目录

素数筛

暴力枚举求素数

枚举每个数，看是否有正整数能整除这个数。复杂度$O(n\sqrt{n})$

```
for(int i=2;i<=n;i++){
    bool g=0;
    for(int j=2;j*j<=i;j++)
        if(i%j==0){g=1;break;}
    if(!g){
        tot++;
        p[tot]=i;
    }
}
```

埃氏筛（普通筛法）

每个合数可以分解为素数的乘积，那么在搜索到一个数为素数的时候，就把它的倍数标记为合数。复杂度$O(nlognlogn)$

```
for(int i=2;i<=n;i++){
    if(prime[i]==0){
        p[++tot]=i;
        for(int j=2;j*i<=n;j++){
            prime[j*i]=1;
        }
    }
}
```

欧拉筛 （线性筛法）

每次用最小质因子来筛素数，避免重复筛选。确保每个合数只被最小质因子p筛一次。

```
for(int i=2;i<=n;i++){
    if(prime[i]==0) p[++tot]=i;
    for(int j=1;j<=tot&&i*p[j]<=n;j++){
        prime[i*p[j]]=1;
        if(i%p[j]==0) break;
    }
}
```

唯一分解定理

任意正整数都有且只有一种方式写出其素因子的乘积表达式。

$A = p_1^{k_1} * p_2^{k_2} * \ldots * p_n^{k_n}$，其中$p_i$均为素数

约数个数定理

一个大于1的数的约数个数为

$$(a_1 + 1) * (a_2 + 1) * \ldots * (a_n + 1)$$

约数和方程

对于已经分解的整数$A = p_1^{k_1} * p_2^{k_2} * \ldots * p_n^{k_n}$

有A的所有因子之和为$S = (1 + p_1 + p_1^2 + \ldots + p_1^{k_1}) * (1 + p_2 + p_2^2 + \ldots + p_2^{k_2}) * \ldots * (1 + p_n + p_n^2 + \ldots + p_n^{k_n})$

欧拉函数

对正整数n，欧拉函数是小于等于n的数中与n互质的数的数目。

欧拉函数又称为ϕ函数，例如$\phi(8) = 4$，因为1，3，5，7均与8互质。

引理

$$①如果n为某一个素数p，则：\phi(p) = p - 1$$
$$②如果n为某一个素数p的幂次p^a，则\phi(p^a) = (p - 1) * p^{a-1}$$
$$③如果n为任意两个互质的数a, b的积，则\phi(a * b) = \phi(a) * \phi(b)$$

证明

$$②共p^a - 1个数比p^a小，其中有p^{a-1} - 1个数能被p整除，表示为p * t(t = 1, 2, \ldots, p^{a-1} - 1), 相减可得$$
$$③在比a * b小的a * b - 1个整数中，有\phi(a)个与a互质的数，有\phi(b)个与b互质的数，必须既与a互质，又与b互质，才会与a * b互质，满足条件的数共有\phi(a)$$

筛选法

```
void init(){ //筛选法打欧拉函数表
    euler[1]=1;
    for(int i=2;i<maxn;i++) euler[i]=i;
    for(int i=2;i<maxn;i++)
        if(euler[i]==i)
            for(int j=i;j<maxn;j+=i)
                euler[j]=euler[j]/i*(i-1);
}
```

直接求解欧拉函数

```
int eu(int n){  //直接求解欧拉函数
        int res=n,a=n;
        for(int i=2;i*i<=a;i++){
                if(a%i==0){  //分解质因数
                        res=res/i*(i-1);
                        while(a%i==0) a/=i; //把质因数彻底分解
                }
        }
        if(a>1) res=res/a*(a-1);
        return res;
}
```

线性的求法看后面。

需要注意的性质：

①phi(p)==p-1是因为素数p除了1以外的因子只有p，所以与p互质的个数是p-1个；

②$phi(p^k) == p^k - p^{k-1} == (p - 1) * p^{k-1}$

判断素数

范围较小的时候，可以从2~sqrt(n)中判断有无n的因子，

范围较大且要找出所有素数时，可以使用素数筛。

Miller Rabin 素数测试

前置知识：唯一分解定理、威尔逊定理、费马定理

威尔逊定理：若p为素数，则$(p-1)! \equiv -1 \mod p$

威尔逊逆定理：若$(p-1)! \equiv -1 \mod p$，则p一定为素数

费马定理：若p为素数，a为正整数，且a和p互质，则$a^{p-1} \equiv 1 \mod p$

测试过程

$$(1)\text{计算奇数}M\text{，使得}N = 2^r * M + 1$$
$$(2)\text{选择随机数}A < N$$
$$(3)\text{对于任意}i < r\text{，若}A^{2^i * M} \mod N = N - 1\text{, 则}N\text{通过随机数}A\text{的测试}$$
$$(4)\text{或者，若}A^M \mod N = 1\text{, 则}N\text{通过随机数}A\text{的测试}$$
$$(5)\text{让}A\text{取不同的值对}N\text{进行}5\text{次测试，若全部通过则判定}N\text{为素数}$$

概括

不断选取不超过n-1的基b（共s次），计算是否每次都有$b^{n-1} \equiv 1 \mod n$，若每次都成立则n是素数，否则为合数。

二次探测定理

如果p是奇素数，则$x^2 \equiv 1 \mod p$的解为$x = 1 || x = p - 1 \mod p$

这个定理可以提高测试效率。

```
#include <bits/stdc++.h>
#define int long long
#define ull unsigned long long
#define ld long double
using  namespace std;

const int p[]={2,3,5,7,11,13,17,19,61}; //越多越准确

int mul(int a, int b, int p){ //快速乘
    ull c=(ull)a*b-(ull)((ld)a/p*b+0.5L)*p; //自动溢出
    return c<p?c:c+p; //转化为正数
}

int fpow(int a, int b, int mod){ //快速幂里的乘法用快速幂替换
    int res = 1;
    while(b){
        if(b&1) res=mul(res,a,mod);
        a=mul(a,a,mod);
        b>>=1;
    }
    return res;
}

bool miller_rabin(int x){ //米勒拉宾质数判定
    if (x < 2) return 0;
    int i,j,y=x-1,a,b;
    while(y&1^1) y>>=1; //相当于先化为奇数
    for(i=0;i<9;i++){
        if (x%p[i]<1)   return x == p[i]; //不可以是质数p的倍数
    }
    for (i=2;i<9;i++){
        for (a=b=fpow(p[i],y,x),j=y;j<x&&a>1;j+=j,a=b){
            if (b=mul(b,b,x),b==1&&a!=x-1)  return 0; //二次检测
            //b和x互质，b^2modx=1，要b=1,要么b=x-1,那么b==1时a应该为x-1
        }
        if(a!=1) return 0; //最终a不为1，没有通过二次检验
    }
    return 1;
}

signed main(){
    int n;
    while(cin>>n){
        puts(miller_rabin(n)?"Y":"N");
    }
    return 0;
}
```

Pollard Rho算法求大数因子

时间效率要求较高时，用来分解一个合数n。

luoguP4718 【模板】Pollard-Rho算法

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef double db;
#define lll __int128
ll max_factor,t,n;
inline ll read(){   ll x=0,f=1;char ch=getchar();while(ch<'0'||ch>'9'){if(ch=='-')
f=f*-1;ch=getchar();}while(ch>='0'&&ch<='9'){x=x*10+ch-'0';ch=getchar();}return x*f;}
inline ll gcd(ll a,ll b){return b?gcd(b,a%b):a;}
inline ll qp(ll x,ll p,ll mod){
    ll ans=1;
    while(p){
        if(p&1) ans=(lll)ans*x%mod;
        x=(lll)x*x%mod;
        p>>=1;
    }
    return ans;
}

inline bool mr(ll x,ll b){ //米勒罗宾素数检验
    ll k=x-1;
    while(k){
        ll cur=qp(b,k,x);
        if(cur!=1&&cur!=x-1) return false;
        if((k&1)==1||cur==x-1) return true;
        k>>=1;
    }
    return true;
}

inline bool prime(ll x){
    if(x==46856248255981ll|| x<2) return false;
    if(x==2||x==3||x==7||x==61||x==24251) return true;
    return mr(x,2)&&mr(x,61);
}

inline ll fc(ll x,ll c,ll n){return ((lll)x*x+c)%n;}

inline ll PR(ll x){ //Pollard Rho算法求素因子
    ll s=0,t=0,c=1ll*rand()%(x-1)+1;
    int stp=0,goal=1;
    ll val=1;
    for(goal=1;;goal<<=1,s=t,val=1){
        for(stp=1;stp<=goal;++stp){
            t=fc(t,c,x);
            val=(lll)val*abs(t-s)%x;
            if((stp%127)==0){
                ll d=gcd(val,x);
                if(d>1) return d;
            }
        }
        ll d=gcd(val,x);
        if(d>1) return d;
    }
}

inline void fac(ll x){ //求最大因子
    if(x<=max_factor||x<2) return;
    if(prime(x)){
        max_factor=max_factor>x?max_factor:x;
        return;
    }
    ll p=x;
    while(p>=x) p=PR(x);
    while((x%p)==0) x/=p;
    fac(x),fac(p);
}

signed main(){
    cin>>t;
    srand((unsigned)time(NULL));
    while(t--){
        n=read();
        max_factor=0;
        fac(n);
        if(max_factor==n) puts("Prime");
        else printf("%lld\n",max_factor);
    }
    return 0;
}
```

常用线性筛模板

线性筛法求欧拉函数

```
void Get_phi(){
    phi[1]=1;
    for(int i=2;i<N;i++){
        if(!np[i]){
            pri[++cnt]=i;
            phi[i]=i-1;
        }
        for(int j=1;j<=cnt&&i*pri[j]<N;j++){
            np[i*pri[j]]=1;
            if(i%pri[j]==0){
                phi[i*pri[j]]=pri[j]*phi[i]; //可以手动证一下
                break;
            }else phi[i*pri[j]]=(pri[j]-1)*phi[i]; //互质，积性
        }
    }
}
```

线性筛法求莫比乌斯函数

定义莫比乌斯函数

$$\mu(n)=\begin{cases}1,n=1\\(-1)^k,n=p_1p_2...p_k\\0,\text{其他}\end{cases}$$

```
void init(){
    mu[1]=1;np[1]=1;
    for(int i=2;i<N;i++){
        if(!np[i]) pri[++cnt]=i,mu[i]=-1;
    }
    for(int j=1;j<=cnt&&i*pri[j]<N;j++){
        np[i*pri[j]]=1;
        if(i%pri[j]==0){
            mu[i*pri[j]]=0;
        }else mu[i*prj[j]]=-mu[i];
    }
}
```

线性筛法求约数个数

d[i]:表示i的约数个数

num[i]:表示i的最小素因子个数

pri[i]:表示第i个素数

```
void init(){
    np[1]=1;d[1]=1;
    for(int i=2;i<N;i++){
        if(!np[i]){
            pri[++cnt]=i;
            num[i]=1;
            d[i]=2;//只有1和他本身
        }
        for(int j=1;j<=cnt&&pri[j]*i<N;j++){
            np[i*pri[j]]=1;
            if(i%pri[j]){ //互质时直接乘素因子的约数个数即可
                d[i*pri[j]]=d[i]*d[pri[j]]; //d[pri[j]]=2
                num[i*pri[j]]=1;
            }else{ //这里推一推约数个数定理，就是除一个数乘一个数
                num[i*pri[j]]=num[i]+1;
                d[i*pri[j]]=d[i]/(num[i]+1)*(num[i]+2)
                break;
            }
        }
    }
}
```

线性筛法求约数和

sd[i]：表示i的约数和

sp[i]：表示i的最小素因子的等比数列之和。

pri[i]表示第i个素数

```
void init(){
    np[1]=sd[1]=1;
    for(int i=2;i<N;i++){
        if(!np[i]){
            pri[++cnt]=i;
            sd[i]=i+1;
            sp[i]=i+1;
        }
        for(int j=1;j<=cnt&&pri[j]*i<N;j++){
            np[pri[j]*i]=1;
            if(i%pri[j]){ //如果互质，实际上是在后面乘上(1+pri[j])
                sd[i*pri[j]]=sd[i]*sd[pri[j]];
                sp[i*pri[j]]=pri[j]+1;
            }else{ //看着约数和公式推就好，就是乘一项除一项
                sp[i*pri[j]]=sp[i]*pri[j]+1;
                sd[i*pri[j]]=sd[i]/sp[i]*sp[i*pri[j]];
                break;
            }
        }
    }
}
```

求逆元

前面提到过，这里不做证明，补充代码。

```
inv[1]=1;
for(ull i=2;i<=n;i++){
    inv[i]=(ull)(p-p/i)*inv[p%i]%p;
}
```

参考资料

信息学奥赛之数学一本通 （林厚从）

https://www.cnblogs.com/zjp-shadow/p/9267675.html#autoid-3-3-0

https://blog.csdn.net/ControlBear/article/details/77527115