Euclid's Algorithm

Sudeshna Chakraborty, Mark Heller, Alex Phipps.
Faculty Advisor: Dr. Ivan Soprunov, Department of Mathematics, Cleveland State University

Introduction

The Euclidean algorithm, also known as Euclid's algorithm, is an algorithm for finding the greatest common divisor (GCD) between two numbers. The GCD is the largest number that divides two numbers without a remainder. The GCD of two numbers can be found by making a list of factors for the two numbers, and finding the largest factor that is in both sets. This works well for small numbers, but it can become quite tedious and time consuming for larger numbers. To address this problem, Euclid's algorithm can be used, which allows for the GCD of large numbers to be found much faster. Euclid's algorithm uses the principle that the GCD of a set of two numbers does not change if you replace the larger of the two with the remainder when you divide the larger of the two by the smaller.

Procedure

The algorithm takes two numbers and finds the GCD between them. It does this in a recursive fashion by replacing the larger of the 2 numbers with the remainder of dividing those two numbers. This continues until the remainder is found to be 0. This process is visually demonstrated in figure 1 below.

b	=	(b / a)	*	a	+	(b % a)
148	=	5	*	26	+	18
26	=	1	*	18	+	8
18	=	2	*	8	+	2
8	=	4	*	2	+	0

Figure 1 demonstrates Euclid's algorithm. The first row shows the equation used to find the GCD.

Proof

Before we can prove Euclid's algorithm, we must prove that **GCD(a, b) = GCD(a, r)**. Where **a** and **b** are integers, and **q** and **r** are integers such that **b = q × a + r**.

We know that:

GCD(a, b) | a and **GCD(a, b) | b**

From the Division Theorem we know:

GCD(a, b) | (1 × b – q × a) and r = (b – q × a)

We can then replace the right side with **r:**

GCD(a, b) | r

From here we can start on the other side of that coin:

GCD(r, a) | r and **GCD(r, a) | a**

Using the Division Theorem again:

GCD(r, a) | (1 × r + q × a) and **b = q × a + r**

We can then replace the right side with a:

GCD(r, a) | b

This shows that a and b, and b and r have the same set of common divisors. Since the set of divisors is the same it follows that the GCD's are the same.

GCD(a, b) = GCD(r, a)

This proof is the basis for Euclid's algorithm. We can continually replace the larger of the pair with the remainder of the division of the two. This is demonstrated again in figure 2.

Proof continued

1	GCD(a, b)	$b \geq a > 0$
2	$b = q_1 \times a + r_1$	$a > r_1 \geq 0$
3	$a = q_2 \times r_1 + r_2$	$r_1 > r_2 \geq 0$
4	$r_1 = q_3 \times r_2 + r_3$	$r_2 > r_3 \geq 0$
	...	
n	$r_{n-1} = q_{n+1} \times r_n + 0$	

Figure 2: Illustration of proof, the number in the red circle will be the GCD.

Runtime

Runtime is an important characteristic of any algorithm. The Euclidean algorithm has an upper bound on the number of steps it will take to find the GCD. This bound is found by the equation:

$$k \leq \log_2(a) + \log_2(b)$$

where **k** is the number of steps. This inequality can be easily proven, if we assume **a** and **b** are positive integers and **(a < b)**, we can replace **b** with **r** as proven earlier so the inequality changes to **(a > r)**. Since **a** is smaller then **b** and **r** is smaller then **a**, we can conclude that:

$$(2 \times r < a + r \leq b)$$

We then come to the conclusion that:

$$(a \times r) < (.5 \times a \times b)$$

With every step of Euclid's algorithm the product of the current **a** and **b** goes down by a factor of 2. So:

$$a \times b \geq 2^k$$

Cleaning this up we get back to where we started:

$$k \leq \log_2(a) + \log_2(b)$$

This gives a fairly decent estimate on the upper bound of steps that it could take to perform the algorithm, but with the inequality it will only be a rough estimate.

Best case

The best case scenario for this algorithm would be the fastest it could possibly find the GCD. The fastest possible would be in one or two steps. This can happen in two different scenarios. The first being either **a** or **b** is **0**. If a or b is zero the algorithm will end and return the other value to you as the GCD. The other scenario is it being two steps with non zero values for **a** and **b**. There are several easy scenarios for this such as relatively small numbers, but what about very large numbers? The Euclidean algorithm can end in one step even if the numbers themselves are arbitrarily large. For example we could choose the pair below:

GCD(2,000,000 , 2,000,001)

Running through the algorithm:

1: 2,000,001 = 1 × 2,000,000 + 1;
2: 2,000,000 = 2,000,000 × 1 + 0;
1 is the GCD of 2,000,000 and 2,000,001.

This example shows that the size of the numbers is not as important as the distance between them.

Worst Case

Earlier we determined that the distance between the numbers was an important factor in determining the runtime, so this leads to the question what is the worst possible distance? Consecutive Fibonacci numbers give the worst possible run time. Fibonacci numbers are recursive in nature adding the two previous numbers to get the next number. This leads to each **q** value (the same q value shown in figure 2) being 1 and the remainder when they are divided being the previous Fibonacci number. The example below demonstrates this on two small Fibonacci numbers **8** and **5**:

1: **8 = 1 × 5 + 3;**
2: 5 = 1 × 3 + 2;
3: 3 = 1 × 2 + 1;
4: 2 = 1 × 1 + 1;
5: 1 = 1 × 1 + 0;
1 is the GCD of 8 and 5.

Even though the numbers are small it took 5 steps to find the GCD using the algorithm. The algorithm goes through all the Fibonacci numbers until it reaches 0.

Experiment

Using java we generated **200** random number pairs between **0** and **2,147,483,647** (the maximum integer value in Java) and found the GCD between them. We tracked the number of steps it took to find the GCD and plotted them in a histogram. The data make a bell curve around an average of 18 steps, shown below. The program used the built in java method Random which can generate random **int** numbers between a given upper bound and **0**. Even for randomly generated arbitrarily large numbers the algorithm can finish reasonably quickly.

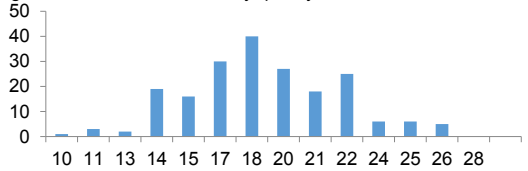

Figure 3:Histogram of the frequency (y-axis) and the number of steps to complete the algorithm (a-axis) on randomly generated pairs in Java.

Conclusion

The Euclidean algorithm is a very fast way of finding the GCD of two numbers no matter how large those numbers are. With good estimates for the maximum number of steps and known worst case scenarios, the algorithm is efficient and predictable.

References

Lovász, L., Pelikán, J., & Vesztergombi, K. (2003). *Discrete mathematics elementary and beyond* (pp. 99-104). New York: Springer.

Acknowledgments

Special thanks to Dr. Soprunov for his guidance and support throughout this project, and a special thanks to the COF and NSF programs for the funding.