Greatest Common Divisor

An example of a loop designed inductively, from a base case and progress case

Greatest Common Divisor

\[
\frac{42}{56} = \frac{3 \cdot 14}{4 \cdot 14} = \frac{3}{4}
\]

gcd(42, 56) = 14

GCD = factors in common

\[
\begin{align*}
42 &= 3 \times 2 \times 7 = 3 \times 14 \\
56 &= 7 \times 2 \times 2 \times 2 = 4 \times 14 
\end{align*}
\]

Useful facts for computing gcd(a, b)

\[
gcd(n, 0) = n \\
because n \times 0 = 0, \text{ for all } n
\]

\[
gcd(a, b) = gcd(b, a)
\]

\[
gcd(a - b, b) = gcd(a, b) \quad \text{if } a > b
\]

this will be the progress case for our loop
Recall the skeleton:

while (more work) {
    if (basis case) {
        just do it;
    } else {
        use progress case to make the problem a little smaller;
    }
}
Often becomes ...

while (not the basis case yet) {  
    use progress case to make  
    the problem a little smaller;
}  
apply the basis case  
to solve the problem in one step;

For GCD

Basis case: gcd(n, 0) or gcd(0, n)  
*They are the same since gcd(a, b) = gcd(b, a)*

Progress case: gcd(a, b) with a > b > 0  
*Again, we can reverse a and b if b > a*

Let’s fill it in ...

while ( ) {  // basis case?
    // make progress
}  // apply basis

Euclid’s algorithm, subtraction version

while (x > 0 && y > 0) {  // basis case?
    if (x > y) {  
        x = x - y;
    } else {  
        y = y - x;
    }
}  // Now the basis case must hold.
if (x == 0) { return y; }  
else { return x; }
How many steps can it take?

What’s the **worst possible** input for this method, in terms of steps (loop iterations) needed?

How bad is it?

*We describe the computational complexity of an algorithm in terms of the worst case, relative to problem size.*

---

Another fact about gcd(a,b)

\[ \gcd(a, b) = \gcd(b, a \mod b) \]

Now that’s progress!

(also note a % b < b, simplifying our logic)

---

Faster! Faster!

We could make minor improvements to the code ...

*Maybe we can cut the cost in half, or a quarter ... but that doesn’t help much.*

For real improvements, we need a better algorithm, based on a better progress case!

---

Basis, progress again

while ( b > 0 ) {
    // Progress changes gcd(a,b) to gcd(b, a%b)
}

// basis case
return a;
How can I swap a and b?

Can I write a “swap” method?

Maybe I don’t want to write out the whole code to swap a and b. I want a method
swap(int a, int b) { ... }

Can this work? Why or why not?

Oh well ...

Now how fast?

while (b > 0) {
   // (a, b) = (a mod b, a)
   int temp = a;
   b = a % b;
   a = temp;
}
return a;

How is the old worst case?
gcd(20025, 1) = gcd(1, 0) = 1
seems a little faster ...

What’s the worst case of this algorithm?
Do we always cut a-b at least in half? How many times can we cut x in half before it’s less than 1?
Induction and Complexity

If the progress case reduces problem size by 1, loop iterations will be proportional to problem size. We call it “linear” complexity.

- Similarly for bites of any constant size k: n/k is still proportional to n.

If the progress case divides the problem size in half, loop iterations will be logarithmic in problem size. Huge difference.

- Logarithmic growth is inverse of exponential explosion.

Summary

Greatest common divisor

- Used to reduce fractions to lowest terms
- Has nice identities, esp \( \text{gcd}(a,b) = \text{gcd}(b,a\%b) \)

Euclid’s algorithm

- A classic application of our inductive loop skeleton, invented 3000 years before computers.

Cost is characterized relative to problem size