12

The Euclidean Algorithm

Friedrich Eisenbrand

EPFL, Lausanne, Switzerland

This chapter deals with one of the oldest algorithms that appears in records from the ancient world. The algorithm is described in *The Elements*, the famous book by Euclid, which was written roughly 300 BC. Nowadays, this algorithm is a cornerstone in many areas of computer science, especially in the area of cryptography, see Chap. 16, where many fundamental routines rely on the fact that the greatest common divisor of two numbers can be efficiently computed.

Imagine that you have two bars of length a and b, respectively, where both a and b are integers. You want to cut both bars into pieces, each having the same length. Your goal is to cut the bars in such a way that the common length of the pieces is as large as possible. We could, for example, cut the bar of length a into a many pieces of length 1 and the bar of length b into b many pieces of length 1. Is a larger common length of the pieces possible?

Our algorithm computes the largest possible common length of the pieces. We describe two versions of the algorithm. The first version is slow, or *inefficient*. The second version is fast, or *efficient*.

Let d denote the largest common length of the pieces that can be possibly achieved. The bar of length a and the bar of length b are cut into a/d and b/d

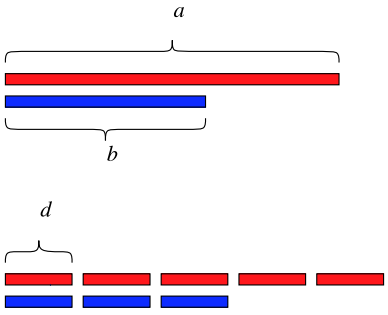

Fig. 12.1. Cutting two bars into pieces of common length d

B. Vöcking et al. (eds.), *Algorithms Unplugged*,
DOI 10.1007/978-3-642-15328-0_12,

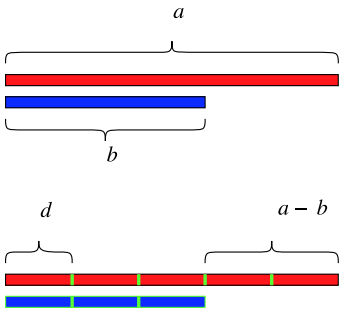

Fig. 12.2. The common length of pieces that we search for a and b is the common length of pieces for $a - b$ and b

many pieces, respectively. The picture above displays a situation where a is cut into 5 and b is cut into 3 pieces. How can we find the largest d?

If both bars have equal length, i.e., if $a = b$, then the value of d is immediately clear. We do not have to cut the bars at all. The largest length d such that we can cut a and b into d-sized pieces is the common length of the bars itself. Let us therefore assume that the length of the two bars is different, where we assume that a is larger than b. As you lay both bars next to each other, you make an important observation, see the figure above. If we can cut both bars into pieces of length d, then we can cut off a piece of length b from the larger bar.

The resulting bar has length $a - b$ and can be cut into pieces of length d as well. Conversely, if we can cut both bars, the one of length $a - b$ and the one of length b, into pieces of length d, then we can cut also a into equal pieces of length d.

We formulate this insight separately. It is the main principle underlying our algorithm.

Principle (P)
If $a = b$, then the length d we are looking for is a. If a is larger than b, then the common length of pieces for a and b is the common length of pieces for $a - b$ and b.

We can now formulate an algorithm that computes the largest length d of pieces into which a and b can be cut.

Largest common length of pieces

While both bars do not have equal length:
  Cut off from the larger bar a piece being as long as the smaller bar and put this piece aside.
Now both bars have equal length. This common length is the length d we are looking for.

At this point we must ask ourselves whether the above algorithm ever finishes or, in computer science terminology, *terminates*. We can observe that it indeed does. Remember that the lengths of the bars in the beginning are integers a and b, respectively. The lengths of the bars remain integers as we cut off a piece from the longer bar that is as long as the shorter bar. In particular, the length of both bars is at least 1. As we cut one bar, we remove at least a piece of length 1. Thus the algorithm performs at most $a + b$ rounds.

The Greatest Common Divisor

Our analysis from above reveals that the length d that we are computing is also an integer. It is an integer which *divides* both a and b. In mathematical terminology this means that there are integers x and y such that $a = x \cdot d$ and $b = y \cdot d$, respectively. The number d is the largest number which has this property that there exist integers x and y as above. The number d is the *greatest common divisor* of a and b. The integers x and y are the number of pieces of length d into which the bars of length a and b are cut, respectively.

We can also describe our algorithm in more abstract terminology, where we no longer use bars. The inputs to our algorithm are two positive inte-

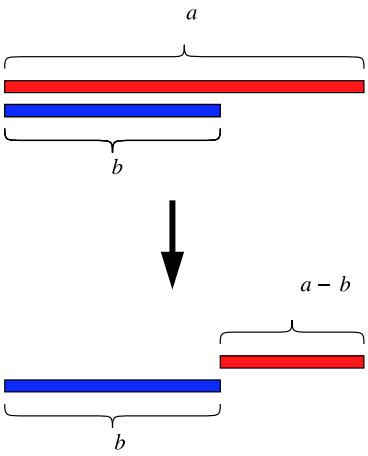

Fig. 12.3. One step of the algorithm

gers a and b. The output of our algorithm is the greatest common divisor of a and b. We call the algorithm *SlowEuclid* for a reason that is soon going to be illuminated.

SlowEuclid

While $a \neq b$
  If a is larger than b, then replace a by $a - b$
  If b is larger than a, then replace b by $b - a$
Return the common value of both numbers.

Let us consider a small example.

The input in this example is 15 and 9. In the first step, we subtract 9 from 15 and obtain the new pair of numbers $6 = 15 - 9$ and 9. In the second step, we obtain 6 and 3. In the third step, we obtain 3 and 3 and the algorithm returns the number 3.

The next example explains why we called the algorithm *SlowEuclid*. Consider the input $a = 1001$ and $b = 2$. The two numbers during the execution of the loop of the algorithm are

1001 and 2
999 and 2
997 and 2
995 and 2
... (many rounds in-between)
3 and 2
1 and 2
1 and 1

The reason for the algorithm to take such a long time is the fact that the second number is excessively smaller than the first number of the input.

An Observation That Speeds up the Algorithm

In the example above, how often is 2 subtracted from 1001? One has $1001 = 2 \cdot 500 + 1$. Thus the number 2 is subtracted 500 times from 1001 until the value of the outcome drops below 2.

A computer can very efficiently perform a *division with remainder*. This operation computes for positive integers a and b two other integers q and r with $a = q \cdot b + r$. The integer r is at least zero and strictly smaller than b. In our example we have $a = 1001$, $b = 2$, $q = 500$ and $r = 1$.

If a and b is the input to our algorithm *SlowEuclid*, where a is larger than b, then b is repeatedly subtracted from a q times, if there is a remainder $r \geq 1$.

If b divides a exactly and $r = 0$, then b is subtracted from a $q - 1$ times and two bars of equal length are the outcome. This means that we can speed up the algorithm by immediately replacing a by the remainder r of this division. It then eventually happens that the remainder r is zero, in which case b is the greatest common divisor we are looking for and the algorithm terminates.

This is the idea of the next algorithm which we now call EUCLID.

EUCLID

```
1  if a < b: swap a and b.
2  while b > 0:
3      compute integers q, r with a = q · b + r, where 0 ≤ r < b;
4      a := b; b := r;
5  return a.
```

Analysis

You probably guess that the algorithm EUCLID is much faster that SLOWEUCLID. Let us now rigorously analyze the number of iterations that the algorithm performs to substantiate this suspicion. Suppose that a is larger than b. How large then is the number r with which we replace a in the first step of the algorithm? The next picture reveals that this remainder is always at most $a/2$. This is because a is at least $b + r$ and since $b > r$ it follows that a is larger than $2 \cdot r$.

Thus in the first round, b is replaced by a number which is at most $a/2$. In the next round, a is replaced by a number which is at most $a/2$ too. Thus after two rounds, both numbers are at most $a/2$.

If we now consider $2 \cdot k$ consecutive rounds, then both numbers have a value that is at most $a/2^k$. If $k > \log_2 a$, then both numbers would have value zero. This, however, cannot happen since then the algorithm would already have finished before. Therefore the number of iterations through the loop of the algorithm is bounded from above by $2 \cdot \log_2 a$, where we again use the logarithm to the base 2.

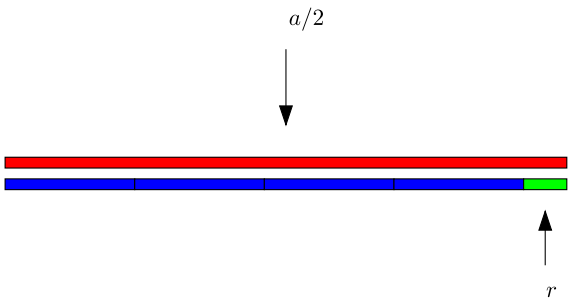

Fig. 12.4. The remainder r is small

The number of *digits* which is required in our decimal system to write down the number a is proportional to $\log_2 a$. This means that, while the algorithm SlowEuclid requires a number of iterations that is proportional to the *values* of a and b, the algorithm Euclid requires only a number of iterations that is proportional to the *number of digits* that we need to write down a and b, respectively. This is an enormous difference in running time.

An Example

Finally we compute by hand the greatest common divisor of $a = 1324$ and $b = 145$.

The first division with remainder is $1324 = 9 \cdot 145 + 19$. Now a is set to 145 and b is set to 19.

The second division with remainder is $145 = 7 \cdot 19 + 12$. The third division with remainder yields $19 = 1 \cdot 12 + 7$. Then one has $12 = 7 + 5$, $7 = 5 + 2$, $5 = 2 \cdot 2 + 1$ and $2 = 2 \cdot 1 + 0$, from which we can conclude that the greatest common divisor of 1324 and 145 is 1. Two integers whose greatest common divisor is 1 are called *coprime*.

Further Reading

1. Donald E. Knuth: Arithmetic. Chapter 4 in *The Art of Computer Programming*, Vol. 2: *Seminumerical Algorithms*. Addison-Wesley, 3rd edition, 1998.
   This classical textbook treats the Euclidean algorithm in Chap. 4.5.2. Among other things, the author explains that our analysis of the Euclidean algorithm is the best possible. More precisely it is shown that there exists a sequence $F_0, F_1, F_2, \ldots$ of natural numbers, for which the number of digits which are necessary to represent F_i is proportional to i while the Euclidean algorithm requires on input F_i and F_{i-1} exactly i iterations.
2. Joachim von zur Gathen, Jürgen Gerhard: *Modern Computer Algebra*. Cambridge University Press, 2nd edition, 2003.
   This very nice textbook for advanced students of Computer Science and Mathematics discusses in Chap. 6 the Euclidean algorithm. The book also discusses the number of *elementary operations* (see Chap. 11) which are required by the Euclidean algorithm and some of its variants. Here, and also in the book of Knuth, the authors describe algorithms to compute the greatest common divisor of two integers that require a number of elementary operations which is proportional to $M(n) \log n$. The number n is then the total number of digits of the input and $M(n)$ denotes the number of elementary operations that are necessary to compute the product of two integers having at most n digits each.

3. In Wikipedia:
http://en.wikipedia.org/wiki/Euclidean_algorithm

Acknowledgement

The author is grateful to M. Dietzfelbinger for many helpful comments and suggestions.