Sieve of Eratosthenes

In mathematics, the **sieve of Eratosthenes**, one of a number of prime number sieves, is a simple, ancient algorithm for finding all prime numbers up to any given limit. It does so by iteratively marking as composite (i.e. not prime) the multiples of each prime, starting with the multiples of 2.

```
#include <stdio.h>
#include <math.h>

#define SIZE_N 100
#define SIZE_P 100

bool flag[SIZE_N+5];
int primes[SIZE_P+5];

int seive()
{
       int i,j,total=0,val;

       for(i=2;i<=SIZE_N;i++) flag[i]=1;

       val=sqrt(SIZE_N)+1;

       for(i=2;i<val;i++)
              if(flag[i])
                     for(j=i;j*i<=SIZE_N;j++)
                            flag[i*j]=0;

       for(i=2;i<=SIZE_N;i++)
              if(flag[i])
                     primes[total++]=i;

       return total;
}
int main()
{
    int total=seive();

    printf("Total Primes: %d\n",total);

    for(int i=0;i<total;i++)
        printf("%d\n",primes[i]);

    return 0;
}

// if SIZE_N 100            Total Primes: 25
// if SIZE_N 1000           Total Primes: 168
// if SIZE_N 1000000        Total Primes: 78498
```

How Many Divisors of a Number

Suppose you wish to find the number of divisors of 48. of prime number sieves Starting with 1 we can work through the set of natural numbers and test divisibility in each case, noting that divisors can be listed in factor pairs.

$$48 = 1\times48 = 2\times24 = 3\times16 = 4\times12 = 6\times8$$

Hence we can see that 48 has exactly ten divisors. It should also be clear that, using this method, we only ever need to work from 1 up to the square root of the number.

Although this method is quick and easy with small numbers, it is tedious and impractical for larger numbers. Fortunately there is a quick and accurate method using the divisor.

Let D(N) be the number of divisors for the natural number, N.

We begin by writing the number as a product of prime factors:

$$N=p_1^{q1} \text{ x } p_2^{q2} \text{ x } p_3^{q3} \text{ x } . . . . \text{ x } p_k^{qk}$$

Then the number of divisors: $D(N) = (q1+1) \text{ x } (q2+1) \text{ x } (q3+1) \text{ x}. . . . . \text{ x } (qk+1)$

The number of divisor function can be quickly demonstrated with the example we considered earlier: $48 = 2^4\times3^1$, therefore $D(48)=5\times2=10$.

```
#include <stdio.h>
#include <math.h>

#define SIZE_N 1000
#define SIZE_P 1000

bool flag[SIZE_N+5];
int primes[SIZE_P+5];

int seive()
{
       int i,j,total=0,val;

       for(i=2;i<=SIZE_N;i++) flag[i]=1;

       val=sqrt(SIZE_N)+1;

       for(i=2;i<val;i++)
              if(flag[i])
                     for(j=i;j*i<=SIZE_N;j++)
                            flag[i*j]=0;

       for(i=2;i<=SIZE_N;i++)
              if(flag[i])
                     primes[total++]=i;
```

```
        return total;
}

int divisor(int N)
{
        int i,val,count,sum;

        val=sqrt(N)+1;
        sum=1;
        for(i=0;primes[i]<val;i++)
        {
                if(N%primes[i]==0)
                {
                        count=0;
                        while(N%primes[i]==0)
                        {
                                N/=primes[i];
                                count++;
                        }
                        sum*=(count+1);
                }
        }
        if(N>1)
                sum=sum*2;
        return sum;
}

int main()
{
        int total=seive();
        int n;

        while(scanf("%d",&n)==1)
        {
                printf("%d No Of divisor:%d\n",n,divisor(n));
        }
        return 0;
}

//      5 No Of divisor:2
//      8 No Of divisor:4
//      100 No Of divisor:9
//      568 No Of divisor:8
//      48 No Of divisor:10
```

Sum of Divisors

Imagine you wish to work out the sum of divisors of the number 72. It would not take long to list the divisors, and then find their sum: 1 + 2 + 3 + 4 + 6 + 8 + 9 + 12 + 18 + 24 + 36 + 72 = 195.

However, this method would become both tedious and difficult for large numbers like 145600. Fortunately, there is a simple and elegant method at hand.

Let $\sigma(n)$ be the sum of divisors of the natural number, n.

For any prime, p: $\sigma(p) = p + 1$, as the only divisors would be 1 and p.

Consider p^a: $\sigma(p^a) = 1 + p + p^2 + ... + p^a$ (1).

Multiplying by p: $p\sigma(p^a) = p + p^2 + p^3 + ... + p^{a+1}$ (2).

Subtracting (1) from (2): $p\sigma(p^a)-\sigma(p^a) = (p-1)\sigma(p^a) = p^{a+1} - 1$.

Hence $\sigma(p^a) = (p^{a+1} - 1)/(p - 1)$.

For example, $\sigma(3^4)=(3^5-1)/(3-1) = 242/2 = 121$,
and checking: 1 + 3 + 9 + 27 + 81 = 121.

Although no proof is supplied here, the usefulness of the function, $\sigma(n)$, is its multiplicativity, which means that $\sigma(a\times b\times...)=\sigma(a)\times\sigma(b)\times...$, where a, b, ..., are relatively prime.

General Form:

If the prime power decomposition of an integer is

$$n = {p_1}^{e_1} * {p_2}^{e_2} * \cdots * {p_k}^{e_k}$$

Then we can write

$$\sigma(n) = \frac{{p_1}^{e_1+1} - 1}{p_1 - 1} * \frac{{p_2}^{e_2+1} - 1}{p_2 - 1} * \cdots * \frac{{p_k}^{e_k+1} - 1}{p_k - 1}$$

Returning to example, we use the fact that $\sigma(72) = \sigma(2^3\times3^2)$. As 2^3 and 3^2 are relatively prime, we can separately work out $\sigma(2^3) = 2^4 - 1 = 15$ and $\sigma(3^2) = (3^3 - 1)/2 = 13$. Therefore, $\sigma(72) = 15\times13 = 195$.

```
#include <stdio.h>
#include <math.h>
```

```
#define SIZE_N 1000
#define SIZE_P 1000

bool flag[SIZE_N+5];
int primes[SIZE_P+5];

int seive()
{
        int i,j,total=0,val;

        for(i=2;i<=SIZE_N;i++) flag[i]=1;

        val=sqrt(SIZE_N)+1;

        for(i=2;i<val;i++)
                if(flag[i])
                        for(j=i;j*i<=SIZE_N;j++)
                                flag[i*j]=0;

        for(i=2;i<=SIZE_N;i++)
                if(flag[i])
                        primes[total++]=i;

        return total;
}

int Sum_Of_Divisor(int N)
{
        int i,val,count,sum,p,s;

        val=sqrt(N)+1;
        sum=1;
        for(i=0;primes[i]<val;i++)
        {
                if(N%primes[i]==0)
                {
                        p=1;
                        while(N%primes[i]==0)
                        {
                                N/=primes[i];
                                p=p*primes[i];
                        }
                        p=p*primes[i];
                        s=(p-1)/(primes[i]-1);
                        sum=sum*s;
                }
        }

        if(N>1)
        {
                p=N*N;
                s=(p-1)/(N-1);
                sum=sum*s;
        }
        return sum;
}

int main()
{
        int total=seive();
        int n;

        while(scanf("%d",&n)==1)
        {
                printf("%d Sum of Of divisor:%d\n",n,Sum_Of_Divisor(n));
        }
        return 0;
}

//      48 Sum of Of divisor:124
//      72 Sum of Of divisor:195
```

Goldbach's Conjecture

For any integer *n* ($n \geq 4$) there exist two prime numbers p_1 and p_2 such that $p_1 + p_2 = n$. In a problem we might need to find the number of essentially different pairs (p_1, p_2), satisfying the condition in the conjecture for a given even number *n* ($4 \leq n \leq 2^{15}$). (The word 'essentially' means that for each pair (p_1, p_2) we have $p_1 \leq p_2$.)For example, for $n = 10$ we have two such pairs: 10 = 5 + 5 and 10 = 3 + 7.

To solve this,as $n \leq 2^{15} = 32768$, we'll fill an array primes[32768] using function seive. We are interested in primes, not greater than 32768.

The function *FindSol*(*n*) finds the number of different pairs (p_1, p_2), for which $n = p_1 + p_2$. As $p_1 \leq p_2$, we have $p_1 \leq n/2$. So to solve the problem we need to find the number of pairs (i, $n - i$), such that *i* and $n - i$ are prime numbers and $2 \leq i \leq n/2$.

```
int FindSol(int n)
{
        int i, res=0;
        for(i=2;i<=n/2;i++)
                if(flag[i] && flag[n-i])
                        res++;
        return res;
}
```

Euler's totient function

The number of positive integers, not greater than *n*, and relatively prime with *n*, equals to Euler's totient function φ (*n*). In symbols we can state that

φ (*n*) ={*a* Î N: 1 ≤ *a* ≤ *n*, gcd(*a*, *n*) = 1}

Properties

The following three simple properties of the Euler - enough to learn how to calculate it for any of:

- If p- is prime, then $\phi(p) = p - 1$.

  (This is obvious, since any number, except for the $p$relatively simple with him.)

- If p- a simple, a- a natural number, then $\phi(p^a) = p^a - p^{a-1}$.

  (Since the number of p^anot only the numbers are relatively prime form , which shares.) $pk\ (k \in \mathcal{N})p^a/p = p^{a-1}$

- If $a$and $b$are coprime, then $\phi(ab) = \phi(a)\phi(b)$("multiplicativity" of Euler's function).

  (This follows from the Chinese Remainder Theorem . Let us consider an arbitrary number $z \le ab$. Let $x$and $y$the remnants of the division $z$at $a$and b, respectively. Then $z$is relatively prime to abif and only if $z$is prime to $a$and $b$individually, or, equivalently, $x$one simply $a$and $y$relatively prime b. Applying the Chinese Remainder Theorem, we see that any pair of numbers $x$and number-one matches , which completes the proof.) $y\ (x \le a,\ y \le b)\ z\ (z \le ab)$

From this we can derive the function for each Euler $n$through its **factorization** (decomposition $n$into prime factors):

if

$$n = p_1^{a_1} \cdot p_2^{a_2} \cdot \ldots \cdot p_k^{a_k}$$

(Where everything is p_i- simple), then

$$\phi(n) = \phi(p_1^{a_1}) \cdot \phi(p_2^{a_2}) \cdot \ldots \cdot \phi(p_k^{a_k}) =$$
$$= (p_1^{a_1} - p_1^{a_1-1}) \cdot (p_2^{a_2} - p_2^{a_2-1}) \cdot \ldots \cdot (p_k^{a_k} - p_k^{a_k-1}) =$$
$$= n \cdot \left(1 - \frac{1}{p_1}\right) \cdot \left(1 - \frac{1}{p_2}\right) \cdot \ldots \cdot \left(1 - \frac{1}{p_k}\right).$$

The function fi(*n*) finds the value of φ(*n*):

For example, to find φ(616) we need to factorize the argument: $616 = 2^3 * 7 * 11$. Then, using the formula, we'll get:

φ(616) = 616 * (1 – 1/2) * (1 – 1/7) * (1 – 1/11) = 616 * 1/2 * 6/7 * 10/11 = 240.

Say you've got a problem that, for a given integer *n* ($0 < n \le 10^9$), asks you to find the number of positive integers less than *n* and relatively prime to *n*. For example, for *n* = 12 we have 4 such numbers: 1, 5, 7 and 11.

The solution: The number of positive integers less than *n* and relatively prime to *n* equals to φ(*n*). In this problem, then, we need do nothing more than to evaluate Euler's totient function.

Or consider a scenario where you are asked to calculate a function Answer(*x*, *y*), with *x* and *y* both integers in the range [1, *n*], 1 ≤ *n* ≤ 50000. If you know Answer(*x*, *y*), then you can easily derive Answer(*k***x*, *k***y*) for any integer *k*. In this situation you want to know how many values of Answer(*x*, *y*) you need to precalculate. The function Answer is not symmetric.

For example, if *n* = 4, you need to precalculate 11 values: Answer(1, 1), Answer(1, 2), Answer(2, 1), Answer(1, 3), Answer(2, 3), Answer(3, 2), Answer(3, 1), Answer(1, 4), Answer(3, 4), Answer(4, 3) and Answer(4, 1).

The solution here is to let res(*i*) be the minimum number of Answer(*x*, *y*) to precalculate, where *x*, *y* Î{1, …, *i*}. It is obvious that res(1) = 1, because if *n* = 1, it is enough to know Answer(1, 1). Let we know res(*i*). So for *n* = *i* + 1 we need to find Answer(1, *i* + 1), Answer(2, *i* + 1), … , Answer(*i* + 1, *i* + 1), Answer(*i* + 1, 1), Answer(*i* + 1, 2), … , Answer(*i* + 1, *i*).

The values Answer(*j*, *i* + 1) and Answer(*i* + 1, *j*), *j* Î{1, …, *i* + 1}, can be found from known values if GCD(*j*, *i* + 1) > 1, i.e. if the numbers *j* and *i* + 1 are not common primes. So we must know all the values Answer(*j*, *i* + 1) and Answer(*i* + 1, *j*) for which *j* and *i* + 1 are coprime. The number of such values equals to 2 * φ (*i* + 1), where φ is an Euler's totient function. So we have a recursion to solve a problem:

res(1) = 1,
res(*i* + 1) = res(*i*) + 2 * j (*i* + 1), *i* > 1

Euler's totient theorem

If n is a positive integer and a is coprime to n, then $a^{\varphi(n)} \equiv 1 \pmod{n}$.

Fermat's little theorem

If p is a prime number, then for any integer a that is coprime to n, we have

$a^{p} \equiv a \pmod{p}$

This theorem can also be stated as: If p is a prime number and a is coprime to p, then

$a^{p-1} \equiv 1 \pmod{p}$

Fermat's little theorem is a special case of Euler's totient theorem when n is prime.

How Many Digits of X^Y

Let, No. of Digits D.

Then, $D = \text{floor}\ [\ \log_{10}(X^Y)\ ] + 1$

$\qquad = \text{floor}\ [\ Y \times \log_{10}(X)\ ] + 1$

How Many Digits of N!

Let, Number of Digits D

D=floor[$\log_{10}$(N!)]+1

=floor[$\log_{10}$(1x2x3x…………xN)]+1

=floor[$\log_{10}$(1) + $\log_{10}$(2) + ……………+$\log_{10}$(N)]+1

Factorial Factors

Theorem 4 Let n denote any positive integer.

$$n! \;=\; \prod_{\substack{p \leq n \\ p\ \text{prime}}} p^{\left[\sum_{k>0} \text{IntegerPart}\left(\frac{n}{p^k}\right)\right]}$$

Proof: Apply **Theorem 2** to each of the primes less than or equal to n. **■Q.E.D.■**

Example: 38! And prime 3

ScratchInteger	**Multiplicity**
38	$M = 0$
$12 = \text{INT}(\frac{38}{3})$	$M = 12$
$4 = \text{INT}(\frac{12}{3})$	$M = 16 = 12 + 4$
$1 = \text{INT}(\frac{4}{3})$	$M = 17 = 16 + 1$
$0 = \text{INT}(\frac{1}{3})$	$M = 17 = 17 + 0$

the returned value of $M = 17$

Trailing Zeros of a Factorial

```
#include <stdio.h>

int Trailing_Zeros(int N)
{
        int sum=0;

        while(N)
        {
                sum+=N/5;
                N=N/5;
        }
        return sum;
}
int main()
{
        int N,ans;

        while(scanf("%d",&N)==1)
        {
                ans=Trailing_Zeros(N);
                printf("Trailing Zeros of %d is %d\n",N,ans);
        }
}
```

BigMod

It's Calculate: $B^P \% M$ where P,B and M integer

```
int BigMod(long long B,long long P,long long M)
{
        long long R=1;
        while(P>0)
        {
                if(P%2==1)
                {
                        R=(R*B)%M;
                }
                P/=2;
                B=(B*B)%M;
        }
        return R;
}
```

UVa Problems: 160, 294, 543, 583, 406, 686, 884,914, 10042, 10061, 10235, 10299, 10392, 10394, 10533, 10539, 10699, 10738, 10780, 10789, 10852, 11064, 11415, 11466

LightOJ Problems: 1045, 1014, 1028, 1035, 1045, 1054, 1067, 1090, 1098, 1109, 1213, 1214, 1245, 1282, 1336, 1340, 1341

//Tutorial Link:

http://www.codechef.com/wiki/tutorial-number-theory

http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=primeNumbers

http://comeoncodeon.wordpress.com

http://e-maxx.ru/algo/ (English Translation By Google Chrome)

http://theoremoftheweek.wordpress.com

Prepared By:
Forhad Ahmed (Email: forhadsustbd@gmail.com)
Ashish Pal (Email: thimpu.cse@gmail.com)