算法基础（十八）：数学基础 - 数论3 - 欧拉函数与定理

欧拉函数

基本思想：

欧拉函数的定义：

`1 - N`中与`N`互质的数的个数被称为欧拉函数，记为 $\phi(N)$。若在算数基本定理中，$N = p_1^{a1}p_2^{a2}p_m^{am}$则：$\phi(N) = N \times \frac{p_1-1}{p_1} \times \frac{p_2-1}{p_2} \times \ldots \frac{p_m-1}{p_m}$

其中，公约数只有`1`的两个数互为质数

比如：

`6 = 2 * 3`

$\phi(6) = 6 * (1 - \frac{1}{2}) * (1 - \frac{1}{3}) = 2$

证明： （利用容斥原理）

假设`N`的质因子有`k`个，`p1 ~ pk`，那么只要我们去掉`1 ~ N`中含有与`N`相同质因子的数即可，也就是去掉`1 ~ N`中`p1 ~ pk`的倍数的数

先从`1 ~ N`中去掉`p1, p2 ...pk`的所有的倍数，`1 ~ N`中`p1`的倍数有`N / p1`个，减掉就剩下`N - N / p1`个，那么这些数的所有的倍数就剩下`N - N/p1 - N/p2 - N/p3....N/pk`个**（这些都是整除，类似于c语言的除法）**，但是这些数可能会被多去掉，因为一个数可能既是`p1`的倍数，同时也是`p2`的倍数

接着得将这些数加回来，加上所有`pi * pj`的倍数，这里`i,j`取`1 ~ k`中任意的两个值，那么结果就有`N - N/p1 - N/p2 - N/p3....N/pk + N/(p1*p2) + N(p1*p3)...`，但是仍然有个问题，如果一个数是多个数的倍数，比如一个数`m`是`p1, p2, p3`的倍数，那么他首先会被减三次，接着又被加三次，等于没减掉，但是其实我们的目的是要将其去掉

所以我们还得减掉`pi * pj * pl`的倍数，这里`i, j, l`在`1 ~ k`中任取三个值，那么结果就是`N - N/p1 - N/p2 - N/p3....N/pk + N/(p1*p2) + N(p1*p3)...-N/(p1 * p2 * p3) - N/(p1 * p2 * p4)....`

那么对于四个数乘积分之`N`，首先被减了`4`次，接着两个数分之`N`被加了`6`次，然后三个数分之`N`被减了`4`次，多减了一次，所以我们得加上四个数乘积分之`N`，`N - N/p1 - N/p2 - N/p3....N/pk + N/(p1*p2) + N(p1*p3)...-N/(p1 * p2 * p3) - N/(p1 * p2 * p4)....+ N/(p1*p2*p3*p4) + N(p1*p2*p3*p5)`

依次类推下去，奇数分之`N`就减掉，偶数分之`N`就加上，得到一大坨这样的式子：

`N - N/p1 - N/p2 - N/p3....N/pk + N/(p1*p2) + N(p1*p3)...-N/(p1 * p2 * p3) - N/(p1 * p2 * p4)....+ N/(p1*p2*p3*p4) + N(p1*p2*p3*p5)...-N(p1*p2*p3*p4*p5)`

那么我们上面那个式子$\phi(N) = N \times \frac{p_1-1}{p_1} \times \frac{p_2-1}{p_2} \times \ldots \frac{p_m-1}{p_m}$展开就是这样一大坨

我们来观察一下：

$\phi(N) = N \times (1 - \frac{1}{p_1}) \times (1 - \frac{1}{p_2}) \times \ldots (1 - \frac{1}{p_m})$

其中：

$\frac{1}{p1}$的系数就是后面的式子中第一个式子取`1/p1`，其他的式子取`1`，那么系数就是`-N`，负数

$\frac{1}{p1*p2}$的系数就是前面两项取`1/p1, 1/p2`后面的都取`1`，那么系数就是`N`，正数

依次向后展开，就得到了两个式子是等价的，证明完毕。

代码实现：

给定 n 个正整数 a_i，请你求出每个数的欧拉函数。

欧拉函数的定义

> $1 \sim N$ 中与 N 互质的数的个数被称为欧拉函数，记为 $\phi(N)$。
> 若在算数基本定理中，$N = p_1^{a1} p_2^{a2} \ldots p_m^{am}$，则：
> $\phi(N) = N \times \frac{p_1 - 1}{p_1} \times \frac{p_2 - 1}{p_2} \times \ldots \times \frac{p_m - 1}{p_m}$

输入格式

第一行包含整数 n。

接下来 n 行，每行包含一个正整数 a_i。

输出格式

输出共 n 行，每行输出一个正整数 a_i 的欧拉函数。

数据范围

$1 \le n \le 100$,
$1 \le a_i \le 2 \times 10^9$

输入样例：

```
3
3
6
8
```

输出样例：

```
2
2
4
```

用这个公式求得时间复杂度是在分解质因数上面，分解质因数得时间复杂度是`O(sqrt(n))`，因此用这个公式来求欧拉函数的时间复杂度是`O(sqrt(n))`，所以这道题目的总的时间复杂度是`O(n * sqrt(ai))`

`n = 100, ai = 2 * 10 ^ 9`整个的时间复杂度在四百万到五百万左右

```
#include<iostream>
#include<algorithm>
using namespace std;

int main()
{
    int n;
    cin >> n;
```

```

    while(n --)
    {
        int a;
        cin >> a;

        //答案一开始是a，然后慢慢减下去
        int res = a;
        //开始分解质因数
        for(int i = 2; i <= a / i; i ++)
        {
            if(a % i == 0)
            {
                //res = res * (1 - 1 / i);
                //为了不出现小数，在这里稍微变换一下式子的结构
                res = res / i * (i - 1);
                while(a % i == 0)
                {

                    a /= i;
                }
            }

        }
        //若最后a > 1说明原来的a中含有大于sqrt(a)的质因子，除去它
        if(a > 1) res = res / a * (a - 1);
        cout << res <<endl;
    }
    return 0;

}
```

欧拉函数（筛法）

基本思想：

上面的求法是我们用公式来做的，我们只能求`N`本身的欧拉函数，但是在一些情况下我们需要求`1 ~ N`所有的数的欧拉函数，这时我们再用公式来做的话，时间复杂度就变成了`O(n*sqrt(n))`，比较大，需要优化

我们可以在筛质数的过程中顺便就求出了欧拉函数

最核心的地方就是当遍历到`i`的时候，用`i`取筛后面的合数有两种情况，详细解释看代码

代码实现：

给定一个正整数 n，求 $1 \sim n$ 中每个数的欧拉函数之和。

输入格式

共一行，包含一个整数 n。

输出格式

共一行，包含一个整数，表示 $1 \sim n$ 中每个数的欧拉函数之和。

数据范围

$1 \le n \le 10^6$

输入样例：

```
6
```

输出样例：

```
12
```

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

//和的结果可能很大，用ll来存
typedef long long LL;
const int N = 1000010;

//primes存的是每一个质数，cnt表示质数的个数
int primes[N], cnt;

//用来存每个数的欧拉函数
int phi[N];

//st[i] = true表示i这个数是某一个数的倍数，删掉
bool st[N];

LL get_eulers(int n)
{
    //从定义出发，phi[1] = 1
    phi[1] = 1;


    //线性筛法
    for(int i = 2; i <= n; i ++)
    {
        //如果当前这个数没有被筛过，说明这个数就是一个质数
        if(!st[i])
        {
            primes[cnt ++] = i;
            //如果这个数i是质数，显然1 - (i-1)都是与其互质
            //其欧拉函数是(i - 1)
            phi[i] = i - 1;
        }
```

```
        //遍历每一个质数，保证不会筛掉n以外的合数
        for(int j = 0; primes[j] <= n / i; j ++)
        {
            //筛掉这个质数的倍数
            //每一个合数都必然是被自己的最小质因子筛掉的
            st[primes[j] * i] = true;
            //如果当前的数是primes[j]的倍数
            //再往后遍历质数就不能保证是用最小质数来筛了
            //所以退出
            if(i % primes[j] == 0)
            {
                //当i % pj == 0的时候pj是i的最小质因子
                //phi(i)与pho(pj * j)分解质因子的p1 ~ pk是相同的
                //两者的欧拉函数只相差pj
                phi[primes[j] * i] = phi[i] * primes[j];
                break;
            }

            //如果i % pj != 0，此时枚举质数的时候和没有枚举到i的最小质因子
            //i分解质因子是p1 ~ pk
            //pj * i分解质因子会比i多了一个最小质因子pj
            //那么phi(pj * i) = phi(i) * pj * (pj - 1)/pj
            phi[primes[j] * i] = phi[i] * (primes[j] - 1);

        }
    }
    LL res = 0;

    for(int i = 1; i <= n; i ++)
    {
        res += phi[i];
    }

    return res;
}


int main()
{
    int n;
    cin >> n;

    //求1 ~ N中每一个欧拉函数的和
    cout << get_eulers(n) << endl;

    return 0;
}
```

我们以`6`为例来模拟这个过程

1. `i = 2`

    1. `` `st[2] = false ``， `ph[2] = 1`
    2. 从小到大遍历质数
    3. `j = 0`
        1. 遍历到质数`2`， `2`与`i = 2`的乘积`4`被最小质因数`2`筛掉，同时`i = 2`又能被此时的质数`2`整除，说明`2`也是`i = 2`的最小质因数
        2. 此时计算质数`2`与`i = 2`的乘积`4`的`phi[4]`， `4`的最小质因数也是`2`， `4`与`2`相比就是数值多乘了`2`，于是`phi[4] = phi[i] * pj = 1 * 2 = 2`
2. `i = 3`
    1. `` `st[3] = false ``， `ph[3] = 2`
    2. 从小到大遍历质数
    3. `j = 0`
        1. 遍历到质数`2`， `2`与`i = 3`的乘积`6`被最小质因子`2`筛掉，此时`i = 3`不能被质数`2`整除，那么此时`phi[6]`与`phi[i=3]`相比数值多乘了`2`并且质数因子多了`2`
        2. 所以此时计算`phi[6] = phi[3] * 2 * (2-1)/2 = phi[3] * (2-1) = 2`
    4. `j = 1`
        1. 遍历到质数`3`，但此时以`3`为最小质数的合数超过`6`，无意义，所以退出
3. `i = 4`
    1. 从小到大遍历质数
    2. 遍历到质数`2`，但此时以`2`为最小质数的合数超过`6`，无意义，所以退出
4. `i = 5`
    1. `` `st[5] = false ``， `ph[5] = 4`
    2. 从小到大遍历质数
    3. 遍历到质数`2`，但此时以`2`为最小质数的合数超过`6`，无意义，所以退出
5. `i = 6`
    1. 从小到大遍历质数
    2. 遍历到质数`2`，但此时以`2`为最小质数的合数超过`6`，无意义，所以退出

所以我们可以发现上面线性筛的过程中

1. 每次筛掉一个合数的时候都会求出这个合数的欧拉函数
2. 对于一个质数，外层循环遍历到它的时候求出它的欧拉函数
3. 由于线性筛筛掉了所有的合数，所以也就求出来所有合数的欧拉函数
4. 由于外层循环遍历了所有的数，所以也就求出来所有质数的欧拉函数

另外为什么在外层遍历到`i`的时候`phi[i]`在之前可以被求出来？

假如`i`是一个质数，那么在遍历到这个质数的时候同时求出来`phi[i] = i - 1`

假如`i`是一个合数

首先`phi[4]`确实是被之前求出来的，当用到的时候是已知的（归纳奠基）

假设当用到`phi[i]`的时候是已知的（归纳假设）

那么当我们用到`phi[i + 1]`的时候，因为根据筛法求质数，当外层遍历到`i + 1`的时候`2 ~ i`中的所有的合数都已经被筛掉了，而`i + 1`一定可以写成`p*l`这种形式，其中`p`是`i + 1`的最小质因子，`l`是另外的因数，且`p <= l <= i`在外层遍历到`l`，从小到大遍历质数的时候，就可以筛掉`i+1`，同时也就求出了`phi[i+1]`

所以在利用`phi[i]`的时候其值总是已知的

欧拉定理

同余：

两个整数`a、b`，若它们除以整数`m`所得的余数相等，则称`a`与`b`对于模`m`同余或`a`同余于`b`模m。

记作：`a≡b (mod m)`

互质：

两个数互质，也就是说这两个数按照算术的基本定理分解后没有相同的质因数

欧拉定理：

若`a`与`n`互质，则有：

$a^{\phi(n)} \equiv 1(mod n)$，也就是说$a^{\phi(n)}$`mod n`为`1`

比如：

$5^{\phi(6)} mod 6 = 25 mod 6 = 1$

证明：

假设在`1 ~ n`中与`n`互质的数为$a_1, a_2, a_3 ... a_{\phi(n)}$一共有$\phi(n)$个这样的数，**那么`a`乘这些数得到：** $a * a_1, a * a_2, ... a * a_\phi(n)$**也与`n`互质**（因为这些数与`n`没有相同的质因数）

并且这些数都两两`mod n`互不相同

假设有两个数同余`a*ai ≡ a*aj(mod n)`，则`a(ai - aj) ≡ 0 (mod n)`，即`n | a(ai - aj)`，`n`可以整除`a(ai - aj)`，但是`a`与`n`互质，并且`ai - aj`又小于`n`，所以`a(ai - aj)`中不可能含有`n`这个因子，所以`n`不可能整除`a(ai - aj)`，矛盾，所以，两两`mod n`互不相同

那么我们构成一个集合：

$\{a * a_1 mod n, a * a_2 mod n, ... a * a_{\phi(n)} mod n\}$

集合中元素互不相同，小于`n`，且数量是$\phi(n)$个，而且与`n`互质，那么这些元素，不就是集合$\{a_1, a_2, a_3 ... a_{\phi(n)}\}$嘛，只是元素的顺序可能不同

那么就有：

$aa_1 mod n * aa_2 mod n ... * aa_{\phi(n)} mod n = a_1 * a_2 * ... a_{\phi(n)}$

所以根据模运算法则：

$a_1 * a_2 * ... a_{\phi(n)} \equiv aa_1 * aa_2 * aa_3 ... aa_{\phi(n)} (mod n)$

于是我们得到：

$(a^{\phi(n)}-1)a_1*a_2*...a_{\phi(n)} \equiv 0(mod n)$

由于：

$a_1*a_2*...a_{\phi(n)}$与`n`互质

所以：

$n|(a^{\phi(n)}-1)$

即：

$(a^{\phi(n)}-1) \equiv 0(mod n)$

所以：
$a^{\phi(n)} \equiv 1(mod n)$

费马小定理（欧拉定理的一个推论）

当`n`是质数的时候记作`p`，则有：

$a^{\phi(p)} \equiv 1(mod p)$

也就是：

$a^{p-1} \equiv 1(mod p)$，其中`p`是质数，`a`与`p`互质