Factoring, Euclidean algorithm and rhythms

Ben Bolker

10:26 06 February 2015

The Euclidean algorithm

definition and history

- oldest known algorithm (Euclid, c. 300 BCE) ("algorithm" from al-Khwrizm, c. 780-850 CE)
- find **greatest common divisor** of two integers: gcd(`a`,`b`) is the largest integer `c` such that `c` divides both `a` and `b` evenly.
- ex.: gcd(36,42)=6
- brute force: enumerate factors of `a` and `b`, multiply common factors ($36 = 2^2 \times 3^2$, $42 = 2 \times 3 \times 7$)

```
def find_factors(a):
    """find integer factors (other than 1 and a) by brute force"""
    factors = []
    i = 2
    while i<=a:
        ## print(i,a%i,a//i,factors)
        if a%i==0:
            factors += [i]
            a = a//i
        else:
            i += 1
    return(factors)

f1 = find_factors(36)
f2 = find_factors(42)
def common_factors(f1,f2):
    for i in f1:
        if not (i in f2):
            f1.remove(i)
    for i in f2:
        if not (i in f1):
            f2.remove(i)
    if (len(f1)<len(f2)):
        return(f1)
    else:
        return(f2)
print(common_factors(f1,f2))
```

```
## [2, 3]
```

- if the gcd is 1, the numbers are **co-prime** (but not necessarily both prime!)

Subtractive form

- subtract the smaller number from the larger; take the smaller and the remainder and repeat, until you get to zero
- $\{42, 36\} \to \{6, 0\}$
- $\{407, 361\} \to \{361, 46\} \to \{315, 46\} \to ... \to \{85, 46\} \to \{46, 39\} \to \{39, 7\} \to \{32, 7... \to \{7, 4\} \to \{4, 3\} \to \{3, 1\} \dots$ co-prime!
- intuition
- code:
- looping:

```
def euc_alg1(a,b):
    '''subtractive Euclid algorithm by looping'''
    while b>0:
        lg = max(a,b)
        small = min(a,b)
        b = lg-small
        a = small
        ## print(a,b)
    return(a)
```

- recursive:

```
def euc_alg2(a,b):
    '''subtractive Euclid algorithm by recursion'''
    if (b==a):
        return(a)
    return(euc_alg2(min(a,b),abs(a-b)))
```

division form

- intuition: there's no need to subtract repeatedly when we could do the same thing in one step by dividing and taking the remainder.
- example:
  - $\{42, 36\} \to \{6, 0\}$ (one step)
  - $\{407, 361\} \to \{361, 46\} \to \{46, 39\} \to \{39, 7\} \to \{7, 4\} \to \{4, 3\} \to \{3, 1\} \dots$ co-prime!

- looping

```
def euc_alg3(a,b):
    '''modular Euclid algorithm by looping'''
    while b>0:
        lg = max(a,b)
        small = min(a,b)
        b = lg % small
        a = small
        print(a,b)
    return(a)
```

- recursive

```
def euc_alg4(a,b):
    '''subtractive Euclid algorithm by recursion'''
    if (b==a):
        return(a)
    m = min(a,b)
    return(euc_alg2(m,max(a,b) % m))
```

Computational complexity and timing

- count operations
- want to know how complexity **scales** with problem size
- e.g. $O(n)$, $O(\log n)$, $O(\exp(n))$ (uh-oh!)
- polynomial-time vs. non-polynomial
- best-case vs average vs worst-case

Bjorklund algorithm

Worked example

References

- old paper on EA applications (in math): http://www.jstor.org/stable/3029367
- http://www.hisschemoller.com/blog/2011/euclidean-rhythms/
- http://cgm.cs.mcgill.ca/~godfried/publications/banff-extended.pdf
- Python implementations: (watch out for Python 2 constructions! need to replace `/` by `//` for integer division; replace `xrange()` with `range()` ...
  - http://www.atonalmicroshores.com/2014/03/bjorklund-py/