Practice 12

Mayra Cristina Berrones Reyes

April 17, 2020

1 Introduction

The greatest common divisior (GCD) of two numbers is the largest number that can divide them both without residual numbers. This concept can easily extend to a bigger set of numbers, because the GDC is the largest number dividing each of them.

The GDC is used for a variety of applications in number theory such as modular arithmetic, but it can also be used in simpler applications, such as simplifying fractions. The GDC is traditionally notated as $gdc(a, b)$ [Alexander Katz, 2020]. The problem to solve is, given two non-negative integers a and b, we need to find their GCD, which is the largest number that can divide both a and b. Mathematically we can define it as Equation 1.

$$gcd(a, b) = \ max\, k. \tag{1}$$

Where:

$k = 1 \ldots \infty : k|a \wedge k|b$

(In several examples found, they use the "|" symbol as divisibility. Meaning that $k|a$ is "k" divides "a")

In this case, if one of the numbers a or b is zero, while the other is not zero, then the GCD by definition, is the non-zero number. For example $gcd(a, 0) = a$. When both numbers are zero, then the GCD remains undefined, as it can be any arbitrarily large number [Kogler, 2014].

There are several ways to solve the GCD. One of them is by simply listing the factors of each number and determining the largest common one. For example, in Figure 1 we have the number 30 and 24, then we have a list with all of the factors. As we can see, the largest common factor is the number 6, so that is our GCD. This practice however is very inefficient, but for small cases it can be done by hand. There are other methods to solve this, such as the euclidean

algorithm [Alexander Katz, 2020].

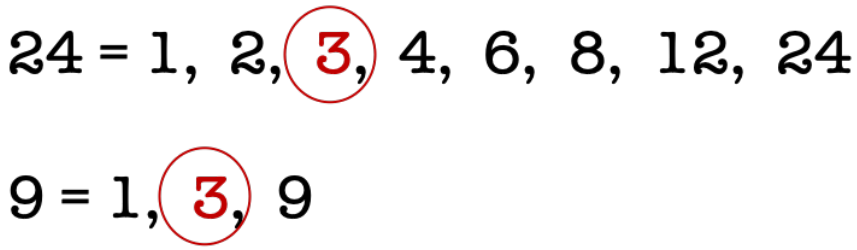

Figure 1: Example of listing the factors of the numbers to find de GCD.

2 Euclidean algorithm

The euclidean algorithm is one of the oldest numerical algorithms that is used commonly to this day. The exact origin of the algorithm is unknown, but it was first described in Euclid's *"Elements"* in 300 B.C. and it solves the problem of computing the GCD of two positive integers.

2.1 GCD by subtraction

The original version of the euclidian algorithm is based on subtraction. We recursively subtract the smaller number from the bigger one [Codility, 2014].

```
def gcd(a, b):
  if a == b:
    return a
  if a > b:
    gcd(a - b, b)
  else:
    gcd(a, b -a)
```

Above we can see the code to solve this problem. In Figure 2 is an example of how this algorithm work, choosing the bigger number, subtracting the smaller one until they both are the same. The worst case complexity for this problem can be linear $\mathcal{O}(n)$, because the value of the integers decreases with every step.

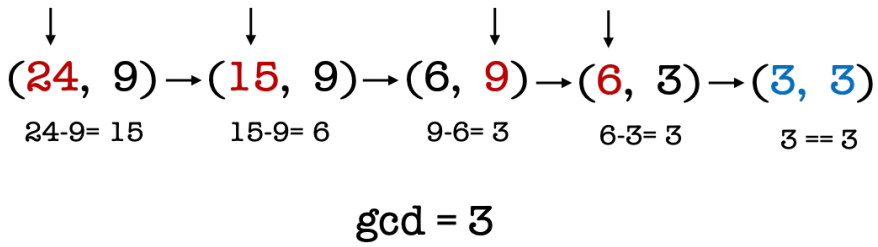

Figure 2: Example of the euclidian algorithm by sustraction.

2.2 GCD by dividing

This is the most common implementation for the euclidean algorithm for GCD. The algorithm for this can be seen in Equation 2.

$$gcd(a,b) = \begin{cases} a & \text{if } b = 0 \\ gcd(b, a \bmod b), & \text{otherwise} \end{cases} \tag{2}$$

This equation is for two given numbers a and b, such that $a \geq b$. The *mod* operation in Equation 2 is the residual of $a \,|\, b$. In Figure 3 we can see how this equation works, and below the code for it.

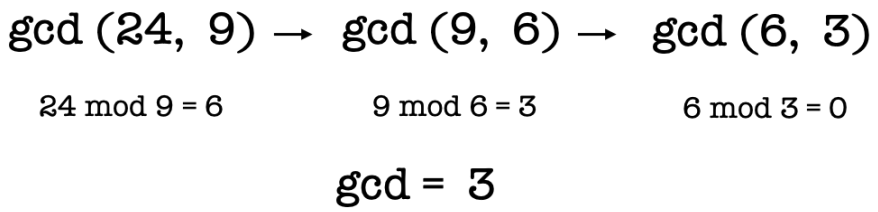

Figure 3: Example of the euclidian algorithm for GCD by dividing.

```
def gcd(a, b):
  if a % b == 0:
    return b
  else:
    return gcd(b, a % b)
```

Now, we can see in each iteration the argument decreases, which means that the algorithm will always terminate, because our elements are non-negative.

For proof of correctness, we already stablished the first part of Equation 2 that if the other number is 0, then the GCD is the non-zero number. So now we need to prove that $gcd(a,b) = gcd(b, a \bmod b)$ for all $a \geq 0$, and $b > 0$.

Let d be the GCD of (a, b). Then by definition $d \mid a$ and $d \mid b$, which means that both a and b are divisible by d.

$$a \, mod \, b = a - b \cdot \left\lfloor \frac{a}{b} \right\rfloor \tag{3}$$

In Equation 3 we represent how the *mod* operation works. From this follows that $d \mid (a \, mod \, b)$ [Trevor, 2015].

How do we know the algorithm works is, if $a = bq + r$ a and b being our integers to find the GCD and q and r some other integers, then we have that $gcd(a, b) = gcd(b, r)$. In Figure 4 we have an example of how this works. First we have the a number represented by 2322. Then the b number which is 654. The q number represents the integer which will multiply the b number, and finally the r represents the residue necessary to get number a. After all recursions we get the GCD as 6.

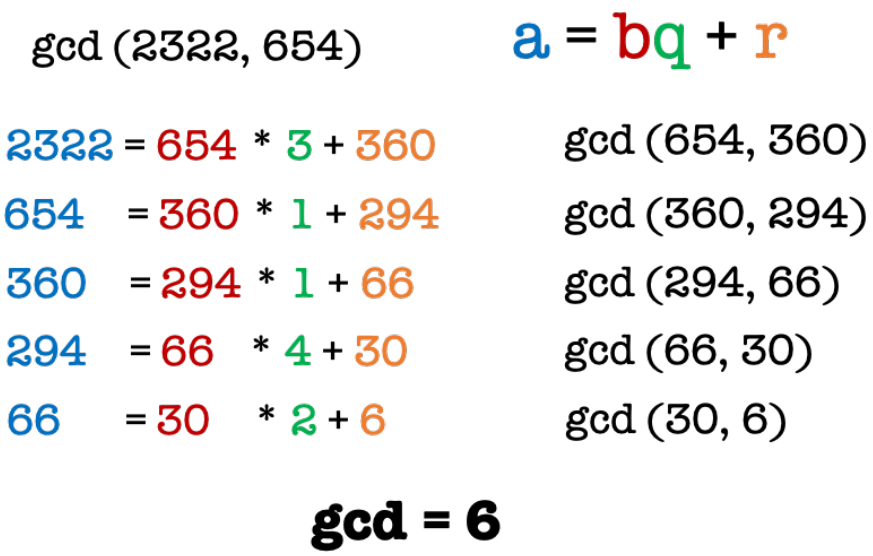

Figure 4: Example of how to get the GCD by dividing.

Now, the only way to prove the algorithm is if $gcd(a, b) = d$ and $gcd(b, r) = d$. We already proved that $d \mid a$ and $d \mid b$, and in Equation 3 we can translate that to $d \mid (a - qb)$. With this, we can use the equation in Figure 4 of $a = bq + r$ we can clear r and we have then $r = a - bq$, which translates as $d \mid r$. So, as shown in Figure 5 we prove both sides and we can conclude that $gcd(a, b) = d$

and $gcd(b, r) = d$.

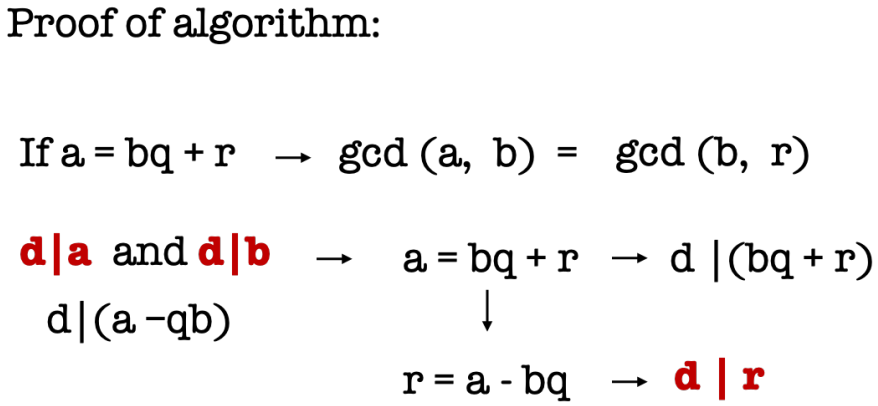

Figure 5: Proof of both sides of the GCD.

For the time complexity, we can estimate the number of recursions this problem may have with the Lame theorem, that establishes a connection between the euclidian algorithm and the Fibonacci sequence. If $a > b \geq 1$ and $b < F_n$ for some n, the euclidean algorithm performs at most $n - 2$ recursive calls. The consecutive Fibonacci numbers are the worst case input for the euclidian algorithm, and given that Fibonacci numbers grow exponentially, we get that this algorithms works in $\mathcal{O}(\log\, min\,(a, b))$.

3 Conclusions

I tried to make this practice more informative for me, than to make it similar to the example on the book, because I struggle a little bit to understand the algebraic format (too many letters and numbers). The part of the Fibonacci sequence to see the time complexity for the algorithm was not completely clear for me, because the Lame theorem is very old, and I got confused with all the different interpretations of it. At the end the only thing clear it was that it was the worst case for the euclidian algorithm because it grows exponentially.

References

[Alexander Katz, 2020] Alexander Katz, Patrick Corn, M. A. (2020). Greatest common divisor. Disponible en: "https://brilliant.org/wiki/greatest-common-divisor/".

[Codility, 2014] Codility (2014). Euclidean algorithm. Disponible en: "https://codility.com/media/train/10-Gcd.pdf".

[Kogler, 2014] Kogler, J. (2014). Euclidean algorithm for computing the greatest common divisor. Disponible en: "https://cp-algorithms.com/algebra/euclid-algorithm.html".

[Trevor, 2015] Trevor (2015). Euclidean algorithm. Disponible en: "https://www.youtube.com/watch?v=cOwyHTiW4KE".