Project Euler

Sean Go

January 2018

1 Problem 1

If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23. Find the sum of all the multiples of 3 or 5 below 1000.

1.1 Brute force or Comprehension

Haskell

```
# dough
sum [n | n <- [1..1000-1], n `mod` 5 == 0 || n `mod` 3 == 0]
# lazcatluc
sum [3,6..999] + sum [5,10..999] - sum [15,30..999]
```

Listing 1: P1. Haskell Comprehension

Python

```
# johanlindberg
sum([x for x in range(1000) if x % 3== 0 or x % 5== 0])
```

Listing 2: P1. Python Comperhension

Assembly

```
        ; for each integer from 1 to 1000
        mov ecx, 3

        mov esi, 3
        mov edi, 5

        xor ebx, ebx        ; sum

_0:        mov eax, ecx
        xor edx, edx
        div esi
        test edx, edx
        je _yes

        mov eax, ecx
        xor edx, edx
        div edi
        test edx, edx
        jne _no

_yes:        add ebx, ecx

_no:        inc ecx
        cmp ecx, 1000
        jne _0
```

Listing 3: P1. Assembler Brute Force

1.2 Math

The sum of consecutive integers from 1 to c is

$$\sum_{k=1}^{c} k = \frac{c(c+1)}{2}$$

c , in this case, represents the count of numbers under N divisible by d. This is

$$c = \left\lfloor \frac{N-1}{d} \right\rfloor$$

The sum of all $3n$ is $\dfrac{3t(t+1)}{2}$, wehre $t = \left\lfloor \dfrac{1000-1}{3} \right\rfloor$;

The sum of all $5n$ is $\dfrac{5t(t+1)}{2}$, wehre $t = \left\lfloor \dfrac{1000-1}{5} \right\rfloor$;

The sum of all $15n$ is $\dfrac{15t(t+1)}{2}$, wehre $t = \left\lfloor \dfrac{1000-1}{15} \right\rfloor$;

The code:

```python
def PE1(N):
    t=(N-1)//3; f=(N-1)//5; x=(N-1)//15
    return 3*t*(t+1)/2 + 5*f*(f+1)/2 - 15*x*(x+1)/2

print PE1(1000)
```

Listing 4: P1. Python Math

2 Problem 2

Each new term in the Fibonacci sequence is generated by adding the previous two terms. By starting with 1 and 2, the first 10 terms will be:
$1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...$
By considering the terms in the Fibonacci sequence whose values do not exceed four million, find the sum of the even-valued terms.

2.1 Brute Force

Assember. Only x86 has an easy way to swap variables (XADD) - other langauges require a temp varaible or XOR trick. Only x86 has an easy way to test for even numbers (bit 0 is zero) - other language use Mod() function.

```
        mov ecx, 1
        mov edx, 0
        xor ebx, ebx        ; sum
_0:        test ecx, 1
        jne _odd
_even:        add ebx, ecx
_odd:        xadd ecx, edx
        cmp ecx, 1000000
        jc _0
```

Listing 5: P2. Assembly. bitRAKE

2.2 method 2

This may be a small improvement. The Fibonacci series is:
$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610...$
Now, replacing an odd number with O and an even with E, we get:
$O, O, E, O, O, E, O, O, E, O, O, E, O, O, E...$
And so each third number is even. We don't need to calculate the odd numbers. Starting from an two odd terms x, y, the series is:
$x, y, x + y, x + 2y, 2x + 3y, 3x + 5y$

```
def calcE():
        x = y = 1
        sum = 0
        while (sum < 1000000):
                sum += (x + y)
                x, y = x + 2 * y, 2 * x + 3 * y
        return sum
```

Listing 6: P2. Python. Begoner

2.3 method 3

So, let define $G_n = F_{3n}$ and $S_n = \sum i = 1^n G_i$:
$Gn : 0, 2, 8, 34, 144, 610, 2584, ...$
$Sn : 0, 2, 10, 44, 188, 798, 3382, ...$
we can proove that $G_n = F_{3n}$ can be constructed more directly by :
$G_0 = 0$, $G_1 = 2$ and $G_n + 2 = 4G_{n+1} + G_n$

```
def PE002(M=4000000):
  a, b, S  =  0, 2, 0
  while b<=M:
    a, b, S =  b, a+4*b, S+b
  return S
```

Listing 7: P2. Python. Begoner

2.4 method 3

So, let define $G_n = F_{3n}$ and $S_n = \sum i = 1^n G_i$:
$Gn : 0, 2, 8, 34, 144, 610, 2584, ...$
$Sn : 0, 2, 10, 44, 188, 798, 3382, ...$
we can proove that $G_n = F_{3n}$ can be constructed more directly by :
$G_0 = 0$, $G_1 = 2$ and $G_n + 2 = 4G_{n+1} + G_n$

```
def PE002(M=4000000):
  a, b, S  =  0, 2, 0
  while b<=M:
    a, b, S =  b, a+4*b, S+b
  return S
```

Listing 8: P2. Python. Begoner

2.5 method 4

Can we do better ? Yes, we can proove that $S_n = (G_n + G_{n+1}\grave{}2)/4$ Now we can save 1/3 of time and memory.

```
def PE002(M=4000000):
  a, b =  0, 2
  while b<=M:
    a, b = b, a + 4*b
  #return (b+a-2)//4 # for the result
  return ((b+a-2)//4).bit_length() # for the bit length of the result
```

Listing 9: P2. Python. Begoner

2.6 method 5

Can we do better ? Yes with the base of the second method It exists method to estimate the indice of n which give the answer,

```
n = int(log(M, 2+5**0.5))# an irrationnal based logarithm
```

n is just an estimation, but we could start here the few left iterations.

Now, we could answer with a $O(log(log(M)))$ complexity !!! Example : with $M = 4*10^(10^5)$ (Yes, mega huge number) the previous method give the answer : a 332194 bits number in 2s. My optimised algo in $O(log(log(M)))$ give me the same in 34ms, and other results

```
PE002(4*10**(10**5)) -> 332194 bits in 34ms
PE002(4*10**(10**6)) -> 3321931 bits in 1.3s
PE002(4*10**(10**7)) -> 33219284 bits in 54s
```

(40MB of total memory print) All in pure Python3, interpreted, 3GHz single thread.

2.7 Estimate

(from RudyPenteado). ϕ (golden ratio) is the approximate ratio between two consecutive terms in a Fibonacci sequence. The ratio between consecutive even terms approaches $\phi^3 \approx 4.236068$ because each 3rd term is even. Use a calculator and round the results to the nearest integer when calculating the next terms:
$2, 8, 34, ..$ multiplying by 4.236068 each time:
$144, 610, 2584, 10946, 46368, 196418, 832040$
The sum is 1089154

3 Problem 3

The prime factors of 13195 are 5, 7, 13 and 29. What is the largest prime factor of the number 600851475143?

3.1 Brute Force

d, n = 3, 600851475143 while (d * d ¡ n): if n else: d += 2 print n

List of source codes