Euler's φ Function

Lab Assignment 2 - Public Key Cryptography, UBB-CS Year 3

Popa Alex Ovidiu, group 936

Problem Statement: Write an algorithm for computing the value of Euler's function for natural numbers. For a given value v and a given bound b, list all natural numbers less than b which have v as the value of Euler's function.

Proposed solution:

Compute the Euler function for a value n using Euler's product formula.

Euler's Product Formula

The first implemented function is nothing more than the theorem presented in the second lecture, the formula of which is:

$$\varphi(n) = n\left(1 - \frac{1}{p_1}\right)\left(1 - \frac{1}{p_2}\right)\cdots\left(1 - \frac{1}{p_j}\right)$$

where

$$p_1, p_2, \cdots p_j$$

are the prime factors of n. Or, equivalently,

$$\varphi(n) = n \cdot \prod_{p|n}(1 - \frac{1}{p}). \tag{1}$$

To prove (1), we will use two the other theorems stated in the lecture, namely the formula for $\varphi(n = p^k)$ (for p prime factor, k its power), as well as the multiplicative property of φ:

$$
\begin{aligned}
\varphi(n) &= \varphi(p_1^{k_1})\varphi(p_2^{k_2})\cdots\varphi(p_j^{k_j}) \\
&= p_1^{k_1}(1-\frac{1}{p_1})p_2^{k_2}(1-\frac{1}{p_2})\cdots p_j^{k_j}(1-\frac{1}{p_j}) \\
&= p_1^{k_1}p_2^{k_2}\cdots p_j^{k_j}(1-\frac{1}{p_1})(1-\frac{1}{p_2})\cdots(1-\frac{1}{p_j}) \\
&= n(1-\frac{1}{p_1})(1-\frac{1}{p_2})\cdots(1-\frac{1}{p_j}).
\end{aligned}
$$
$$\implies (1)$$

Running example for $\varphi(10)$:

1. n = 10, eulercount = 10
2. primefactor = 2 is a prime factor, so n = n/primefactor = 5, eulercount = 10 * (1.0 - 1.0/2.0) = 10 * (1.0 - 0.5) = 10 * 0.5 = 5.0
3. primefactor = 3 is not a prime factor.
4. 4*4=16 > 5 => the for loop ends.
5. After the for loop, eulercount = 5.0, n = 5
6. n > 1 => eulercount = 5.0 * (1.0 - 1.0/5.0) = 5.0 * 0.8 = 4.0
7. $\lfloor n \rfloor$ = 4 is returned

```
<<EulersFunctionFractions>>=
from math import sqrt
def phiFractions(n):
    euler_count = n  # start from n as the theorem shows
    for prime_factor in range(2,int(sqrt(n))+1):
        if n % prime_factor == 0:
            while n % prime_factor == 0:
                n //= prime_factor
            euler_count = euler_count * (1.0 - (1.0 / float(prime_factor)))
    if n > 1: #check for sqrt edge case
        euler_count = euler_count * (1.0 - (1.0 / float(n)))
    return int(euler_count)
@
```

The function below implements the first formula for testing the Euler function from the slides.

```
<<EulerTestDivisors>>=
<<EulersFunctionFractions>>
def eulerTestDivisors(n):
    d = 1
    sum = 0
    while d<=n//2:
        if n%d==0:
            sum+=phiFractions(d)
        d+=1
    sum+=phiFractions(n)
    return sum == n
@
```

The function below implements the second formula for testing the Euler function from the slides, namely the one proposed by Prof. Andrica, which says that the respectiv sum has to converge to value 2.

```
<<EulerTestAndrica>>=
<<EulersFunctionFractions>>
from math import ceil
def eulerTestAndrica(n):
    sum = 0.0
    for i in range(1, n+1):
        sum += (phiFractions(i)) / (2.0**i - 1)
    return ceil(sum)==2.0
@
```

Group the two tests into a single one and run it with a default value of 1000, which can be changed accordingly.
Please note that 1000 is already kind of testing the limits of python's numeric capabilities, and testing for 2000 will result in an overflow.

```
<<MainTest>>=
<<EulerTestDivisors>>
<<EulerTestAndrica>>
def runTests(n=1000):
    assert(eulerTestAndrica(n) and eulerTestDivisors(n) is True)
    print("Tests passed!")
@
```

The actual algorithm concerning the value v and bound b is trivial, in the sense that it iterates until the bound and compares the result of the euler function with the value and adds the current number to a list if the two are equal. The function below does just that, namely prints out the values of the euler function for each bound which matches the corresponding value.
Furthermore, it returns the running time taken by the algorithm, to be used in the main function.

```
<<RunnerFunction>>=
```

```
<<EulersFunctionFractions>>
from timeit import default_timer as time
def runner(value, bound, function):
    result=[]
    start = time()
    for i in range(1, bound):
        if function(i) == value:
            result.append(i)
    end = time()
    print("(" + str(value) + "," + str(bound) + ")->" + str(result)
+ "->time elapsed:" + str(end-start))

@
```

The function below plots the histogram for a bound b, defaulted to 1000. First it keeps in a dictionary the value of the Euler function for each number up to the bound b, and then fetches the top 5 most common values.
For b=1000, those are: 240, 288, 192, 144, 216. One can notice that the common values result from combinations of the powers of 2,3 and 5, for instance:

$$240 = 2^4 * 3 * 5$$
$$288 = 2^5 * 3^2$$

Furthermore, for large numbers, one can notice from the plotted distribution that the values with the highest frequency are mostly located in the first 40%, i.e. for b=10000, most of them are <=4000.

```
<<Histogram>>=
<<EulersFunctionFractions>>
import matplotlib.pylab as pyplt

def histogram(b=1000):
    data = {i:0 for i in range(1, b+1)}
    for n in range(1,b+1):
        data[phiFractions(n)]+=1

    results = sorted(data.items(), key=lambda pair: pair[1])
    values = [pair[0] for pair in results]

    values.reverse()
    print("For bound = {}, ".format(b) + "the top 5 repeated values are: " + str(values[:5])

    resultsToPlot = sorted(data.items(), key=lambda pair: pair[0])
    xAxis = [pair[0] for pair in resultsToPlot]
```

```
    yAxis = [pair[1] for pair in resultsToPlot]

    pyplt.plot(xAxis, yAxis, color='red')
    pyplt.title("Euler function distribution for b={}".format(b))
    pyplt.show()
@
```

In the main function the runner function is called for a value and a bound, which can be arbitrarily set.

```
<<*>>=

<<RunnerFunction>>
<<MainTest>>
<<Histogram>>

def main():
    value = 12
    bound = 468
    runner(value,bound,phiFractions)

runTests()
main()
histogram()

@
```