动机（copy）

```
#include<bits/stdc++.h>
#define maxn 1000001
using namespace std;
struct kkk{
        int son[26],flag,fail;
}trie[maxn];
int n,cnt;
char s[1000001];
queue<int >q;
void insert(char* s){
        int u=1,len=strlen(s);
        for(int i=0;i<len;i++){
                int v=s[i]-'a';
                if(!trie[u].son[v])trie[u].son[v]=++cnt;
                u=trie[u].son[v];
        }
        trie[u].flag++;
}
void getFail(){
        for(int i=0;i<26;i++)trie[0].son[i]=1;                  //初始化0的所有儿子都是1
        q.push(1);trie[1].fail=0;                               //将根压入队列
        while(!q.empty()){
                int u=q.front();q.pop();
                for(int i=0;i<26;i++){                          //遍历所有儿子
                        int v=trie[u].son[i];                   //处理u的i儿子的fail，这样就可以不用记父亲了
                        int Fail=trie[u].fail;                  //就是fafail，trie[Fail].son[i]就是和v值相同的
                        if(!v){trie[u].son[i]=trie[Fail].son[i];continue;}      //不存在该节点，第二种情况
                        trie[v].fail=trie[Fail].son[i]; //第三种情况，直接指就可以了
                        q.push(v);                                                      //存在实节点才压入队列
                }
        }
}
int query(char* s){
        int u=1,ans=0,len=strlen(s);
        for(int i=0;i<len;i++){
                int v=s[i]-'a';
                int k=trie[u].son[v];           //跳Fail
                while(k>1&&trie[k].flag!=-1){   //经过就不统计了
                        ans+=trie[k].flag,trie[k].flag=-1;      //累加上这个位置的模式串个数，标记已经过
                        k=trie[k].fail;                 //继续跳Fail
                }
                u=trie[u].son[v];                       //到下一个儿子
        }
        return ans;
}
int main(){
        cnt=1;scanf("%d",&n);
        for(int i=1;i<=n;i++){
                scanf("%s",s);
                insert(s);
        }
        getFail();
        scanf("%s",s);
        printf("%d\n",query(s));
        return 0;
}
```

后缀数组(copy)

```
#include<iostream>
#include<cstdio>
#include<cstring>
#define rint register int
#define inv inline void
#define ini inline int
#define maxn 1000050
using namespace std;
char s[maxn];
int y[maxn],x[maxn],c[maxn],sa[maxn],rk[maxn],height[maxn],wt[30];
int n,m;
inv putout(int x)
{
    if(!x) {putchar(48);return;}
    rint l=0;
    while(x) wt[++l]=x%10,x/=10;
    while(l) putchar(wt[l--]+48);
}
inv get_SA()
{
        for (rint i=1;i<=n;++i) ++c[x[i]=s[i]];
        //c数组是桶
        //x[i]是第i个元素的第一关键字
    for (rint i=2;i<=m;++i) c[i]+=c[i-1];
    //做c的前缀和，我们就可以得出每个关键字最多是在第几名
    for (rint i=n;i>=1;--i) sa[c[x[i]]--]=i;
    for (rint k=1;k<=n;k<<=1)
    {
        rint num=0;
        for (rint i=n-k+1;i<=n;++i) y[++num]=i;
        //y[i]表示第二关键字排名为i的数，第一关键字的位置
                //第n-k+1到第n位是没有第二关键字的 所以排名在最前面
        for (rint i=1;i<=n;++i) if (sa[i]>k) y[++num]=sa[i]-k;
        //排名为i的数 在数组中是否在第k位以后
                //如果满足(sa[i]>k) 那么它可以作为别人的第二关键字，就把它的第一关键字的位置添加进y就行了
                //所以i枚举的是第二关键字的排名，第二关键字靠前的先入队
        for (rint i=1;i<=m;++i) c[i]=0;
        //初始化c桶
        for (rint i=1;i<=n;++i) ++c[x[i]];
        //因为上一次循环已经算出了这次的第一关键字 所以直接加就行了
        for (rint i=2;i<=m;++i) c[i]+=c[i-1];//第一关键字排名为1~i的数有多少个
        for (rint i=n;i>=1;--i) sa[c[x[y[i]]]--]=y[i],y[i]=0;
        //因为y的顺序是按照第二关键字的顺序来排的
        //第二关键字靠后的，在同一个第一关键字桶中排名越靠后
        //基数排序
        swap(x,y);
                //这里不用想太多，因为要生成新的x时要用到旧的，就把旧的复制下来，没别的意思
        x[sa[1]]=1;num=1;
        for (rint i=2;i<=n;++i)
            x[sa[i]]=(y[sa[i]]==y[sa[i-1]] && y[sa[i]+k]==y[sa[i-1]+k]) ? num : ++num;
        //因为sa[i]已经排好序了，所以可以按排名枚举，生成下一次的第一关键字
        if (num==n) break;
        m=num;
        //这里就不用那个122了，因为都有新的编号了
    }
    for (rint i=1;i<=n;++i) putout(sa[i]),putchar(' ');
}
inv get_height()
```

```
{
    rint k=0;
    for (rint i=1;i<=n;++i) rk[sa[i]]=i;
    for (rint i=1;i<=n;++i)
    {
        if (rk[i]==1) continue;//第一名height为0
        if (k) --k;//h[i]>=h[i-1]+1;
        rint j=sa[rk[i]-1];
        while (j+k<=n && i+k<=n && s[i+k]==s[j+k]) ++k;
        height[rk[i]]=k;//h[i]=height[rk[i]];
    }
    putchar(10);for (rint i=1;i<=n;++i) putout(height[i]),putchar(' ');
}
int main()
{
    gets(s+1);
    n=strlen(s+1);m=122;
        //因为这个题不读入n和m所以要自己设
        //n表示原字符串长度，m表示字符个数，ascll('z')=122
        //我们第一次读入字符直接不用转化，按原来的ascll码来就可以了
        //因为转化数字和大小写字母还得分类讨论，怪麻烦的
    get_SA();
    //get_height();
}
```

后缀自动机(copy)

```
struct NODE
{
    int ch[26];
    int len,fa;
    NODE(){memset(ch,0,sizeof(ch));len=0;}
}dian[MAXN<<1];
int las=1,tot=1;
void add(int c)
{
    int p=las;int np=las=++tot;
    dian[np].len=dian[p].len+1;
    for(;p&&!dian[p].ch[c];p=dian[p].fa)dian[p].ch[c]=np;
    if(!p)dian[np].fa=1;//以上为case 1
    else
    {
        int q=dian[p].ch[c];
        if(dian[q].len==dian[p].len+1)dian[np].fa=q;//以上为case 2
        else
        {
            int nq=++tot;dian[nq]=dian[q];
            dian[nq].len=dian[p].len+1;
            dian[q].fa=dian[np].fa=nq;
            for(;p&&dian[p].ch[c]==q;p=dian[p].fa)dian[p].ch[c]=nq;//以上为case 3
        }
    }
}
char s[MAXN];int len;
int main()
{
    scanf("%s",s);len=strlen(s);
    for(int i=0;i<len;i++)add(s[i]-'a');
}
```

回文自动机(copy)

```
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;

const int N = 2e6 + 5;
struct PAM_Trie
{
        int ch[26];
        int fail, len, num;
};
struct PAM
{
        PAM_Trie b[N];
        int n, length, last, cnt, s[N];
        char c[N];

        PAM()
        {
                b[0].len = 0; b[1].len = -1;
                b[0].fail = 1; b[1].fail = 0;
                last = 0;
                cnt = 1;
        }
        void read()
        {
                scanf("%s", c + 1);
                length = strlen(c + 1);
        }
        int get_fail(int x)
        {
                while(s[n - b[x].len - 1] != s[n])
                {
                        //printf("%d %d %d\n", x, n - b[x].len - 1, b[x].fail);
                        x = b[x].fail;
                }
                return x;
        }
        void insert()
        {
                int p = get_fail(last);
                if(!b[p].ch[s[n]])
                {
                        b[++cnt].len = b[p].len + 2;
                        int tmp = get_fail(b[p].fail);
                        b[cnt].fail = b[tmp].ch[s[n]];
                        b[cnt].num = b[b[cnt].fail].num + 1;
                        b[p].ch[s[n]] = cnt;
                }
                last = b[p].ch[s[n]];
        }
        void solve()
        {
                int k = 0;
                s[0] = 26;
                for(n = 1; n <= length; n++)
                {
                        c[n] = (c[n] - 97 + k) % 26 + 97;
```

```
                        s[n] = c[n] - 'a';
                        insert();
                        printf("%d ", b[last].num);
                        k = b[last].num;
                }
        }
}P;
int main()
{
        P.read();
        P.solve();
        return 0;
}
```