

# GeeksforGeeks

A computer science portal for geeks

CA GATE CS Practice

IDE

Q&A

GeeksQuiz

## Euler's Totient Function

Euler's Totient function  $\Phi(n)$  for an input  $n$  is count of numbers in  $\{1, 2, 3, \dots, n\}$  that are relatively prime to  $n$ , i.e., the numbers whose GCD (Greatest Common Divisor) with  $n$  is 1.

Examples:

$\Phi(1) = 1$

$\text{gcd}(1, 1)$  is 1

$\Phi(2) = 1$

$\text{gcd}(1, 2)$  is 1, but  $\text{gcd}(2, 2)$  is 2.

$\Phi(3) = 2$

$\text{gcd}(1, 3)$  is 1 and  $\text{gcd}(2, 3)$  is 1

$\Phi(4) = 2$

$\text{gcd}(1, 4)$  is 1 and  $\text{gcd}(3, 4)$  is 1

$\Phi(5) = 4$

$\text{gcd}(1, 5)$  is 1,  $\text{gcd}(2, 5)$  is 1,

$\text{gcd}(3, 5)$  is 1 and  $\text{gcd}(4, 5)$  is 1

$\Phi(6) = 2$

$\text{gcd}(1, 6)$  is 1 and  $\text{gcd}(5, 6)$  is 1,

### How to compute $\Phi(n)$ for an input $n$ ?

A **simple solution** is to iterate through all numbers from 1 to  $n-1$  and count numbers with gcd with  $n$  as 1.

Below is C implementation of the simple method to compute Euler's Totient function for an input integer  $n$ .

```
// A simple C program to calculate Euler's Totient Function
```

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

```
// Function to return gcd of a and b
```

```
int gcd(int a, int b)
```

```
{
```

```
    if (a == 0)
```

```
        return b;
```

```
    return gcd(b%a, a);
```

```
}
```

```
// A simple method to evaluate Euler Totient Function
```

```
int phi(unsigned int n)
```

```
{
```

```

    unsigned int result = 1;
    for (int i=2; i<n; i++)
        if (gcd(i, n) == 1)
            result++;
    return result;
}

// Driver program to test above function
int main()
{
    int n;
    for (n=1; n<=10; n++)
        printf("phi(%d) = %d\n", n, phi(n));
    return 0;
}

```

[Run on IDE](#)

Output:

```

phi(1) = 1
phi(2) = 1
phi(3) = 2
phi(4) = 2
phi(5) = 4
phi(6) = 2
phi(7) = 6
phi(8) = 4
phi(9) = 6
phi(10) = 4

```

The above code calls gcd function  $O(n)$  times. Time complexity of the gcd function is  $O(h)$  where  $h$  is number of digits in smaller number of given two numbers. Therefore, an upper bound on time complexity of above solution is  $O(n \log n)$  [How? there can be at most  $\log_{10} n$  digits in all numbers from 1 to  $n$ ]

Below is a **Better Solution**. The idea is based on Euler's product formula which states that value of totient functions is below product over all prime factors  $p$  of  $n$ .

$$\varphi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right),$$

The formula basically says that the value of  $\Phi(n)$  is equal to  $n$  multiplied by product of  $(1 - 1/p)$  for all prime factors  $p$  of  $n$ . For example value of  $\Phi(6) = 6 * (1-1/2) * (1 - 1/3) = 2$ .

We can find all prime factors using the idea used in [this](#) post.

- 1) Initialize : result = n
- 2) Run a loop from 'p' = 2 to sqrt(n), do following for every 'p'.
  - a) If p divides n, then
    - Set: result = result \* (1.0 - (1.0 / (float) p));
    - Divide all occurrences of p in n.
- 3) Return result

Below is C implementation of Euler's product formula.

```

// C program to calculate Euler's Totient Function

```

```
// using Euler's product formula
#include <stdio.h>

int phi(int n)
{
    float result = n;    // Initialize result as n

    // Consider all prime factors of n and for every prime
    // factor p, multiply result with (1 - 1/p)
    for (int p=2; p*p<=n; ++p)
    {
        // Check if p is a prime factor.
        if (n % p == 0)
        {
            // If yes, then update n and result
            while (n % p == 0)
                n /= p;
            result *= (1.0 - (1.0 / (float) p));
        }
    }

    // If n has a prime factor greater than sqrt(n)
    // (There can be at-most one such prime factor)
    if (n > 1)
        result *= (1.0 - (1.0 / (float) n));

    return (int)result;
}

// Driver program to test above function
int main()
{
    int n;
    for (n=1; n<=10; n++)
        printf("phi(%d) = %d\n", n, phi(n));
    return 0;
}
```

[Run on IDE](#)

Output:

```
phi(1) = 1
phi(2) = 1
phi(3) = 2
phi(4) = 2
phi(5) = 4
phi(6) = 2
phi(7) = 6
phi(8) = 4
phi(9) = 6
phi(10) = 4
```

We can avoid floating point calculations in above method. The idea is to count all prime factors and their multiples and subtract this count from n to get the totient function value (Prime factors and multiples of prime factors won't have gcd as 1)

- 1) Initialize result as n
- 2) Consider every number 'p' (where 'p' varies from 2 to  $\sqrt{n}$ ).  
If p divides n, then do following
  - a) Subtract all multiples of p from 1 to n [all multiples of p

- will have gcd more than 1 (at least p) with n]
- b) Update n by repeatedly dividing it by p.
- 3) If the reduced n is more than 1, then remove all multiples of n from result.

Below is C implementation of above algorithm.

```
// C program to calculate Euler's Totient Function
#include <stdio.h>

int phi(int n)
{
    int result = n;    // Initialize result as n

    // Consider all prime factors of n and subtract their
    // multiples from result
    for (int p=2; p*p<=n; ++p)
    {
        // Check if p is a prime factor.
        if (n % p == 0)
        {
            // If yes, then update n and result
            while (n % p == 0)
                n /= p;
            result -= result / p;
        }
    }

    // If n has a prime factor greater than sqrt(n)
    // (There can be at-most one such prime factor)
    if (n > 1)
        result -= result / n;
    return result;
}

// Driver program to test above function
int main()
{
    int n;
    for (n=1; n<=10; n++)
        printf("phi(%d) = %d\n", n, phi(n));
    return 0;
}
```

[Run on IDE](#)

Output:

```
phi(1) = 1
phi(2) = 1
phi(3) = 2
phi(4) = 2
phi(5) = 4
phi(6) = 2
phi(7) = 6
phi(8) = 4
phi(9) = 6
phi(10) = 4
```

Let us take an example to understand the above algorithm.

```

n = 10.
Initialize: result = 10

2 is a prime factor, so n = n/i = 5, result = 5
3 is not a prime factor.

The for loop stops after 3 as 4*4 is not less than or equal
to 10.

After for loop, result = 5, n = 5
Since n > 1, result = result - result/n = 4

```

### Some Interesting Properties of Euler's Totient Function

- 1) For a prime number  $p$ ,  $\Phi(p)$  is  $p-1$ . For example  $\Phi(5)$  is 4,  $\Phi(7)$  is 6 and  $\Phi(13)$  is 12. This is obvious, gcd of all numbers from 1 to  $p-1$  will be 1 because  $p$  is a prime.
- 2) For two numbers  $a$  and  $b$ , if  $\gcd(a, b)$  is 1, then  $\Phi(ab) = \Phi(a) * \Phi(b)$ . For example  $\Phi(5)$  is 4 and  $\Phi(6)$  is 2, so  $\Phi(30)$  must be 8 as 5 and 6 are relatively prime.
- 3) For any two prime numbers  $p$  and  $q$ ,  $\Phi(pq) = (p-1)*(q-1)$ . This property is used in RSA algorithm.
- 4) If  $p$  is a prime number, then  $\Phi(p^k) = p^k - p^{k-1}$ . This can be proved using Euler's product formula.
- 5) Sum of values of totient functions of all divisors of  $n$  is equal to  $n$ .

$$\sum_{d|n} \varphi(d) = n,$$

For example,  $n = 6$ , the divisors of  $n$  are 1, 2, 3 and 6. According to Gauss, sum of  $\Phi(1) + \Phi(2) + \Phi(3) + \Phi(6)$  should be 6. We can verify the same by putting values, we get  $(1 + 1 + 2 + 2) = 6$ .

- 6) The most famous and important feature is expressed in **Euler's theorem** :

The theorem states that if  $n$  and  $a$  are coprime (or relatively prime) positive integers, then

$$a^{\Phi(n)} \equiv 1 \pmod{n}$$

The **RSA cryptosystem** is based on this theorem:

In the particular case when  $m$  is prime say  $p$ , Euler's theorem turns into the so-called **Fermat's little theorem** :

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

### References:

[http://e-maxx.ru/algo/euler\\_function](http://e-maxx.ru/algo/euler_function)

[http://en.wikipedia.org/wiki/Euler%27s\\_totient\\_function](http://en.wikipedia.org/wiki/Euler%27s_totient_function)

This article is contributed by **Ankur**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



**Learn to Code  
from  
Harvard  
Professors**

**edX**  
www.edx.org

**Get Started**

Comments off Category: [Mathematical](#) Tags: [MathematicalAlgo](#)

## Related Posts:

- [How to avoid overflow in modular multiplication?](#)
- [Runge-Kutta 4th Order Method to Solve Differential Equation](#)
- [Lagrange's Interpolation](#)
- [Farey Sequence](#)
- [Adjoint and Inverse of a Matrix](#)
- [Find smallest values of x and y such that  \$ax - by = 0\$](#)
- [Sieve of Atkin](#)
- [Hungarian Algorithm for Assignment Problem | Set 1 \(Introduction\)](#)

([Login](#) to Rate and Mark)

**2.3** Average Difficulty : **2.3/5.0**  
Based on **3** vote(s)

☐

Add to TODO List

☐

Mark as DONE

[Like](#) [Share](#) 20 people like this.

Writing code in comment? Please use [code.geeksforgeeks.org](http://code.geeksforgeeks.org), generate link and share the link here.

[@geeksforgeeks](#), [Some rights reserved](#)

[Contact Us!](#)

[About Us!](#)

[Advertise with us!](#)