Project Euler, Problem 2:
Even Fibonacci Numbers

John Butler

Contents

1 Problem Description **2**

2 Theorems whose results aid in my solution **2**

2.1 Only every third Fibonacci number is even 2

2.2 $F_n = \frac{(1+\sqrt{5})^n-(1-\sqrt{5})^n}{2^n\sqrt{5}}$ 2

3 Application to Code **3**

4 Comlexity Analysis/ Further Optimizations **3**

1 Problem Description

Find the sum of all $F_n < 4,000,000$ such that F_n is even, where

$$F_n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F_{n-1} + F_{n-2} & n \geq 2 \end{cases}$$

2 Theorems whose results aid in my solution

2.1 Only every third Fibonacci number is even

Proof by induction:
case $n \in \{0, 1, 2\}$

$F_0 = 0$	$F_1 = 1$	$F_2 = F_1 + F_0$
	$F_1 = 0 + 1$	
$F_0 = 2 \cdot 0$	$F_1 = 2 \cdot 0 + 1$	$F_2 = F_1 + 0$
Let $k_1 = 0$	Let $k_2 = 0$	
$\therefore F_0 = 2k_1$	$\therefore F_1 = 2k_2 + 1$	$F_2 = F_1$
$\therefore F_0$ is even	$\therefore F_1$ is odd	$\therefore F_2$ is odd

Assume F_{n-3} is even, F_{n-2} is odd, and F_{n-1} is odd

$F_n = F_{n-1} + F_{n-2}$	$F_{n+1} = F_n + F_{n-1}$	$F_{n+2} = F_{n+1} + F_n$
$F_n =$ odd + odd	$F_{n+1} =$ even + odd	$F_{n+2} =$ odd + even
$F_n =$ even	$F_{n+1} =$ odd	$F_{n+2} =$ odd

Therefore the first of each three subsequent Fibonacci numbers are even.

2.2 $F_n = \frac{(1+\sqrt{5})^n - (1-\sqrt{5})^n}{2^n\sqrt{5}}$

$$\begin{aligned}
\text{Assume } F_n &= cr^n. \\
F_n &= F_{n-1} + F_{n-2} \\
\implies cr^n &= cr^{n-1} + cr^{n-2} \\
&= cr^n(r^{-1} + r^{-2}) \\
\therefore 1 &= \frac{1}{r} + \frac{1}{r^2} \\
\therefore r^2 &= r + 1 \text{ Assuming } r \neq 0 \\
\therefore r^2 - r - 1 &= 0 \\
\therefore r &= \frac{1 \pm \sqrt{(-1)^2 - 4 \cdot 1 \cdot (-1)}}{2} \text{ using the quadratic formula} \\
&= \frac{1 \pm \sqrt{5}}{2} \\
\text{Say } r_1 &= \frac{1 + \sqrt{5}}{2} \\
\text{and } r_2 &= \frac{1 - \sqrt{5}}{2}
\end{aligned}$$

$$
\begin{aligned}
\text{Now assume } F_n &= \alpha r_1^n + \beta r_2^n \\
F_0 &= 0 \\
\therefore 0 &= \alpha r_1^0 + \beta r_2^0 \\
&= \alpha + \beta \\
\therefore \alpha &= -\beta \\
F_1 &= 1 \\
\therefore 1 &= \alpha r_1^1 + \beta r_2^1 \\
&= \alpha r_1 + \beta r_2 \\
&= -\beta r_1 + \beta r_2 \\
&= \beta(r_2 - r_1) \\
&= \beta\left(\frac{1-\sqrt{5}}{2} - \frac{1+\sqrt{5}}{2}\right) \\
&= \beta\left(\frac{1-\sqrt{5}-1-\sqrt{5}}{2}\right) \\
&= \beta\left(-\frac{2\sqrt{5}}{2}\right) \\
1 &= \beta\left(-\sqrt{5}\right) \\
\therefore \beta &= -\frac{1}{\sqrt{5}} \\
\therefore \alpha &= \frac{1}{\sqrt{5}} \\
\therefore F_n &= \frac{1}{\sqrt{5}} \cdot \left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}} \cdot \left(\frac{1-\sqrt{5}}{2}\right)^n \\
&= \frac{(1+\sqrt{5})^n - (1-\sqrt{5})^n}{2^n\sqrt{5}}
\end{aligned}
$$

3 Application to Code

Now that we know that every third Fibonacci number is even, we don't have to bother checking if a number is even. Instead, since we know that $F_0 = 0$ is the first Fibonacci number which is even, we can add F_0 to our sum, then compute F_1, F_2, and then add $F_3 = 2$ to our sum, and so on. Granted, checking if a number is even is computationally light.

Additionally, F_0 doesn't affect our sum, so we can start with $F_3 = 2$; Project Euler begins the Fibonacci sequence at F_2 in the problem description, reinforcing this fact.

4 Comlexity Analysis/ Further Optimizations

For this section, assume that n is the index of the maximum Fibonacci number we must execute, and L is the limit $(4,000,000)$

Considering the definition, it calculating the Fibonacci numbers would seem to be most natural to use a top-down recurrsive algorithm, but this would be incredibly slow, $O(2^n)$ for *each* number, meaning we end up with a time complexity of $O(n2^n)$. Of course, this is a common example used by programming

courses for recursion, so this may be many peoples' first approach. There are certainly optimizations to this algorithm, such as storing the result of previous computations, so we don't have to recompute F_{n-1} and F_{n-2}, bringing time complexity down to $O(1)$, but there is still a linear space complexity. We don't need to store F_{n-3}, since we're always only computing the next number. Fully optimized this way, the time complexity should be $O(n)$ and the space complexity should be $O(1)$

The algorithm I'm using uses a bottom up approach, which also has a space complexity of $O(1)$ and a time complexity of $O(n)$. The idea is to store only F_{n-1} and F_{n-2}, so that we can compute F_n. After that we shift over what we're storing, so that we forget F_{n-2} and keep F_n and F_{n-1} in order to calculate F_{n+1}.