知识点：欧拉函数

- **什么是欧拉函数?**
  - 欧拉函数是小于x的整数中与x互质的数的个数，一般用φ(x)表示。特殊的，φ(1)=1。
- **如何计算欧拉函数?**
  - 其通式为$\varphi(x)=x\prod_{i=1}^{n}\left(1-\frac{1}{p_i}\right)$ φ(1)=1
  - 其中$p_1p_2...p_i$为x 的所有质因数，x是正整数。 那么，怎么理解这个公式呢？对于x的一个质因数p_i, 因为x以内p_i的倍数是均匀分布的，所以x以内有$\frac{1}{p_i}$ 的数是p_i的倍数，那么有$1-\frac{1}{p_i}$ 的数不是p_i的倍数。再对于p_j，同理，有$1-\frac{1}{p_j}$的数不是p_j的倍数所以有$(1-\frac{1}{p_i})*(1-\frac{1}{p_j})$的数既不是$p_i$的倍数,又不是$p_j$的倍数。最后就有$x\prod_{i=1}^{n}(1-\frac{1}{p_i})$的数与x互质，个数自然就是$x\prod_{i=1}^{n}(1-\frac{1}{p_i})$
- **什么是积性函数?**
  - 先介绍一下什么是积性函数，后面将会用到。若当m与n互质时，f(m∗n)=f(m)∗f(n)，那么f是积性函数。若对任意正整数，都有f(m*n)=f(m)*f(n)成立，则f是完全积性函数。
- **欧拉函数的几个性质**
  - 对于质数p，φ(p)=p−1
  - 若p为质数，n=p^k， 则φ(n)=p^k * p^{k-1}
  - 欧拉函数是积性函数，但不是完全积性函数。若m,n互质，则φ(m∗n)=φ(m)∗φ(n)。特殊的，当m=2，n为奇数时，φ(2*n)=φ(n)。
  - ## 素数GCD对
    - **题目描述**

      给定整数 N，求 1≤x,y≤N 且 Gcd(x,y) 为素数的数对 (x,y) 有多少对。

      输入

      一个整数 N。

      输出

      一个数，为对数。

      样例输入

      ```
      4
      ```

      样例输出

      ```
      4
      ```

      样例说明

(2,2),(2,4),(3,3),(4,2)

- **解答**
  - 题目分析
    - 由题意可知我们可以采取构造gcd（a，b）对的形式来寻找该题的解，首先找到gcd（a, b） = 1的a和b对，然后通过线性筛打一个素数表，假设素数为p则可以采用p * gcd(a, b) => gcd (p * a, p * b) = p;的形式来构造满足条件的gcd对，假设存在函数f（n）为求小于n的质数有多少个的函数，则对于任意小于等于n的b来说满足条件的数对有2 * ϕ (b) * f(n)个，对于所有满足条件的b求2 * ϕ (b) * f(n)和，得到结果
  - 主要代码如下：
  -
```
typedef long long ll;
#define MAX_N 1000000
ll prime[MAX_N + 5], phi[MAX_N + 5];

void init() {
    for (ll i = 2; i <= MAX_N; i++) {
        if (!prime[i]) {
            prime[++prime[0]] = i;
            phi[i] = i - 1;
        }
        for (ll j = 1; j <= prime[0]; j++) {
            if (i * prime[j] > MAX_N) break;
            prime[i * prime[j]] = 1;
            if (i % prime[j] == 0) {
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            } else {
                phi[i * prime[j]] = phi[i] * (prime[j] - 1);
            }
        }
    }
    return;
}
int main() {
    init();
    ll n, k = prime[0], ans = 0;
    cin >> n;
    while (k && prime[k] > n) k--;
    ans += k;
    for (ll i = 2; i <= n; i++) {
        while (k && prime[k] * i > n) k--;
        ans += 2 * phi[i] * k;
    }
    cout << ans << endl;
    return 0;
}
```
  -