Euclidean Algorithm

GCD stands for “greatest common divisor”.

E.g. \( \text{GCD}(10, 25) = 5 \). If \( \text{GCD}(A,B)=1 \), we say \( A \) and \( B \) are relatively prime.

Note that \( \text{GCD}(N, 0) = N \). (At least for \( N>0 \).)

Thm: If \( A + kB = C \) and \( B \neq 0 \), then \( \text{GCD}(A,B) = \text{GCD}(B,C) \).

Proof. Let \( g = \text{GCD}(A,B) \) and \( h = \text{GCD}(B,C) \).

Since \( g \) divides \( A \) and \( B \), it must divide \( A + kB = C \).

Since \( g \) is a common divisor of \( B \) and \( C \), it must be \( \leq \) their greatest common divisor, i.e. \( g \leq h \).

Reversing the roles of \( A \) and \( C \), we conclude that \( h \leq g \).

Thus, \( g = h \). QED

The Euclidean Algorithm finds \( \text{GCD}(A,B) \).

Given \( A > B > 0 \), set \( C = A \mod B \) (so \( C = A - kB \) for some \( k \) and \( C < B \)).

This reduces the problem of finding \( \text{GCD}(A,B) \) to the smaller problem, finding \( \text{GCD}(B,C) \). Eventually, we’ll get to \( \text{GCD}(X,0) = X \).
Euclidean Algorithm

Example: Find GCD of 38 and 10.
38 mod 10 = 8
10 mod 8 = 2
8 mod 2 = 0
GCD(2,0) = 2

Extended Euclidean Algorithm

Example: Find GCD of 38 and 10.
38 mod 10 = 8  \quad \text{Lets you write 8 in terms of 38 and 10} \quad 8 = 38 - 3\times 10
10 mod 8 = 2  \quad \text{Lets you write 2 in terms of 10 and 8.} \quad 2 = 10 - 1\times 8
8 mod 2 = 0  \quad \text{Change 8's to 10's and 38's} \quad = 10 - 1\times(38-3\times10)
GCD(2,0) = 2  \quad \text{Voila!} \quad 2 = 4\times10 - 1\times38

Can write “2” as linear combination of 10 and 38.
Euclidean Magic

• Solve Diophantine Equations:
  - E.g. “a \times 38 + b \times 10 = 6”
    • First write \text{GCD}(38,10) using 10 & 38 (2 = 4 \times 10 - 1 \times 38),
      then multiply by 6/\text{GCD} = 3
      \hspace{1cm} (6 = 12 \times 10 - 3 \times 38).

• Rational approximations:
  - E.g. 31416 = 3\times10000 + 1416
    \hspace{1cm} 10000 = 7\times1416 + 88
  - Let’s pretend 88 \approx 0. We then have 1416 \approx 10000/7
    \hspace{1cm} Thus, 31416 \approx 3\times10000 + 10000/7 = 10000\times(3+1/7)
    \hspace{1cm} Thus, 3.1416 \approx 3+1/7 = 22/7 \ (\approx 3.142857...)
    \hspace{1cm} We can get all “close” approximations this way.

Morals

• Once you have solved something (e.g. GCD),
  see if you can solve other problems (e.g. linear integer equations, approximations, ...)

• I’ve shown some key ideas without writing out the algorithms – they are actually simple.
  “Continued fractions” does this all nicely!

• If this is your idea of fun, try cryptography.
Divide & Conquer

• Basic Idea
  - Break big problem into subproblems
  - Solve each subproblem
    • Directly if it's very small
    • Otherwise recursively break it up, etc.
  - Combine these solutions to solve big problem

Faster Multiplication ??

Standard method for multiplying long numbers:

\[(1000a+b)\times(1000c+d) = 1,000,000 \, ac \]
\[+ 1000 \, (ad + bc) \]
\[+ bd \]

Clever trick:

\[(1000a+b)\times(1000c+d) = 1,000,000 \, ac \]
\[+ 1000 \, ((a+b)(c+d) - ac - bd) \]
\[+ bd \]

One length-\(k\) multiply = 3 length-\(k/2\) multiplies and a bunch of additions and shifting.

On computer, might use \(2^{16}\) in place of 1000

Fine print: If "a+b" has 17 bits, drop the top bit, but add extra c+d in the right place. Handle overflow for c+d similarly.
Faster Multiplication!!

To magnify savings, use recursion (Divide & Conquer).

Let $T(n) =$ time to multiply two $n$-chunk numbers.

For $n$ even, we have $T(n) < 3T(n/2) + c \cdot n$

This is called a recurrence equation

Make $c$ big enough so that $cn$ time lets you do all the adds, subtracts, shifts, handling carry bits, and dealing with +/-.

Reasonable, since operations are on $n$-chunk numbers.

Recursively divide until we have 1-chunk numbers

Need $\lg n$ “divide” steps.

Also make $c$ big enough so $T(1) < c$.

Recursion Tree

1 depth-0 node

3 depth-1 nodes

9 depth-2 nodes

$3^{\lg n}$ depth-$\lg n$ nodes
From the picture, we know ...

\[ T(n) < cn \left( 1 + 3(1/2) + 9(1/4) + \ldots + 3^{\lg n} \right) \]
\[ < cn \left( 1 + \frac{3}{2} + \left(\frac{3}{2}\right)^2 + \ldots + \left(\frac{3}{2}\right)^{\lg n} \right). \]

**Lemma 1:** For \( a \neq 1 \), \( a^k + a^{k-1} + \ldots + 1 = \frac{a^{k+1} - 1}{a-1} \)

**Proof:** \( (a^k + a^{k-1} + \ldots + 1)(a-1) = a^{k+1} - 1 \)

It's obvious! (Multiply it out – middle terms cancel.)

By lemma 1, \[ T(n) < cn \left( \frac{\left(3/2\right)^{\lg n + 1} - 1}{(3/2)-1} \right) \]
\[ < cn \left( \frac{(3/2)^{\lg n} (3/2) - 0}{1/2} \right) \]
\[ = cn \left( \frac{(3/2)^{\lg n} (3/2)}{1/2} \right) \]
\[ = 3cn \left( \frac{3}{2} \right)^{\lg n}. \]

And finally ...

**Lemma 2:** \( a^{\lg b} = b^{\lg a} \)

**Proof:** \( a^{\lg b} = (2^{\lg a})^{\lg b} = 2^{\lg a \cdot \lg b} = (2^{\lg b})^{\lg a} = b^{\lg a} \)

Applying Lemma 2, we have

\[ T(n) < 3cn \left( \frac{3}{2} \right)^{\lg n} = 3cn \left( n^{\lg(\frac{3}{2})} \right) = 3cn^{1+\lg(\frac{3}{2})}. \]

Thus, \( T(n) \in O(n^{1+\lg(\frac{3}{2})}) = O(n^{\lg 3}) = O(n^{1.58\ldots}). \)
Key points

• Divide & Conquer can magnify a small advantage.

• Recursion tree gives a method of solving recurrence equations.
  - We’ll look at other methods next.

• Lemmas 1 and 2 are useful.
  - Make sure you’re comfortable with the math.