

Number theory with Clojure

Valery Kocubinsky

March 20, 2024

Contents

1	About	2
2	Notation	2
3	Some basic functions	3
3.1	Power function	3
3.2	Sign function	3
3.3	Order function	3
3.4	The greatest common divisor	3
4	Performance and cache	4
5	Primes	5
6	Integer factorization	5
7	Divisors	7
8	Arithmetical functions	7
8.1	Function equality	7
8.2	Additive functions	7
8.3	Multiplicative functions	8
8.4	Higher order function for define multiplicative and additive functions	8
8.5	Some additive functions	9
8.5.1	Count of distinct primes - ω	9
8.5.2	Total count of primes - Ω	9
8.6	Some multiplicative functions	9
8.6.1	Mobius function - μ	9

8.6.2	Euler totient function - ϕ	10
8.6.3	Unit function - ϵ	10
8.6.4	Constant one function - 1	10
8.6.5	Divisors count - σ_0	11
8.6.6	Divisors sum - σ_1	11
8.6.7	Divisors square sum	11
8.6.8	Divisors higher order function - σ_x	12
8.6.9	Liouville - λ	12
8.7	Some other arithmetic functions	12
8.7.1	Mangoldt - Λ	12
8.7.2	Chebyshev functions θ and ψ	13
8.8	Dirichlet convolution	13

1 About

This project cover some topics in number theory, especially arithmetic functions, further more multiplicative functions. There are set of well known arithmetic functions and one can define custom arithmetic functions.

I wrote this document `readme.org` with Emacs Org Mode. Then I generate markdown file `readme.md` with Org Mode export to markdown `C-c C-e m m`, and generate pdf file `readme.odf` with Org Mode export to pdf `C-c C-e l p`. Github by default show `readme.md` if a project has such file. It looks enough good, even math equation is supported. But I see some issue with greek characters in table of content. If it is a problem `readme.pdf` looks better. Emacs file "make.el" can be used to export markdown and pdf with one call. I use Emacs babel for clojure to produce real output inside the document.

In this document I load number theory packages as:

```
(require '[vk.ntheory.basic :as b])
(require '[vk.ntheory.primes :as p])
(require '[vk.ntheory.ar-func :as f])
(require '[vk.ntheory.congruence :as c])
(require '[clojure.math :as math])
```

So below I will use above aliases.

2 Notation

- **N** - Natural numbers, positive integers 1, 2, 3, ...

- **C** - Complex numbers
- **Z** - Integers $\cdots - 3, -2, -1, 0, 1, 2, 3, \dots$

3 Some basic functions

3.1 Power function

Clojure has built-in `clojure.math/pow` function, but it return `java.lang.Double`. The library provide integer analog

```
(b/pow 2 3)
```

8

3.2 Sign function

```
(mapv b/sign [(- 5) 10 0])
```

```
[-1 1 0]
```

3.3 Order function

Order function $ord_p(n)$ is a greatest power of p divides n

```
(b/order 2 24)
```

3

3.4 The greatest common divisor

The greatest common divisor of two integer a and b is an positive integer d which divide a and b and any other common divisor a and b divides d .

```
(b/gcd 12 18)
```

6

For convenience (gcd 0 0) is 0.

Furthermore, if for any two integers a and b exists integers s and t such that $as + bt = d$, where d is the greatest common divisor. For example, $6 = 12(-1) + 18(1)$

```
(b/gcd-extended 12 18)
```

```
[6 -1 1]
```

4 Performance and cache

This library is designed to work with realtive small integers. Library keep in cache least prime divisor table for fast integer factorization. Cache grows automatically. The strategy of growing is extends cache to the least power of 10 more than required number. For instance, if client asked to factorize number 18, cache grows to 100, if client asked to factorize number 343, cache grows to 1000. List of primes also cached and recalculated together with least prime divisor table. Recalculation is not incremental, but every recalculation of least prime divisor table make a table which is in 10 times more than previous, and time for previous calculation is 10 times less than for new one. So we can say that recalculation spent almost all time for recalculate latest least prime divisor table.

Internally, least prime divisor table is java array of int, so to store least divisor table for first 1 000 000 number approximately 4M memory is required, 4 bytes per number.

Cache can be reset:

```
(p/cache-reset!)
```

```
{:least-divisor-table , :primes , :upper 0}
```

Least prime divisor table is implementation details, but one can see it:

```
;; load first 10 numbers into cache
```

```
(p/int->factors-map 5)
```

```
{5 1}
```

For instance, for get least prime divisor of number 6 we need to get element with index 6, which is 2. Index zero is not used, value for index 1 is 1.

5 Primes

`primes` function returns prime numbers which not exceeds given `n`.

```
(p/primes 30)
```

```
(2 3 5 7 11 13 17 19 23 29)
```

6 Integer factorization

Every integer more than 1 can be represented uniquely as a product of primes.

$$n = p_1^{a_1} p_2^{a_2} \dots p_k^{a_k}$$

or we can write it in more compact form:

$$n = \prod_{i=1}^k p_i^{a_i}$$

or even write as:

$$n = \prod_{p|n} p^a$$

If we accept that empty product is 1 we can say that every natural number can be represent uniquely as a product of primes. For example $360 = 2^3 3^2 5^1$.

There are some functions to factorize integers. Each of them accept natural number as an argument and returns factorized value. It have slightly different output, which may be more appropriate to different use cases. For each factorize function there is also inverse function, which accept factorized value and convert it back to integer.

1-st factorization representation is ordered sequence of primes:

```
(p/int->factors 360)
```

```
(2 2 2 3 3 5)
```

```
(p/factors->int [2 2 2 3 3 5])
```

360

2-nd factorization representation is ordered sequence of primes splitted by partitions by a prime:

```
(p/int->factors-partitions 360)
```

```
((2 2 2) (3 3) (5))
```

```
(p/factors-partitions->int [[2 2 2] [3 3] [5]])
```

360

3-rd factorization representation is ordered sequence of pairs $[p \ k]$, where p is a prime and k is a power of prime

```
(p/int->factors-count 360)
```

```
([2 3] [3 2] [5 1])
```

```
(p/factors-count->int [[2 3] [3 2] [5 1]])
```

360

4-th factorization representation is very similar to 3-rd, but it is a map. And it has the same inverse function as 3-rd.

```
(p/int->factors-map 360)
```

```
{2 3, 3 2, 5 1}
```

```
(p/factors-count->int {2 3, 3 2, 5 1})
```

360

Implementation of factorization use least prime divisor table. To factorize number n it is enough to calculate least divisor table with size less or equals to \sqrt{n} .

7 Divisors

For get list of all divisors of number `n` there is `divisor` function. List of divisors is unordered.

```
(f/divisors 30)
```

```
(1 2 3 6 5 10 15 30)
```

8 Arithmetical functions

Arithmetical function is an any function which accept natural number and return complex number $f : \mathbf{N} \rightarrow \mathbf{C}$. The library mostly works with functions which also returns integer $f : \mathbf{N} \rightarrow \mathbf{Z}$.

8.1 Function equality

Two arithmetical function f and g are equal if $f(n) = g(n)$ for all natural n . There is helper function `f-equals` which compare two functions on some sequence of natural numbers. Function `f-equals` accept two functions and optionally sequence of natural numbers. There is a default for sequence of natural numbers, it is a variable `default-natural-sample`, which is currently `range(1,100)`.

If we like identify does two function `f` and `g` equals on some sequence of natural number we can for example do next:

```
;; Let we have some f and g
(def f identity)
(def g (constantly 1))
;; Then we able to check does those functions are equals
(f/f-equals f g)
(f/f-equals f g (range 1 1000))
(f/f-equals f g (filter even? (range 1 100)))
```

8.2 Additive functions

Additive function is a function for which

$$f(mn) = f(m) + f(n)$$

if m relatively prime to n . If above equality holds for all natural m and n function called completely additive.

To define an additive function it is enough to define how to calculate a function on power of primes. If $n = p_1^{a_1} p_2^{a_2} \dots p_k^{a_k}$ then:

$$f(n) = \sum_{i=1}^k f(p_i^{a_i})$$

8.3 Multiplicative functions

Multiplicative function is a function not equal to zero for all n for which

$$f(mn) = f(m)f(n)$$

if m relatively prime to n . If above equality holds for all natural m and n function called completely multiplicative.

To define multiplicative function it is enough to define how to calculate a function on power of primes. If $n = p_1^{a_1} p_2^{a_2} \dots p_k^{a_k}$ then:

$$f(n) = \prod_{i=1}^k f(p_i^{a_i})$$

8.4 Higher order function for define multiplicative and additive functions

As we have seen, to define either multiplicative or additive function it is enough define function on power of a prime. There is helper function `reduce-on-prime-count` which provide a way to define a function on power of a prime. The first parameter of `reduce-on-prime-count` is reduce function which usually `*` for multiplicative function and usually `+` for additive function, but custom reduce function also acceptable.

For instance, we can define function which calculate number of divisors of integer `n`. If $n = p_1^{a_1} p_2^{a_2} \dots p_k^{a_k}$ count of divisors of number `n` can be calculated by formula:

$$\sigma_0(n) = \prod_{i=1}^k (a_i + 1)$$

With helper function it can be defined as


```
(def my-divisors-count
  (f/reduce-on-prime-count * (fn [p k] (inc k))))
(my-divisors-count 6)
```

4

Of course there is predefined function `divisors-count`, but it is an example how to define custom function.

8.5 Some additive functions

8.5.1 Count of distinct primes - ω

Count of distinct primes is a number of distinct primes which divides given n . If $n = p_1^{a_1} p_2^{a_2} \dots p_k^{a_k}$ then $\omega = k$.

```
(f/primes-count-distinct (* 2 2 3))
```

2

8.5.2 Total count of primes - Ω

Total count of primes is a number of primes and power of primes which divides n . If $n = p_1^{a_1} p_2^{a_2} \dots p_k^{a_k}$ then:

$$\Omega = a_1 + a_2 + \dots + a_k$$

```
(f/primes-count-total (* 2 2 3))
```

3

8.6 Some multiplicative functions

8.6.1 Mobius function - μ .

Mobius function defined as:

$$\mu(n) = \begin{cases} 1 & \text{if } n = 1 \\ (-1)^k & \text{if } n \text{ product of distinct primes} \\ 0 & \text{otherwise} \end{cases}$$

For example, $\mu(6) = \mu(2 \cdot 3) = 1$

(f/mobius 6)

1

8.6.2 Euler totient function - ϕ

Euler totient function is a count of numbers relative prime to given number n . Totient function can be calculated by formula:

$$\phi(n) = \prod_{p|n} (p^a - p^{a-1})$$

For example, count of numbers relative prime to 6 are 1 and 5, so $\phi(6) = 2$

(f/totient 6)

2

8.6.3 Unit function - ϵ

Unit function defined as

$$\epsilon(n) = \begin{cases} 1, & \text{if } n = 1 \\ 0, & \text{if } n > 1 \end{cases}$$

(f/unit 6)

0

8.6.4 Constant one function - 1

$$1(n) = 1$$

(f/one 6)

1

8.6.5 Divisors count - σ_0

Divisors count is number of divisors which divides given number n .

$$\sigma_0(n) = \sum_{d|n} 1$$

For example, number 64 has 4 divisors, namely 1, 2, 3, 6, so $\sigma_0(6) = 4$

(f/divisors-count 6)

4

8.6.6 Divisors sum - σ_1

$$\sigma_1(n) = \sum_{d|n} d$$

For number 6 it is $12 = 1 + 2 + 3 + 6$

(f/divisors-sum 6)

12

8.6.7 Divisors square sum

$$\sigma_2(n) = \sum_{d|n} d^2$$

For number 6 it is $50 = 1^2 + 2^2 + 3^2 + 6^2$

(f/divisors-square-sum 6)

50

8.6.8 Divisors higher order function - σ_x

In general σ_x function is a sum of x-th powers divisors of given n

$$\sigma_x(n) = \sum_{d|n} d^x$$

If $x \neq 0$ σ_x can be calculated by formula:

$$\sigma_x(n) = \prod_{i=1}^k \frac{p_i^{(a_i+1)x} - 1}{p_i^x - 1}$$

and if $x = 0$ by formula:

$$\sigma_0(n) = \prod_{i=1}^k (a_i + 1)$$

There is higher order function `divisors-sum-x` which accept `x` and return appropriate function.

```
(def my-divisors-square-sum (f/divisors-sum-x 2))
```

8.6.9 Liouville - λ

Liouville function can be defined by formula:

$$\lambda(n) = (-1)^{\Omega(n)}$$

where Ω have been described above.

```
(f/liouville (* 2 3))
```

1

8.7 Some other arithmetic functions

8.7.1 Mangoldt - Λ

$$\Lambda(n) = \begin{cases} \log p, & \text{if } n \text{ is power of prime i.e. } n = p^k \\ 0, & \text{otherwise} \end{cases}$$

For example $\Lambda(8) = \log 2$, $\Lambda(6) = 0$

(f/mangoldt 2)

0.6931471805599453

(f/mangoldt 6)

0

8.7.2 Chebyshev functions θ and ψ

There are two Chebyshev functions, one θ is defined as

$$\theta(x) = \sum_{p \leq x} \log p$$

second ψ defined as

$$\psi = \sum_{n \leq x} \Lambda(n)$$

where Λ have been described above

(f/chebyshev-first 2)

0.6931471805599453

(f/chebyshev-second 2)

0.6931471805599453

8.8 Dirichlet convolution

For two arithmetic functions f and g Dirichlet convolution is a new arithmetic function defined as

$$(f * g)(n) = \sum_{d|n} f(d)g\left(\frac{n}{d}\right)$$

Dirichlet convolution is associative

$$(f * g) * h = f * (g * h)$$

Commutative

$$f * g = g * f$$

Has identify

$$f * \epsilon = \epsilon * f = f$$

For every f , which $f(1) \neq 0$ exists inverse function f^{-1} such that $f * f^{-1} = \epsilon$. This inverse function called Dirichlet inverse and can be calculated recursively by formula:

$$f^{-1}(n) = \begin{cases} \frac{1}{f(1)} & \text{if } n = 1 \\ \frac{-1}{f(1)} \sum_{d|n, d < n} f\left(\frac{n}{d}\right) f^{-1}(d) & n \geq 1 \end{cases}$$

For example, $1(n) * 1(n) = \sigma_0$

```
(f/f-equals
  (f/d-* f/one f/one)
  f/divisors-count
)
```

```
class clojure.lang.Compiler$CompilerException
```

Dirichlet convolution is associative so clojure method support more than two function as parameter of `f*`

```
(f/f-equals
  (f/d-* f/mobius f/one f/mobius f/one)
  f/unit
)
```

```
class clojure.lang.Compiler$CompilerException
```

Another example, functions $\mu(n)$ and $1(n)$ are inverse of each other

```
(f/f-equals (f/d-inv f/one) f/mobius)
```

```
class clojure.lang.Compiler$CompilerException
```

```
(f/f-equals (f/d-inv f/mobius) f/one)
```

```
class clojure.lang.Compiler$CompilerException
```

Function `d-inv` defined as recursive function, it may execute slow. But inverse of completely multiplicative function $f(n)$ is $f(n)\mu(n)$ (usual multiplication), for instance inverse of identity function, let's denote it $N(n)$ is $N(n)\mu(n)$

```
(f/f-equals  
  (f/d-*  
    #(* (identity %) (f/mobius %))  
    identity  
  )  
  f/unit)
```

```
class clojure.lang.Compiler$CompilerException
```