数论

基础知识

算术基本定理：对于每一个正整数N都可以被分解成如右的形式：$N=p_1^{a_1}\times p_2^{a_2}\times...\times p_k^{a_k}$
其中p_i是不同的质数，a_i是正整数
N中能被x整除的数的个数：$\lfloor\dfrac{N}{x}\rfloor$

约数个数：对于$N=p_1^{a_1}\times p_2^{a_2}\times...\times p_k^{a_k}$，它的约数个数是$(a_1+1)(a_2+1)...(a_k+1)$
证明：对于N的约数d，其同样可以写成$p_1^{b_1}\times p_2^{b_2}\times...\times p_k^{b_k}$，其中$0\le b_i\le a_i$
于是b_i就会有a_i+1种取值

约数之和：对于$N=p_1^{a_1}\times p_2^{a_2}\times...\times p_k^{a_k}$，它的约数之和是
$(p_1^0+...+p_1^{a_1})(p_2^0+...+p_2^{a_2})...(p_k^0+...+p_k^{a_k})$
证明：展开上式就会发现实际上就是上面的全部d的和

质数个数：在$[1,n]$中，约有n/lnn个质数
$[1,n]$中 1到n的约数之和的总和为$nlogn$级别

1e6 78498个质数

1e7 664579个质数

1e8 5761455个质数

$gcd(a,b,c)=gcd(a,b-a,c-b)$

$gcd(a,b)=gcd(a,b-a)$

$gcd(a,b)=gcd(a,b\%a)$

$gcd(a\times d,b\times d)=gcd(a,b)\times d$

解二元一次方程

考虑二元一次方程的一般形式

$$ax+by=c$$

在这样的一个方程中，我们只有一个式子，这就意味着，这个方程，要么无解，要么有无限组解

那么我们首先需要知道什么时候有解，很明显，当且仅当c是$gcd(a,b)$的倍数时，有解

考虑裴蜀定理

$$ax + by = gcd(a, b)$$

这个式子是成立的，而且这里的$ax_0 + by_0$是最小的一组解

考虑最简单的情况，a与b互质，那么就有

$$ax + by = 1$$

设$ax_0 + by_0 = 1$，那么则有$a(cx_0) + b(cy_0) = c$

那么我们只需要解出这个方程的解再乘上c即可

需要注意的是，得出来的x, y不一定是符合我们要求的解，即对于x来说，可能不位于$[0, b)$

而对于y来说，可能不位于$[0, a)$

也就是说，他们可能不是最小正整数解

于是我们需要使用下面的变换

```
x = (x % b + b) % b;
y = (y % a + a) % a;
```

质数筛

```
const int N = (2 << 20) + 5;
#define int long long
int st[N];//用来记录每个数是否为质数，如果为0的话就是质数
int cnt;
int primes[N];//质数数组
int minp[N];//每个数对应的最小质因子
void getPrime(int n)//线性素数筛，O(n)，可以计算最小质因子
{
    for (int i = 2; i <= n; i++)
    {
        if (!st[i])primes[cnt++] = i, minp[i] = i;//质数的质因子是其本身
        for (int j = 0; primes[j] * i <= n; j++)
        {
            st[primes[j] * i] = 1;//将质数数组中的质数的倍数设为非质数
            minp[primes[j] * i] = primes[j];//那么质数的倍数的最小质因子自然就是这个质数
            if (i % primes[j] == 0)break;
        }
    }
}
```

求约数

单个数

$O(\sqrt{n})$

```
void solve()
{
    int x;
```

```
    cin >> x;
    for (int i = 1; i * i <= x; i++)
    {
        if (x % i == 0)
        {
            //得到两个因子，i, x / i
            cout << i << " ";
            if (i != x / i) cout << x / i << " ";
        }
    }
    cout << "\n";
}
```

前n个数

$O(nlogn)$

```
for (int i = 1; i <= 1e5; i++)
{
    for (int j = i; j <= 1e5; j += i)
    {
        factor[j].emplace_back(i);//i是j的因子
    }
}
```

分解质因数

复杂度$O(\sqrt{N})$

```
map<int, int> mp;

void solve()
{
    int x;
    cin >> x;
    for (int i = 0; i < cnt; i++)
    {
        if (x % primes[i] == 0)
        {
            while (x % primes[i] == 0)mp[primes[i]]++, x /= primes[i];
        }
    }
    if (x != 1) mp[x]++;
    for (PII p : mp)
    {
        cout << p.x << " " << p.y << "\n";
    }

    cout << "\n";
    mp.clear();
}
```

欧拉函数

欧拉函数$\varphi(N)$指的是，对于$1 \sim N$，其中与N互质的数的个数

考虑反向考虑，用总数减去与N不互质的数，同时考虑容斥原理

首先尝试枚举单个质数

可以发现，应该有有两个质数的数被减去，所以要加回去

然后加上三个的

则有

$$
\begin{gathered}
N = p_1^{a_1} \times p_2^{a_2} \times \ldots \times p_k^{a_k} \\
\varphi(N) = N \\
-(\frac{N}{p_1^{a_1}} + \ldots + \frac{N}{p_k^{a_k}}) \\
+(\frac{N}{p_1^{a_1} * p_2^{a_2}} + \ldots + \frac{N}{p_{k-1}^{a_{k-1}} * p_k^{a_k}}) \\
- \ldots
\end{gathered}
$$

实际上合并后可得

$$
\begin{gathered}
N = p_1^{a_1} \times p_2^{a_2} \times \ldots \times p_k^{a_k} \\
\varphi(N) = N(1 - \frac{1}{p_1}) \ldots (1 - \frac{1}{p_k}) \\
= N(\frac{p_1 - 1}{p_1}) \ldots (\frac{p_k - 1}{p_k})
\end{gathered}
$$

算法复杂度同质因数分解

求单个欧拉函数

```
map<int, int> mp;

void solve()
{
    int x;
    cin >> x;
    int tmp = x;
    for (int i = 0; i < cnt; i++)
    {
        if (x % primes[i] == 0)
        {
            while (x % primes[i] == 0)mp[primes[i]]++, x /= primes[i];
        }
    }
    int ans = tmp;
    for (PII p : mp)
    {
        ans /= p.x;
        ans *= (p.x - 1);
    }
    if(x != 1)
    {
```

```
        ans /= x;
        ans *= (x - 1);
    }
    cout << ans << "\n";
    mp.clear();
}
```

求前n个数的欧拉函数

考虑线性筛

```
const int N = 1e6 + 5;
#define int long long
int st[N];//用来记录每个数是否为质数，如果为0的话就是质数
int cnt;
int primes[N];//质数数组
int minp[N];//每个数对应的最小质因子
void getPrime(int n)//线性素数筛，O(n)，可以计算最小质因子
{
    for (int i = 2; i <= n; i++)
    {
        if (!st[i])primes[cnt++] = i, minp[i] = i;//质数的质因子是其本身
        for (int j = 0; primes[j] * i <= n; j++)
        {
            st[primes[j] * i] = 1;//将质数数组中的质数的倍数设为非质数
            minp[primes[j] * i] = primes[j];//那么质数的倍数的最小质因子自然就是这个质数
            if (i % primes[j] == 0)break;
        }
    }
}
```

考虑质数p的欧拉函数

显然质数p的欧拉函数是$p-1$

```
phi[p] = p - 1;
```

考虑$primes[j]*i$的欧拉函数

如果$i\%primes[j]==0$

那么说明$primes[j]$是i的因子

所以$primes[j]$不会给$primes[j]*i$提供更多的因子，只会改变总值

则

```
phi[i * primes[j]] = phi[i] * primes[j]
```

而如果$i\%primes[j]!=0$

那么$primes[j]$会给$primes[j]*i$提供一个新的因子，从而使得还需要再乘上$\frac{primes[j]-1}{primes[j]}$

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

```
const int N = 1e6 + 5;
```

```
int st[N];//用来记录每个数是否为质数，如果为0的话就是质数
int cnt;
int primes[N];//质数数组
int minp[N];//每个数对应的最小质因子
int phi[N];
int getPrime(int n)//线性素数筛，O(n)，可以计算最小质因子
{
    phi[1] = 1;
    int ans = 0;
    for (int i = 2; i <= n; i++)
    {
        if (!st[i])primes[cnt++] = i, minp[i] = i, phi[i] = i - 1;//质数的质因子是
其本身
        for (int j = 0; primes[j] * i <= n; j++)
        {
            st[primes[j] * i] = 1;//将质数数组中的质数的倍数设为非质数
            minp[primes[j] * i] = primes[j];//那么质数的倍数的最小质因子自然就是这个质数
            if (i % primes[j] == 0)
            {
                phi[primes[j] * i] = phi[i] * primes[j];
                break;
            }
            phi[primes[j] * i] = phi[i] * primes[j] * (primes[j] - 1) /
primes[j];
        }
    }
    for (int i = 1; i <= n; i++)ans += phi[i];
    return ans;
}
```

可以发现，我们用欧拉函数的前缀和来表示，前N个数中，互质的数的个数，即上面的`ans`

特殊的欧拉函数

给定N，求$1 \le x, y \le N$，且$gcd(x, y)$为素数的数对有多少个

考虑式子$gcd(x, y) = p$其中p是质数

转化可得$gcd(x/p, y/p) = 1$

令$x/p = x', y/p = y'$，则有

$1 \le x', y' \le N$，且$gcd(x, y) = 1$，这实际上就是欧拉函数的前缀和

那么我们就可以枚举p，然后直接求欧拉函数的前缀和即可

```
void solve()
{
    int n;
    cin >> n;
    getPrime(n);
    int ans = 0;
    for (int i = 0; i < cnt; i++)
    {
        int p = primes[i];
        ans += pre[n / p] * 2 + 1;
    }
    cout << ans << "\n";
}
```

有一个细节是，数对$(1,1)$是$gcd(1,1)=1$，但是不符合题意，所以我们需要让$phi[1]=0$，这与普通的模板不同