Romanian Master of Informatics
6th Edition, Bucharest, 10th - 13th of October 2018

Squirrel – solution

Author: Cristian Frâncu

The problem consists of a relatively easy recursion plus a massive number of coprimality tests. We discuss several ways of testing coprimality.

Euclid's GCD algorithm

By definition, two numbers are coprime if their greatest common divisor (GCD) is 1. Therefore, we can answer tests using Euclid's algorithm. This can require up to 25 modulo operations per test, given that the 25th Fibonacci number is 46368 (see this Wikipedia article for an in-depth analysis). The average number of operations will be lower, but still significant.

This very basic approach is expected to win 15 points.

Binary GCD algorithm

Euclid's algorithm uses very inefficient modulo operations. Instead, the binary GCD algorithm performs no divisions, only simple bit operations (shifting and OR-ing). Wikipedia has a comprehensive explanation and code sample. The code can further be optimized by using built-in functions (or precomputed tables) for shifting off all the trailing zeroes in one operation instead of using a loop.

This runs roughly twice as fast as Euclid's algorithm and is expected to win 25 points.

Factor merging

So far we answered coprimality tests by computing the actual GCD. For further improvements, we recognize that any common divisor suffices.

One idea is to compute the sorted list of prime factors, $L(x)$, for every $x \in [1, 50\,000]$.

For example, $L(63) = \{3, 7\}$ and $L(180) = \{2, 3, 5\}$. Two numbers are coprime if their factor lists have no common elements. Since $L(63) \cap L(180) = \{3\} \neq \varnothing$, it follows that 63 and 180 are not coprime. The actual GCD is 9, but that is irrelevant.

$L(x)$ can have up to 6 elements, since $2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 < 50\,000$. However, the average length is 2.6 elements (determined experimentally). Therefore, if we look for a common element by merging two sorted lists, we expect to make just 5.2 comparisons on the average, and likely even fewer because we terminate the merge process as soon as we encounter a common element. There will be additional checks for corner cases (0 and 1).

This approach runs roughly twice as fast as the binary GCD algorithm and is expected to win 50 points.

Bit masks

We begin by observing that any number $x \leq 50\,000$ can have at most one prime factor greater than $\lfloor\sqrt{50\,000}\rfloor = 223$. Therefore, we can rewrite $L(x) = l(x) \cup \{p_x\}$, where $l(x)$ is the set of prime factors of x up to and including 223 and p_x is an optional prime factor larger than 223. If x does not have a prime factor larger than 223, then p_x is undefined. With this notation, two numbers x and y are coprime if

1. $l(x) \cap l(y) = \varnothing$ and
2. either p_x or p_y are undefined or they are different.

To quickly compute intersections of factor lists, note that 223 is the 48th prime. We can represent factor lists efficiently using one bit per prime number. Bit 0 (the least significant bit) denotes the presence of 2, bit 1 denotes the presence of 3 and so on. For example, $L(63) = \{3, 7\}$ is represented as the bit mask $000 \cdots 01010$, where bits 1 and 3 are set to denote the presence of the prime factors 3 and 7 respectively.

The big payoff of this representation is that we can test coprimality using very few operations:

1. The bitwise AND of $l(x)$ and $l(y)$. If it is nonzero, then x and y have a common divisor of at most 223.
2. The comparison of p_x of p_y, once we establish that both are defined.
3. A few additional corner cases for 0 and 1, depending on the implementation.

This representation needs just 10 bytes per coordinate (one `long long` for $l(x)$ plus one `unsigned short` for p_x), for a total of 500 KB. Grouping these in a struct will cause the `short` (16-bit) field to be aligned to 64 bits, which increases the actual memory usage to 800 KB. The small memory footprint is important because less memory is easier to cache, which can improve the performance considerably.

We can further reduce the memory needed to 400 KB by actually storing just 48 bits per bit mask. Since there is no native 48-bit data type, we can simulate it by splitting our masks into 32 bits (one `unsigned`) plus 16 bits (one `unsigned short`).

This approach runs more than twice as fast as the factor merging algorithm and is expected to win 100 points.