Selected Project Euler Exercises

Oleg Sivokon

<2015-10-15 Thu>

Contents

1 Foreword

This document contains several Project Euler problems solved in Python and C++. It was written using org-mode for Emacs. To uninitiated, this means that this document was originally written in plain text and later converted to LaTeX source. This also means that this document contains Babel source code blocks which allow code evaluation during conversion to LaTeX.

I made the decision to structure the submission in this way because this allows for controlled and at the same time typographically appealing presentation of my work. While I was required to pretend to work on a "real" project, I find it hard to think about a set of disparate number-theoretic problems as comprising a software engineering project in the common sense of the word. It seemed like an article might prove better experience due to having more room for discussion illustrated by small code snippets.

Few disclaimers are due. I am not really a C++ programmer. This project, perhaps, contains the largest amount of C++ code I've ever written. The way I worked through the problems was to first solve them in Python and then to translate them into C++. While my Python code would probably meet the industry standards, I excpect my C++ code to be subpar, both as not-idiomatic and inefficient, or, perhaps even outright wrong in its dealing with memory allocation and other complications of C++. Basically, I worked on it until I could produce the result I knew to be correct, and left it immediately afterwards.

As of time of this writing, I didn't solve the 136^{th} problem, though I do have some thoughts on how to approach it. I've written some of it in the last paragraph.

The project has a Makefile, and, provided you have all the dependencies, you should be able to reproduce this document. Plain "vanilla" Emacs should be enough for the `org-mode` and `babel` dependencies. My Python code doesn't use anything outside of standard library, while C++ code relies on GNU MP library for biginteger support. This means that you need to have development header `gmpxx.h` on your includes path. On top of the typical LaTeX installation you would need the minted package (for coloring the source code blocks).

Not all code is presented in the document. The auxilary functions are found in `./python` and `./cxx` directories. There is also some documentation generated for them using Doxygen. It was difficult to include the documentation directly in this document (LaTeX style clashes etc.) thus the documentation takes up a separate document.

2 Exercises

2.1 1000-digits Fibonacci number PE-25

2.1.1 Problem Statement

The Fibonacci sequence is defined by the recurrence relation:

$$F_n = F_{n-1} + F_{n-2},\ \textit{where}\ F_1 = 1\ \textit{and}\ F_2 = 1\ .$$

Hence the first 12 terms will be:

$$\begin{aligned}
F_1 &= 1 \\
F_2 &= 1 \\
F_3 &= 2 \\
F_4 &= 3 \\
F_5 &= 5 \\
F_6 &= 8 \\
F_7 &= 13 \\
F_8 &= 21 \\
F_9 &= 34 \\
F_{10} &= 55 \\
F_{11} &= 89 \\
F_{12} &= 144
\end{aligned}$$

The 12^{th} term, F_{12}, is the first term to contain three digits.

What is the index of the first term in the Fibonacci sequence to contain 1000 digits?

2.1.2 Discussion

Start with some initial guess: $fib(1000)$ and double the size until we find a number with more than 1000 digits once done, subtract the half of the last increment, depending on whether we overshoot or undershoot subtract or add the quarter of the last increment and so on, until we find two adjacent Fibonacci numbers such that the first has at most 999 digits and the second has 1000 or more digits. This will ensure that we calculate at most $log(1000) \approx 10$ operations. Afterwards, because there could be several Fibonacci numbers of the same length, we'll find the smallest one using linear search (this won't take more than $log(2^4) = 4$ operations.)

2.1.3 Solution

```
from python.combinatorics import fib, digits

n = 1000
guess = fib(n)
while len(digits(guess)) < 1000:
    n *= 2
    guess = fib(n)
m = n // 4
n -= m
while m > 1:
    guess = fib(n)
    m //= 2
    if len(digits(guess)) > 1000:
        n -= m
    else:
        n += m
while len(digits(guess)) >= 1000:
    n -= 1
    guess = fib(n)
return n + 1
```

```
4782
```

```
int n = 1000;
mpz_class guess = fib(n);

while (mpz_sizeinbase(guess.get_mpz_t(), 10) < 1000) {
    n *= 2;
    guess = fib(n);
}
int m = n / 4;
n -= m;
while (m > 1) {
    guess = fib(n);
    m /= 2;
    if (mpz_sizeinbase(guess.get_mpz_t(), 10) > 1000) {
        n -= m;
    } else {
        n += m;
    }
}
while (mpz_sizeinbase(guess.get_mpz_t(), 10) >= 1000) {
    n--;
    guess = fib(n);
}
std::cout << n + 1;
```

```
4782
```

2.2 Goldbach's other conjecture PE-46

2.2.1 Problem statement

It was proposed by Christian Goldbach that every odd composite number can be written as the sum of a prime and twice a square.

$$\begin{aligned}
9 &= 7 + 2 \times 12 \\
15 &= 7 + 2 \times 22 \\
21 &= 3 + 2 \times 32 \\
25 &= 7 + 2 \times 32 \\
27 &= 19 + 2 \times 22 \\
33 &= 31 + 2 \times 12
\end{aligned}$$

It turns out that the conjecture was false.

What is the smallest odd composite that cannot be written as the sum of a prime and twice a square?

2.2.2 Discussion

There doesn't seem to be an easier way than just a brute-force search. In other words, the code simply generates subsequent odd composite numbers and tests whether the number complies with the contstraints of conjecture.

2.2.3 Solution

```python
import math
from python.primes import odd_composite, primes

def decomposes_goldbach(n):
    """
    Returns True iff n can be decomposed into p + 2 * i^2, where
    p is a prime and i is an integer.
    """
    for p in primes():
        if p > n:
            return False
        rest = n - p
        if rest % 2 == 0 and int(math.sqrt(rest // 2)) ** 2 == rest // 2:
            return True

for c in odd_composite():
    if not decomposes_goldbach(c):
        return c
```

```
5777
```

```
primes ps;
int result, p, c;
bool found = false;

while (!found) {
    p = ps.next();
    c = p + 2;
    p = ps.next();
    while (c < p) {
        if (!decomposes_goldbach(c)) {
            result = c;
            found = true;
            break;
        }
        c += 2;
    }
}
std::cout << result;
```

```
5777
```

2.3 Largest sum of consequtive primes PE-50

2.3.1 Problem statement

The prime 41, can be written as the sum of six consecutive primes:

$$41 = 2 + 3 + 5 + 7 + 11 + 13$$

This is the longest sum of consecutive primes that adds to a prime below one-hundred.

The longest sum of consecutive primes below one-thousand that adds to a prime, contains 21 terms, and is equal to 953.

Which prime, below one-million, can be written as the sum of the most consecutive primes?

2.3.2 Discussion

In the preparation phase the code generates all primes which add up to less than a million, while at the same time calculating sums of the primes generated so far. This acts as a cache for the sums we would have to calculate repeatedly whenever we'd verify every other sequence of primes.

During the next step the code chooses sums in order of decreasing number of summands, iteratively updating the maximum prime for the longest sequence found so far. Once no sums of the longest-so-far sequence can be found, the lagorithm terminates.

2.3.3 Solution

```python
from python.primes import primes, is_prime

sieve, prefixes, max_prime, longest_seq = [], [0], 0, 0
for p in primes():
    if prefixes[-1] + p > 1000000:
        break
    sieve.append(p)
    prefixes.append(prefixes[-1] + p)
terms = 1
for i in range(len(prefixes)):
    for j in range(i + terms, len(prefixes)):
        n = prefixes[j] - prefixes[i]
        if j - i > terms:
            is_p, sieve = is_prime(n, sieve)
            if is_p:
                terms, max_prime = j - i, n
return max_prime
```

```
997651
```

```
primes ps(1000000);
std::vector<int> sums(ps.size());
int max_prime;
std::size_t terms = 1;
std::size_t top;

sums[0] = ps[0];
for (std::size_t i = 1; i < ps.size(); i++) {
    if (sums[i - 1] + ps[i] > 1000000) {
        top = i;
        break;
    }
    sums[i] = sums[i - 1] + ps[i];
}
for (std::size_t i = 0; i < top; i++) {
    for (std::size_t j = terms + i; j < top; j++) {
        int n = sums[j] - sums[i];
        if (j - i > terms && ps.is_prime(n)) {
            terms = j - i;
            max_prime = n;
        }
    }
}
std::cout << max_prime;
```

```
997651
```

2.4 Poker game PE-54

2.4.1 Problem statement

In the card game poker, a hand consists of five cards and are ranked, from lowest to highest, in the following way:

High Card Highest value card.

One Pair Two cards of the same value.

Two Pairs Two different pairs.

Three of a Kind Three cards of the same value.

Straight All cards are consecutive values.

Flush All cards of the same suit.

Full House Three of a kind and a pair.

Four of a Kind Four cards of the same value.

Straight Flush All cards are consecutive values of same suit.

Royal Flush Ten, Jack, Queen, King, Ace, in same suit.

The cards are valued in the order:

$$2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King, Ace.$$

If two players have the same ranked hands then the rank made up of the highest value wins; for example, a pair of eights beats a pair of fives (see example 1 below). But if two ranks tie, for example, both players have a pair of queens, then highest cards in each hand are compared (see example 4 below); if the highest cards tie then the next highest cards are compared, and so on.

Consider the following five hands dealt to two players:

Player 1	Player 2	Winner
5H 5C 6S 7S KD	2C 3S 8S 8D TD	Player 2
Pair of Fives	Pair of Eights	
5D 8C 9S JS AC	2C 5C 7D 8S QH	Player 1
Highest card Ace	Highest card Queen	
2D 9C AS AH AC	3D 6D 7D TD QD	Player 2
Three Aces	Flush with Diamonds	
4D 6S 9H QH QC	3D 6D 7H QD QS	Player 1
Pair of Queens	Pair of Queens	
Highest card Nine	Highest card Seven	
2H 2D 4C 4D 4S	3C 3D 3S 9S 9D	Player 1
Full House	Full House	
With Three Fours	with Three Threes	

The file, poker.txt, contains one-thousand random hands dealt to two players. Each line of the file contains ten cards (separated by a single space): the first five are Player 1's cards and the last five are Player 2's cards. You can assume that all hands are valid (no invalid characters or repeated cards), each player's hand is in no specific order, and in each hand there is a clear winner.

How many hands does Player 1 win?

2.4.2 Discussion

Player's hand will be represented by a class `Hand`. This class is initalized with the raw card data. During initialization the cards are sorted in order from least to highest denomination. `Hand` class will aslo implement operators required for comparison. Once such operator is called, the code will determine (and cache):

1. The value of the hand. Hands are given values using the following scheme:
    - If no special combination is found (s.a. *flush*), then the hand is worth as much as its highest denomination card.
    - Special denominations recieve one point more than the highest card, plus their relative rank, i.e. *one pair* receives 15, *two pairs* recieves 16 and so on.
    - Ties aren't broken at this time.
2. The stretches of cards of the same denomination.
3. If the code encounters a tie, it tries to break it using the rules given below.
    - Given the `stretches` information the code will select only the cards that will influence the decision.
    - Compare selected cards in order from highest to lowest denomination.

The `cards_txt` file can be found in `./etc/p054_poker.txt`.

2.4.3 Solution

```python
from python.cards import Hand

with open(cards_txt, 'r') as f:
    wins = 0
    for line in f:
        cards = line.strip().split(" ")
        if Hand(cards[:5]) > Hand(cards[5:]):
            wins += 1
    return wins
```

```
std::ifstream in(poker_txt);
std::string line;
int result = 0;
while (std::getline(in, line)) {
    hand a(line.substr(0, 3 * 5));
    hand b(line.substr(3 * 5, 3 * 10 - 1));
    if (a > b) {
        result++;
    }
}
std::cout << result;
```

376

2.5 Unique arithmetic progressions

2.5.1 Problem statement

The positive integers, x, y, and z, are consecutive terms of an arithmetic progression. Given that n is a positive integer, the equation, $x^2 - y^2 - z^2 = n$, has exactly one solution when $n = 20$:

13^2 - 10^2 - 7^2 $=$ 20

In fact there are twenty-five values of n below one hundred for which the equation has a unique solution.

How many values of n less than fifty million have exactly one solution?

2.5.2 Discussion

First, we could reduce the number of variables:

$$\begin{aligned}
a^2 - b^2 - c^2 = n &\iff \\
(d + 2x)^2 - (d + x)^2 - d^2 = n &\iff \\
\textit{from arithmetic progression definition} & \\
d^2 + 4dx + 4x^2 - d^2 - 2dx - x^2 - d^2 = n &\iff \\
-d^2 + 2dx + 3x^2 = n &\iff \\
(d + x)^2 - 2(d^2 - x^2) = n &\iff \\
(d + x)^2 - 2(d + x)(d - x) = n &\iff \\
(d + x - 2d + 2x)(d + x) = n &\iff \\
(3x - d)(d + x) = n &
\end{aligned}$$

We also know that both d and x must be positive. My guess is that we could look for solution by differentiating this equation with respect to either of these variables.

I was looking for ways to give upper bounds to either one of the variables, s.a. to make sure that if I try to verify that n is unique, I would only need to do finite amount of work, but I don't have any good idea so far. I have a feeling that both $x < \sqrt{n}$ and $d < \sqrt{n}$, but I cannot prove it.