Investigating GCD In $\mathbb{Z}[\sqrt{2}]$

Rohil Prasad

January 11, 2014

Abstract

We attempt to optimize the time needed to calculate greatest common divisors in the Euclidean domain $\mathbb{Z}[\sqrt{2}]$.

1 Introduction

The greatest common divisor (GCD) of two integers a and b is the largest integer that divides both a and b. Finding the GCD of two integers is ubiquitous in many important number-theoretical algorithms, including the AKS primality test and the RSA encryption algorithm. We introduce some of the many fast solutions to this problem below.

1.1 Euclidean Algorithm

The Euclidean algorithm runs in $O(n^2)$ time, where n is the maximum bitsize of the inputs.

Given two integers a and b with $a < b$, each step of the Euclidean algorithm replaces the ordered pair (a, b) with a new pair (b', a). The number b' is the remainder upon division of b by a, or $b - qa$ where $q = \lfloor \frac{b}{a} \rfloor$. From the second equality we can see that, since $\gcd(a, b)$ divides a and b, it must also divide b'. Therefore, we reduce the problem of finding the GCD of a and b to finding the GCD of the smaller a and b'. We iterate the process until the numbers become so small that the problem is trivial, i.e. the ordered pair is $(u, 0)$ for some positive integer u. Then, since $\gcd(u, 0) = u$, the GCD is u itself.

1.2 Binary Algorithm

The binary algorithm also runs in $O(n^2)$ time.

Starting with our initial inputs a and b, we consider each modulo 2. If a is odd and b is even or vice versa, then we divide out by a factor of 2 since $\gcd(a, b)$ is equal to $\gcd(a, b/2)$ or $\gcd(a/2, b)$, respectively. If both are even, we find $\gcd(a, b) = 2\gcd(a/2, b/2)$, so we replace (a, b) with $(a/2, b/2)$ and store the factor of 2 elsewhere. If both are odd, then assuming $b > a$, we replace (a, b) with $(a, b-a)$. The algorithm terminates when either of the elements of the pair is equal to 0, and the larger element is found to be the GCD.

Despite the similar theoretical runtime to the Euclidean algorithm, the binary algorithm is about 15 percent faster in a practical setting [2] since division by 2 can be implemented quickly by a binary right-shift.

1.3 Subquadratic Algorithms

The first example of a subquadratic GCD algorithm was due to Schonhage [1] and ran in $O(n(\log n)^2 \log\log n)$ time. In general, these algorithms are prohibitively slow for any inputs that are not tens of thousands of bits in length.

2 Extension to Euclidean Domains

The Euclidean algorithm for the integers makes use of the property that, when dividing one integer by another, there is always a 'quotient' and a 'remainder'. Euclidean domains are integral domains that have similar properties. A *Euclidean domain* is an integral domain R equipped with a function N called a *norm* that maps elements of R to the natural numbers. Given nonzero elements a and b in R, there exist elements q and r such that $a = bq + r$ and $r = 0$ or $N(r) < N(b)$. The element q is called the *quotient* of a and b, while the element r is called the *remainder*. Note that $\mathbb{Z}$ is a Euclidean domain, with its norm mapping each integer to its absolute value.

The concept of GCD can also be easily generalized to commutative rings, of which Euclidean domains are a subset. Given a commutative ring R and elements a and b in R, an element g is the *greatest common divisor* of a and b if g divides both a and b and any other element dividing both a and b also divides g.

It is important to note that in the general case, a pair of elements can have more than one GCD. This is due to the existence of *units*, elements of R that have multiplicative inverses. If g is a GCD of elements a and b, and u is a unit, then it is easy to verify that gu is also a GCD of a and b. Therefore, our algorithms only attempt to find the GCD that is unique up to multiplication by units.

3 Description of Algorithms

We test three approaches for calculating the GCD of elements in the integer ring $\mathbb{Z}[\sqrt{2}]$. The ring $\mathbb{Z}[\sqrt{2}]$ is the set of all numbers of the form $x + y\sqrt{2}$, where x and y are integers.

The first approach is using the Euclidean algorithm. The ring $\mathbb{Z}[\sqrt{2}]$ is a Euclidean domain with norm $N(x + y\sqrt{2}) = |x^2 - 2y^2|$. The algorithm is essentially identical to the integer Euclidean algorithm.

The second approach is division by $2 + \sqrt{2}$. This works similarly to the binary algorithm for integers. We begin with our two elements a and b in $\mathbb{Z}[\sqrt{2}]$. There are four possible cases: both are divisible by $2+\sqrt{2}$, a is divisible by $2+\sqrt{2}$, b is divisible by $2 + \sqrt{2}$, and neither are divisible by $2 + \sqrt{2}$. In the first case, we can divide both by $2 + \sqrt{2}$ since $\gcd(a, b) = (2 + \sqrt{2}) \gcd(\frac{a}{2+\sqrt{2}}, \frac{b}{2+\sqrt{2}})$. In the second and third cases, we can divide the respective element by $2 + \sqrt{2}$ and the GCD remains the same. In the final case, we replace the element with larger norm with the difference of the two elements. The process is re-iterated until one of the elements is zero, and the nonzero element is the GCD.

The third approach is approximate division. The slowest step of the Euclidean algorithm is determining the quotient of the two elements, which requires division and multiplication of large numbers. However, an approximate quotient can be obtained by replacing the original elements with much smaller numbers and performing the multiplication/divisions steps with these replacements. This is done by bitshifting the elements to the right by a fixed amount. An approximate division algorithm that reported significant improvements over other algorithms for finding the GCD of Gaussian integers was published in 2002 by [3]. Our algorithm for $\mathbb{Z}[\sqrt{2}]$ calculates the approximate quotient q of elements a and b by bitshifting each of the components of a and b by about half their bitsize.

4 Results

We assessed the performance of each algorithm with the Python time library. We took the average runtime (in seconds) of the algorithm over 100 trials with randomly generated inputs of bitsize less than a fixed ceiling k. The results are tabulated below for k equal to 100, 200, 300, 400, and 500.

	Euclidean	Prime	Approximate
100	1.45	2.70	1.46
200	2.88	5.37	2.90
300	4.36	8.62	4.78
400	6.48	12.57	6.64
500	8.21	15.96	8.65

In their present implementations, the Euclidean algorithm is the fastest, but it is closely followed by the approximation algorithm.

5 Acknowledgements

The author would like to thank Tanya Khovanova and Jesse Geneson for their mentorship and guidance on the project. He would also like to thank Stefan Wehmeier and Ben Hinkle of MathWorks for proposing the project.

References

[1] N. Moller, On Schonhage's Algorithm and Subquadratic Integer GCD Computation, Mathematics of Computation Vol. 77, No. 261 (Jan., 2008), pp. 589-607.

[2] D. Knuth, Seminumerical Algorithms, The Art of Computer Programming 2 (3rd ed.), Addison-Wesley, ISBN 0-201-89684-2

[3] G.E. Collins, A Fast Euclidean Algorithm for Gaussian Integers, J. Symbolic Computation (2002) 33, 385392, doi:10.1006/jsco.2001.0518