Project Euler

Prematurely optimized & overengineered solutions

Oistein Sovik

November 11, 2017

Contents

Preface

This project has been ressurected twice, and thus this version might be thought of as Project Euler 1.2. One main difference is that the codebase is now avaible on Github:

https://github.com/Oisov/Project-Euler.

Another major change is the addition of several other programming languages, with the most prominent being **Julia**. The code in these notes are now written in Julia as it closely resembles psudocode, with just a few minor changes.

1. The logical expressions $||, \&\&, !$ has been replaced with OR, AND, NOT
2. Variable declaration is changed from $=$ to $\rightarrow$
3. Variable comparison $==$ is changed to $=$
4. The $+=$ syntax used to represent increasing a variable is replaced with $\uparrow$

What has *not* changed is the purpose of this document. The goal is still to present well written, clever and interesting solutons to Project Euler.

I have seen too many hastily written solutions to these problems. While the solutions work, they are not written in a clear matter, single letter variablenames and no function declarations. Code like this is incredibly hard to read for someone who has not produced the mess, and even for the author the code can be incomprehensible after just a few weeks.

While I can not hope to remedy every badly written code with these notes, hopefully someone who reads these notes, or browses the github repo finds some inspiration.

Project Euler 1: Even Fibonacci numbers

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.

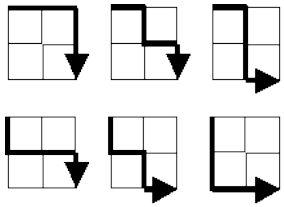
Algorithm 1.1 Naive recursive solution

```
def fib(n):
    if n in [0, 1]:
        return n
    else:
        return fib(n-1) + fib(n-2)

function PE_002_recursive_naive(limit)
    n ← 0
    total ← 0
    while fib(n) < limit
        n ↑ 1
        if fib(n) % 2 = 0:
            total ↑ fib(n)
    return total
```

Project Euler 15: Lattice paths

Starting in the top left corner of a 2×2 grid, and only being able to move to the right and down, there are exactly 6 routes to the bottom right corner.



How many such routes are there through a 20×20 grid?

15.1 Recursive solution

Algorithm 15.2 Recursive route-counting function

```
function PE_015_recursive(rows←20, columns←20)
    path_sum ← Dict{Any, BigInt}()

    function count_routes(m, n)
        if n = 0 OR m = 0
            return 1
        end

        try
            return path_sum[(m, n)]
        end
        path_sum[(m, n)] ← count_routes(m, n - 1) + count_routes(m - 1, n)
        return path_sum[(m, n)]
    end
    count_routes(rows, columns)
end
```

To speed up this code we could have defined path_sum[(m,n)]← path_sum[(n,m)] and as we know that there are only one way to reach each of the points on the first column and first row. However, while this halves the number of function calls this is not where the bottleneck lies. Most of the time is spent looking up dictionary values.

15.2 Iterative solution

While the memoized recursion solves the problem, it is not only slow, but requires more memory, and some languages have trouble with deeply-nested recursive calls.

If we could translate our recursion into and *iterative* solution using bottom-up programming technique called *dynamic programming* we would solve most of the concerns raised above.

Algorithm 15.3 Recursive route-counting function

```
function PE_015_iterative(n ← 20, m ← 20)
    rows, columns ← n+1, m+1
    grid ← ones(BigInt, rows, columns)

    for i ← 2:rows
        for j ← 2:columns
            grid[i, j] ← grid[i-1, j] + grid[i, j-1]
        end
    end

    grid[rows, columns]
end
```

15.3 Combinatorial solution

Algorithm 15.4 Recursive route-counting function

```
function PE_015(rows ← 20, columns ← 20)
    binomial(BigInt(rows + columns), BigInt(columms))
end
```