```
ll power(ll a,ll b,ll mod)
{
    ll res=1;
    a=a%mod;
    while(b>0)
    {
        if((b&1))
            res=(res*a)%mod;
        b>>=1;
        a=(a*a)%mod;
    }
    return res;
}
ll mulmod(ll a, ll b, ll c) {
  ll x = 0, y = a % c;
  while (b) {
    if (b & 1) x = (x + y) % c;
    y = (y << 1) % c;
    b >>= 1;
  }
  return x % c;
}


bool isPrime(ll n) {
  ll d = n - 1;
  int s = 0;
  while (d % 2 == 0) {
    s++;
    d >>= 1;
  }

  // It's guranteed that these values will work for any number smaller than
3*10**18 (3 and 18 zeros)
  int a[9] = { 2, 3, 5, 7, 11, 13, 17, 19, 23 };
  for(int i = 0; i < 9; i++) {
    bool comp = fastPow(a[i], d, n) != 1;
    if(comp) for(int j = 0; j < s; j++) {
      ll fp = fastPow(a[i], (1LL << (ll)j)*d, n);
      if (fp == n - 1) {
        comp = false;
        break;
      }
    }
    if(comp) return false;
  }
  return true;
}

bool is(ll n)
{
    if(n==2 )return true;
```

```
    if(n==1)return false;
    if(n%2==0)return false;
    for(ll i=3;i*i<=n;i+=2)
    {
        if(n%i==0)
        return false;
    }
    return true;
}


//counting divisors in cube root n
         if(n<=1000000)
        {
            while(spf[n]!=1)
            {
                ll s=spf[n],cnt=0;
                while(spf[n]==s)
                {
                    cnt++;
                    n/=s;
                }
                nof*=(cnt+1);
            }
        }
        else{
        for(ll i=0;i<ll(primes.size());i++)
        {
            ll ele=primes[i];
            if(n%ele==0)
            {
                //cout << ele << "\n";
                ll cnt=0;
                while(n%ele==0)
                {
                    cnt++;n/=ele;
                }
                cnt+=1;
                nof*=cnt;
            }
        }
        if(n>1000000){

         if(n>1000000 && isPrime(n))
        {
            nof*=2;
        }
        else
        {
            if(n==pow(ll(sqrt(n)+0.5),2))
                nof*=3;
            else nof*=4;
```

```
        }
        }
        else
        {
            while(spf[n]!=1)
            {
                ll s=spf[n],cn=0;
                while(spf[n]==s)
                {
                    cn++;n/=s;
                }
                nof=(nof*(cn+1));
            }
        }
        }
        //cout << nof << "\n";
        ll ans=power(tt,nof/2,modd);
        if(nof&1)
            ans=(ans*ll(sqrt(tt)))%modd;


//lca finding


ll deg[50001],parent[50001];
vector<ll>adj[50001];
void degree(ll n)
{
bool visited[n];memset(visited,false,sizeof visited);
  queue<ll>q;
  q.push(1);deg[1]=0;
  while(!q.empty())
  {
        ll node=q.front();q.pop();
        visited[node]=true;
        for(ll i=0;i<adj[node].size();i++)
        {
                if(!visited[adj[node][i]]){
        parent[adj[node][i]]=node;

visited[adj[node][i]]=true;deg[adj[node][i]]=deg[node]+1;q.push(adj[node][i]);}
        }
  }
}


ll dp[no of nodes in tree][max level];
ll lca(ll x,ll y)
{
    ll i;
    if(lvl[x]<lvl[y])//lvl[x] gives the level of node x given the root is at 0
    swap(x,y);
```

```
    ll dist=lvl[x]-lvl[y];
    for(i=19;i>=0;i--)
    {
        if(dist&(1<<i))
        {
            dist-=(1<<i);
            x=dp[x][i];
        }
    }
    if(x==y)
    return x;
    for(i=19;i>=0;i--)
    {
        if(dp[x][i]!=dp[y][i])
        {
            x=dp[x][i];
            y=dp[y][i];
        }
    }
    return dp[x][0];
}

int main()
{
   //before querying we have to set the dp array.
    memset(dp,-1,sizeof dp);
  //apply dfs here to set the level of each node
       for(i=1;i<=19;i++)//19 is the log of n
    {
        for(j=1;j<=n;j++)//n is the number of nodes in the tree
        {
            if(dp[j][i-1]!=-1)
            {
                dp[j][i]=dp[dp[j][i-1]][i-1];
            }
        }
    }
    //now we can make lca queries
}



//euler totient and continued gcd

void calcphi()

{
    phi[1]=1;

for(ll i=2;i<=500000;i++)

phi[i]=i;
```

```
for(ll i=2;i<=500000;i++)

{
        if(phi[i]==i)

         {

                for(ll j=i;j<=500000;j+=i)

                 {

                 phi[j]/=i;

                 phi[j]*=(i-1);

                 }

         }

}

//f is pillai's arithmetical function for continued gcd sum
for(ll i=1;i<=500000;i++)

{

   for(ll j=i;j<=500000;j+=i)

   {

       f[j] = (f[j] + (i*phi[j/i])%mod)%mod;

   }

}

return;

}


//kmp string searching

#include<bits/stdc++.h>
using namespace std;
typedef int ll;

int main()

{
    int lp;
```

```
     while(cin >> lp){
    string p,t;
    cin >> p >> t;
    int lt=t.length();
    int pref[lp];
    pref[0]=0;

     for(int i=1;i<lp;i++)

     {
 int j=pref[i-1];

       while(j>0 && p[i]!=p[j])
j=pref[j-1];

       if(p[i]==p[j])
j++;

       pref[i]=j;

     }

     int i=0,j=0,cnt=0;

     while(i<lt)

     {

     if(t[i]==p[j])
{
i++;j++;
}

     else if(t[i]!=p[j])

{
 if(j>0)
j=pref[j-1];
 else j=0;

     if(t[i]==p[j])
{
i++;j++;}

     else
 i++;

     }

     if(j==lp)
{
```

```
cout << i-lp << "\n";
cnt++;
j=pref[j-1];
}

     }

     if(cnt==0)cout << "\n\n";
    }

return 0;
}
```