mathslib

Release 2.4.0

Igor van Loo

May 26, 2023

CONTENTS:

1 Overview **1**
- 1.1 Breakdown . . . 3

2 Installation **5**
- 2.1 Stable release . . . 5
- 2.2 From sources . . . 5

3 Usage **7**

4 mathslib **9**
- 4.1 mathslib package . . . 9

5 Indices and tables **27**

Python Module Index **29**

Index **31**

CHAPTER

ONE

OVERVIEW

mathslib is a compilation of Mathematical Functions and Algorithms. Unless credit was given all of the functions were written by me. Relevant articles are also linked where the implementation is complex.

I have used most of these for Project Euler.

See my website ivl-projecteuler.com for their implementation

See the full documentation here

1.1 Breakdown

numtheory.py	• divisors_of(x, include_x) • divisors(x, n) • continued_fraction(x) • overall_fraction(x) • phi(x) • phi_sieve(x) • phi_sum(x) • mobius(x) • mobius_k_sieve(limit, k) • count_k_free(n, k) • ppt(limit, non_primitive) • k_smooth_numbers(max_prime, limit) • k_powerful(k, limit, count) • legendre_factorial(x) • tonelli_shanks(a, p) • chinese_remainder_theorem(a1, a2, n1, n2) • generalised_CRT(a1, a2, n1, n2) • frobenius_number(*integers)
prime.py	• prime_sieve(limit, block_size, segment, values) • is_prime(x) • prime_factors(x) • primepi(x) • primepi_sieve(x) • sum_of_primes(x) • fermat_primality_test(x) • miller_primality_test(n, millerrabin, numoftests)
linalg.py	• gauss_jordan_elimination(matrix, augmented-part) • solve(M, b) • inverse(matrix) • determinant(matrix) • matrix_addition(A, B, subtract) • identity(l, val) • concatenate(A, B) • argmax(alist) • fillmatrix(size, val) • matrix_mul(A, B)
fib.py	• fibonacci(n) • fib_till(limit) • zeckendorf_representation(x)
algorithms.py	• prims_algorithm(matrix) • dijkstras_algorithm(graph, start_node, INFIN-ITY) • floyd_warshall_algorithm(graph, INFINITY) • knap_sack(values, weights, n, W, no_values) • knap_sack_values(values, weights, n, W, no_values) • BFS(g, start_node, end_node) • DFS(g, start_node, end_node) • convex_hull_gift_wrapping(pts)

CHAPTER

TWO

INSTALLATION

2.1 Stable release

To install mathslib, run this command in your terminal:

```
$ pip install mathslib
```

or if you wish to update the package

```
$ pip install mathslib --upgrade
```

This is the preferred method to install mathslib, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for mathslib can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/igorvanloo/mathslib
```

CHAPTER

THREE

USAGE

In the python console you can import functions as you need them

```
from mathslib import divisor
print(divisor(2, 9)) #91
```

Otherwise you can import the whole module and use the mathslib. prefix

```
import mathslib
print(mathslib.tonelli_shanks(5, 41)) #28
print(mathslib.ChineseRemainderTheorem(2, 3, 3, 5)) #8
print(mathslib.ZeckendorfRepresentation(64)) #[55, 8, 1]
```

Example Solved Project Euler problem 10.

```
from mathslib import sum_of_primes
print(sum_of_primes(2*10**6)) #142913828922
```

CHAPTER

FOUR

MATHSLIB

4.1 mathslib package

4.1.1 Submodules

4.1.2 mathslib.numtheory module

Various Number Theory functions

Author: Igor van Loo

mathslib.numtheory.**divisors_of**(*x*, *include_x=True*)

Finds all the divisors of x

Parameters

- **x** – Integer to be checked
- **include_x** – Optional boolean value, If true it will include x as a divisor of x

Returns

A list which contains all divisors of x

```
print(divisors_of(15)) #[1, 3, 5, 15]
print(divisors_of(15, include_x = False)) #[1, 3, 5]
```

mathslib.numtheory.**divisor**(*x*, *n*)

Implementation of Divisor function sigma(x, n)

Parameters

- **x** – An integer, denotes the power till which the divisors will be summed
- **n** – An integer, denotes the number to find the divisors of

Returns

An integer

```
print(divisor(0, 9)) #3
print(divisor(1, 9)) #13
print(divisor(2, 9)) #91
```

mathslib.numtheory.**continued_fraction**(*x*)

Finds the continued Fraction of the sqrt(x)

Parameters
x – An integer

Returns
A list containing the continued fraction of x

```
print(continued_fraction(19)) #[4, 2, 1, 3, 1, 2, 8]
```

Note: The continued fraction may repeat forever, the code stops once it repeats, otherwise once we find the 100th number in the continued fraction

mathslib.numtheory.**overall_fraction**(*cf*)

Parameters
cf – A list, this list represents the continued fraction of a number

Returns numerator
An integer, the numerator of the fraction

Returns denominator
An integer, the denominator of the fraction

```
print(overall_fraction([4, 2, 6, 7])) #(415, 93)
```

mathslib.numtheory.**phi**(*n*)

Implementation of Eulers Totient Function counts the positive integers up to a given integer n that are relatively prime to n

Parameters
n – An integer

Returns
An integer, numbers, a, less than n, such that gcd(a, n) = 1

```
print(phi(20)) #8
print(phi(100)) #40
```

mathslib.numtheory.**phi_sieve**(*n*)

A sieve for Eulers Totient Function which counts the positive integers up to a given integer n that are relatively prime to n

Parameters
n – An integer

Returns
An array, where array[x] = phi(x)

```
print(phi_sieve(10)) #[0, 1, 1, 2, 2, 4, 2, 6, 4, 6, 4]
print(phi_sieve(20)[11:]) #[10, 4, 12, 6, 8, 8, 16, 6, 18, 8]
```

mathslib.numtheory.**phi_sum**(*n*)

Computes the Totient Summatory Function

The algorithm is based on Overview of Project Euler Problem 351 specifically, this is an implementation of Algorithm 6, which you may view after Solving Problem 351

Parameters
n – An integer

Returns
sum of phi(x) where x goes from 1 to n

```
print(phi_sum(10**4)) #30397486
print(sum(phi(i) for i in range(1, 10**4 + 1))) #30397486
print(sum(phi_sieve(10**4))) #30397486
```

mathslib.numtheory.**mobius**(*n*)

Implementation of the Mobius function of n

Parameters
n – An integer

Returns
An integer

```
print(Mobius(10)) #1 - 10 = 2*5, therefore (10) = (-1)*(-1) = 1
print(Mobius(9)) #0 - Divisble by 3*3
print(Mobius(7)) #-1 - 7 is prime therefore (7) = -1
```

Note:

- returns 0 if n is divisible by p^2, where p is a prime
- returns (-1)^k, where k is number of distinct prime factors

---

mathslib.numtheory.**mobius_k_sieve**(*limit*, *k*)

A sieve for the Generalized Mobius function, the mathematics of this function can be read in the following PDF

Parameters

- **limit** – An integer
- **k** – An integer

Returns
An array where array[x] = (k, x)

```
print(mobius_k_sieve(10, 2)) #[0, 1, -1, -1, 0, -1, 1, -1, 0, 0, 1]
print(mobius_k_sieve(10, 3)) #[0, 1, -1, -1, -1, -1, 1, -1, 0, -1, 1]
```

mathslib.numtheory.**count_k_free**(*n*, *k*)

A function that counts the integers n which are k-power free. count_k_free(n, 2) would count the number of squarefree integers. The mechanics of this function come from this PDF

Parameters

- **n** – An integer
- **k** – An integer

Returns
The number of integers n which are k-power free

From Project Euler Problem 193

```
print(count_k_free(2**50, 2)) #684465067343069
```

mathslib.numtheory.**ppt**(*limit*, *non_primitive=True*)

Generates all Pythagorean Triplets up to the limit

Parameters

- **limit** – An integer, will generate all Pythagorean Triplets such that no side is longer than the limit
- **non_primitive** – Optional boolean value, If True, returns all triplets, if False returns only primitive triplets

Returns

A list containing all desired triplets

```
print(ppt(20)) #[[3, 4, 5], [6, 8, 10], [9, 12, 15], [12, 16, 20], [5, 12, 13], [15,
↪ 8, 17]]
print(ppt(20, False)) #[[3, 4, 5], [5, 12, 13], [15, 8, 17]]
print(len(ppt(100, False))) #16
```

mathslib.numtheory.**legendre_factorial**(*x*)

Implementation of Legendres' Formula

Parameters

x – An integer

Returns

A dictionary containing the prime factorisation of x!

```
print(legendre_factorial(6)) #{2: 4, 3: 2, 5: 1}
```

mathslib.numtheory.**k_smooth_numbers**(*max_prime*, *limit*)

Find all k max_prime smooth numbers up to the limit

Parameters

- **max_prime** – The maximum prime allowed
- **limit** – limit up till which to find max_prime smooth numbers

Returns

A list containing all k max_prime smooth numbers less that limit

From Project Euler Problem 204

```
print(len(k_smooth_numbers(5, 10**8))) #1105
```

mathslib.numtheory.**k_powerful**(*k*, *limit*, *count=True*)

Find all k-powerful numbers less than or equal to upper_bound. Inspired by Rosetta

Parameters

- **k** – k, representing k-powerful
- **limit** – limit up till which to k-powerful numbers
- **count** – Optional, Boolean

Returns

if count is True, it counts the number of k-powerful numbers, otherwise it will return them as a list

```
print(kpowerful(2, 10^2)) #14
print(kpowerful(2, 10^2, False)) #[1, 4, 8, 9, 16, 25, 27, 32, 36, 49, 64, 72, 81,␣
↪100]
```

mathslib.numtheory.**legendre_symbol**(*a*, *p*)

Finds the legendre symbol of a/p

Parameters

- **a** – An integer
- **p** – An odd prime

Results

The legendre symbol of a/p

```
print(legendre_symbol(3, 3)) #0
print(legendre_symbol(10, 31)) #1
print(legendre_symbol(2, 11)) #-1
```

Note:

- returns 1 if a is a quadratic residue modulo p and p does not divide a
- returns -1 if a is a non-quadratic residue modulo p
- returns 0 if p divides a

---

mathslib.numtheory.**tonelli_shanks**(*a*, *p*)

Implementation of Tonelli Shanks algorithm

Full credit for this alogrithm goes to Eli Bendersky

Parameters

- **a** – An integer
- **p** – An integer

Returns

solution to x^2 = a (mod p)

```
print(tonelli_shanks(5, 41)) #28
```

Note: This function solves the congruence of the form x^2 = a (mod p) and returns x. Note that p - x is also a root.

0 is returned if no square root exists for these a and p.

---

mathslib.numtheory.**chinese_remainder_theorem**(*a1*, *a2*, *n1*, *n2*)

Simple Chinese Remiander Theorem to solve x = a1 mod n1, x = a2 mod n2

Parameters

- **a1** – An integer
- **a2** – An integer
- **n1** – An integer

- **n2** – An integer

Returns
Unique solution to x = a1 mod n1, x = a2 mod n2

```
#We solve x = 2 mod 3 = 3 mod 5 = 2 mod 7
#First we solve x = 2 mod = 3 mod 5
print(chinese_remainder_theorem(2, 3, 3, 5)) #8
#Then we solve x = 8 mod 15 = 2 mod 7
print(chinese_remainder_theorem(8, 2, 15, 7)) #23
```

mathslib.numtheory.**generalised_CRT**(*a1*, *a2*, *n1*, *n2*)

A generalised Chinese Remiander Theorem which solves for non-coprime moduli

Parameters

- **a1** – An integer
- **a2** – An integer
- **n1** – An integer
- **n2** – An integer

Returns
Unique solution to x = a1 mod n1, x = a2 mod n2

```
print(generalised_CRT(2, 3, 3, 5)) #8, note that we can use ChineseRemainderTheorem␣
↪function for this case
print(generalised_CRT(2, 4, 4, 6)) #10
print(generalised_CRT(3, 4, 4, 6)) #'No solution'
```

mathslib.numtheory.**frobenius_number**(**integers*)

Generates the Frobenius Number for given integers.

The below algorithm is based on Faster Algorithms for Frobenius Numbers specifically, this is an implementation of their Breadth-First Method which you may find on page 9

Param
integers

Returns
Frobenius number

```
print(frobenius_number(3, 5)) #7
print(frobenius_number(6, 9, 20)) #43
print(frobenius_number(1000, 1476, 3764, 4864, 4871, 7773)) #47350
```

4.1.3 mathslib.primes module

Prime related functions

Author: Igor van Loo

mathslib.primes.**prime_sieve**(*limit*, *segment=False*, *values=True*)

A prime sieve I made with a few different options

Parameters

- **limit** – The limit up till which the function will generate primes

- **segment** – Optional boolean value, if segment == True, it will perform a segmented sieve
- **values** – Optional boolean value, if values == False, it will return an array such that array[x] = True if x is prime

Returns
All primes < limit

```
print(prime_sieve(50)) #[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
print(prime_sieve(10, values = False)) #[False, False, True, True, False, True,␣
↪False, True, False, False, False]

print([i for (i, isprime) in enumerate(prime_sieve(10, values = False)) if␣
↪isprime]) #[2, 3, 5, 7]
```

mathslib.primes.**is_prime**(*x*)

A simple is prime function, checks if a number is prime

Parameters
x – An integer

Returns
True if x is prime, False otherwise

```
print(is_prime(10)) #False
print(is_prime(160517089)) #True
```

mathslib.primes.**prime_factors**(*n*)

Finds all the prime factors of n

Parameters
n – An integer

Returns
A dictionary containing the prime divisors as keys and value as the corresponding exponent of the prime

```
print(prime_factors(123123)) #{3: 1, 7: 1, 11: 1, 13: 1, 41: 1}
print(prime_factors(1123619623)) #{7: 1, 160517089: 1}
```

mathslib.primes.**primepi**(*x*)

Primepi function is commonly known as Prime Counting Function

This function computes primepi(x) based on the Meissel-Lehmer Method

The following article by Adithya Ganesh helped me a lot in understanding and implementing this function

Parameters
x – An integer

Returns
primepi(x)

```
print(primepi(10**6)) #78498
print(primepi(10**7)) #664579
print(primepi(10**8)) #5761455
print(primepi(10**9)) #50847534
```

mathslib.primes.**primepi_sieve**(*limit*)

Primepi function is commonly known as Prime Counting Function

This function generates an array such that array[x] = primepi(x)

Parameters
limit – An integer,

Returns
An array length limit such that array[x] = primepi(x)

```
print(primepi_sieve(10)) #[0, 0, 1, 2, 2, 3, 3, 4, 4, 4, 4]
```

mathslib.primes.**sum_of_primes**(*n*)

Ultra fast sum of Primes made by Lucy The HedgeHog on Project Euler

You may view it here once you've completed problem 10

Parameters
n – An integer

Returns
The sum of all primes up till n

```
print(sum_of_primes(2*10**6)) #142913828922
print(sum_of_primes(10**10)) #22208224325817292 38
```

mathslib.primes.**fermat_primality_test**(*n*, *tests=2*)

A Fermat Primality Test

Parameters

- **n** – The integer to be tested
- **tests** – Optional, Number of tests to make. The default is 2

Returns
True if n is probably prime

```
print(fermat_primality_test(17969800575241)) #True and it is actually True
print(fermat_primality_test(101101)) #True but it is wrong 101101 is not prime
print(fermat_primality_test(101101, 6)) #False, after using 6 tests we see that it␣
↪is not prime
```

Note: This function will always guess a prime correctly due to Fermats Theorem, but may guess a composite to be a prime. Therefore, it is very useful when we test large numbers, otherwise it is dangerous to use, and hence if n < 10^5 the program will automatically use the is_prime function

You can test more terms to make it more accurate

---

mathslib.primes.**miller_primality_test**(*n*, *millerrabin=False*, *numoftests=2*)

The Miller Primality Test with the option to use the Miller-Rabin test

Parameters

- **n** – The integer to be tested
- **millerrabin** – Optional, default False. Decides with the use Miller-Rabin or Miller primality test

- **numoftests** – Optional, default is 2. If millerrabin is True, it uses numoftests bases

Returns

True if n is probably prime, False if n is not prime

```
print(miller_primality_test(17969800575241)) #True
print(miller_primality_test(101101)) #False
print(len([x for x in range(1, 10**6) if miller_primality_test(x, True, 1)]))
↪#78544, 46 more primes than needed
print(len([x for x in range(1, 10**6) if miller_primality_test(x, True, 2)]))
↪#78498, correct now
```

Note: Automatically uses the Miller Primality Test to get an exact an answer if n < 3317044064679887385961981 and swaps to using the first 17 primes, but no longer guarantees a correct answer. You may optionally use a regular miller-rabin test with a specified number of tests

---

4.1.4 mathslib.linalg module

Various Linear Algebra Functions

Author: Igor van Loo

mathslib.linalg.**gauss_jordan_elimination**(*matrix*, *augmentedpart=None*)

Performs Gauss Jordan Elimination on the given matrix

Parameters

- **matrix** – Matrix to perform Algoithm on
- **augmentedpart** – Optional argument, will attach the augmented part onto the matrix and then perform the algorithm

Returns

True if algorithm was successful, false otherwise

```
matrix = [[2, 1, -1],
          [-3, -1, 2],
          [-2, 1, 2]]
print(gauss_jordan_elimination(matrix)) #True
```

Note: This function simply performs the Gauss Jordan Algorithm, it is used with with solve and inverse to solve Ax = b, and the find the inverse of a matrix

---

mathslib.linalg.**solve**(*M*, *b*)

Finds the solution, x, to the equation Mx = b

Parameters

- **M** – A Matrix
- **b** – A Vector

Returns

The Vector x, or an error message

```
matrix = [[2, 1, -1],
          [-3, -1, 2],
          [-2, 1, 2]]
b = [[8],
     [-11],
     [-3]]
print(solve(matrix, b)) #[[2.0], [3.0], [-1.0]]
```

mathslib.linalg.**inverse**(*matrix*)

Finds the inverse of the given matrix by performing Gauss Jordan Elimination

Parameters
matrix – Matrix to be inverted

Returns
Inverted matrix, or an error message

```
matrix = [[1, -1, 0],
          [-8, 9, -1],
          [-9, 0, 10]]
print(inverse(matrix)) #[[90.0, 10.0, 1.0], [89.0, 10.0, 1.0], [81.0, 9.0, 1.0]]
```

mathslib.linalg.**determinant**(*matrix*)

Finds the determinant of a matrix

Parameters
matrix – Matrix

Returns
det(Matrix)

```
matrix = [[7, -1, 0],
          [-8, 9, -1],
          [-9, 0, 10]]
print(determinant(matrix)) #541.0
```

mathslib.linalg.**matrix_addition**(*A*, *B*, *subtract=False*)

Performs matrix addition/subtraction on matrix A

Parameters

- **A** – First Matrix
- **B** – Second Matrix
- **subtract** – If True, will perform matrix subtraction

Returns
A + B

```
matrix = [[1, 0, 0],
          [0, 1, 0],
          [0, 0, 1]]
print(matrix_addition(matrix, matrix)) #[[2, 0, 0], [0, 2, 0], [0, 0, 2]]
print(matrix_addition(matrix, matrix, True)) #[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

mathslib.linalg.**identity**(*l*, *val=1*)

Generats an Square Identity Matrix

Parameters

- **l** – Size of matrix
- **val** – diagonal values, default is 1

Returns

val * identity matrix of size l

```
print(identity(2)) #[[1, 0], [0, 1]]
print(identity(2, 5)) #[[5, 0], [0, 5]]
```

mathslib.linalg.**concatenate**(*A*, *B*)

Concatenates 2 matrices, A and B

Parameters

- **A** – First Matrix
- **B** – Second Matrix

Returns

A concatenated with B

```
A = [[1, 0],
     [0, 1]]
print(concatenate(A, A)) #[[1, 0, 1, 0], [0, 1, 0, 1]]
```

Note: This is a helper function for inverse and solve

mathslib.linalg.**argmax**(*alist*)

Finds the argmax of a list

Parameters

alist – A list

Returns

the arg max of the list

```
print(argmax([1, 3, 2])) #1
```

mathslib.linalg.**fillmatrix**(*size*, *val=0*)

Fills a matrix with a value

Parameters

- **size** – A tuple, denoted (width, height) of matrix
- **val** – Value to fill matrix with, default is 0

Returns

A matrix

```
print(fillmatrix((2, 2))) #[[0, 0], [0, 0]]
print(fillmatrix((2, 3))) #[[0, 0, 0], [0, 0, 0]]
```

mathslib.linalg.**matrix_mul**(*A*, *B*)

Matrix multplication of 2 matrices, A and B

Parameters

- **A** – First Matrix
- **B** – Second Matrix

Returns

Matrix AB

```
A = [[2, 0],
     [0, 2]]
B = [[1, 2],
     [3, 1]]
print(matrix_mul(A, B)) #[[2, 4], [6, 2]]
```

4.1.5 mathslib.fib module

Fibonacci related functions

Author: Igor van Loo

mathslib.fib.**fibonacci**(*n*)

Finds the n-th Fibonacci using matrix exponentiation Method is outlined here

Parameters

n – An integer

Returns

The n-th Fibonacci number

```
print(fibonacci(100)) #354224848179261915075
```

mathslib.fib.**fib_till**(*limit*)

Finds all Fibonacci number up till a limit

Parameters

limit – An integer

Returns

A list containing all the fibonacci numbers < limit

```
print(fib_till(100)) #[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
print(sum(fib_till(1000))) #2583
```

mathslib.fib.**zeckendorf_representation**(*x*)

Finds the Zeckendorf Representation of x

Parameters

x – An integer

Returns

A list containing the zeckendorf representation of x

```
print(zeckendorf_representation(64)) #[55, 8, 1]
```

4.1.6 mathslib.algorithms module

Various known algorithms. They are graph theory and optimization related

Author: Igor van Loo

mathslib.algorithms.**prims_algorithm**(*matrix*)

Implementation of Prim's algorithm It finds a Minimum Spanning Tree (MST) for a weighted undirected graph.

Parameters
matrix – Takes a Adjacency matrix as input

Returns Weight
The sum of the minimum spanning tree

Returns mask
The corresponding Adjacency matrix of the MST

Example from Project Euler Problem 107

```
matrix = [[0, 16, 12, 21, 0, 0, 0],
          [16, 0, 0, 17, 20, 0, 0],
          [12, 0, 0, 28, 0, 31, 0],
          [21, 17, 28, 0, 18, 19, 23],
          [0, 20, 0, 18, 0, 0, 11],
          [0, 0, 31, 19, 0, 0, 27],
          [0, 0, 0, 23, 11, 27, 0]]

print(prims_algorithm(matrix)) #(93,
                               #[[0, 16, 12, 0, 0, 0, 0],
                               #[16, 0, 0, 17, 0, 0, 0],
                               #[12, 0, 0, 0, 0, 0, 0],
                               #[0, 17, 0, 0, 18, 19, 0],
                               #[0, 0, 0, 18, 0, 0, 11],
                               #[0, 0, 0, 19, 0, 0, 0],
                               #[0, 0, 0, 0, 11, 0, 0]])
```

mathslib.algorithms.**dijkstras_algorithm**(*graph*, *start_node=0*, *INFINITY=10000000000*)

Implementation of Dijkstra's algorithm It finds the the shortest paths between nodes in a graph

Parameters

- **graph** – Takes an adjacency list as input
- **start_node** – Optional tuple, default is node 0.
- **INFINITY** – Optional integer, default is 10^10. It is used to set the "Infinty" value

Returns
Shortest path between start_node and all other nodes

Example from the wikipedia page

```
g = [[[1, 7], [2, 9], [5, 14]],
     [[0, 7], [2, 10], [3, 15]],
     [[0, 9], [1, 10], [3, 11], [5, 2]],
     [[1, 15], [2, 11], [4, 6]],
     [[3, 6], [5, 9]],
     [[0, 14], [2, 2], [4, 9]]
```

(continues on next page)

(continued from previous page)

```
    ]

print(dijkstras_algorithm(g)) #[0, 7, 9, 20, 20, 11]
```

Note: A quick comment on this adjacency list. The way it works is for example g[i] contains all the nodes node i is connected to. For example, using the above graph g[0] = [[1, 7], [2, 9], [5, 14]] means node 0 is connected to nodes 1, 2, and 5 and the weight between the edges are 7, 9, and 14 respectively

mathslib.algorithms.**floyd_warshall_algorithm**(*graph*, *INFINITY=10000000000*)

Implementation of the Floyd-Warshall algorithm It finds the the shortest paths between every node in the graph to every node in the graph

Parameters

- **graph** – Takes an adjacency list as input
- **INFINITY** – Optional integer, default is 10^10. It is used to set the "Infinty" value

Returns

Shortest path between every node to every node

Example from the wikipedia page

```
g = [[[1, 7], [2, 9], [5, 14]],
     [[0, 7], [2, 10], [3, 15]],
     [[0, 9], [1, 10], [3, 11], [5, 2]],
     [[1, 15], [2, 11], [4, 6]],
     [[3, 6], [5, 9]],
     [[0, 14], [2, 2], [4, 9]]
    ]

print(floyd_warshall_algorithm(g)) #[[0, 7, 9, 20, 20, 11],
                                   # [7, 0, 10, 15, 21, 12],
                                   # [9, 10, 0, 11, 11, 2],
                                   # [20, 15, 11, 0, 6, 13],
                                   # [20, 21, 11, 6, 0, 9],
                                   # [11, 12, 2, 13, 9, 0]]
```

Note: This process is like applying Dijkstras Algorithm on every node

mathslib.algorithms.**knap_sack**(*values*, *weights*, *n*, *W*, *no_values=True*)

Implementation of dynamic programming solution to the 0-1 Knapsack Problem

Parameters

- **values** – A list of values
- **weights** – A list with weight of corresponding values
- **n** – Number of items
- **W** – Desired weight
- **no_values** – Optional boolean value

Returns

- If no_values == True - It returns the optimal sum of weights
- If no_values == False - it returns the entire array used to build up the solution

```
values = [60, 100, 120]
weights = [10, 20, 30]
W = 50
n = len(values)

print(knap_sack(values, weights, n, W)) #220
```

mathslib.algorithms.**knap_sack_values**(*values*, *weights*, *n*, *W*)

Extension to KnapSack function It finds the actual values used to obtain the optimal sum

Parameters

- **values** – A list of values
- **weights** – A list with weight of corresponding values
- **n** – Number of items
- **W** – Desired weight

Returns

A set with the optimal values which form the solution to the knapsack problem

```
values = [60, 100, 120]
weights = [10, 20, 30]
W = 50
n = len(values)

print(knap_sack_values(values, weights, n, W)) #{20, 30}
```

mathslib.algorithms.**BFS**(*g*, *start_node=0*, *end_node=False*)

Implementation of Breadth First Search

Parameters

- **g** – An Adjacency List
- **start_node** – Optional, pick your start node. Default is 1st node
- **end_node** – Optional, pick your end node. Default is last node

Returns

A list of nodes which create a path from your start_node to end_node if it exists

```
G = [[4, 1], [0, 5], [6, 3], [2, 7],
     [0, 8], [1, 6], [2, 5, 10], [3, 11],
     [4, 9], [8, 13], [6], [7, 15],
     [13], [9, 12, 14], [13, 15], [11, 14] ]

print(BFS(G)) #[0, 4, 8, 9, 13, 14, 15]
```

Note: A quick comment on adjacency lists. The way it works is for example G[i] contains all the nodes node i is connected to. For example using the above graph G[0] = [4, 1] means node 0 is connected to nodes 1, and 4.

---

mathslib.algorithms.**DFS**(*g*, *start_node=0*, *end_node=False*)

Implementation of Depth First Search

Parameters

- **g** – An Adjacency List
- **start_node** – Optional, pick your start node. Default is 1st node
- **end_node** – Optional, pick your end node. Default is last node

Returns

A list of nodes which create a path from your start_node to end_node if it exists

```
G = [[4, 1], [0, 5], [6, 3], [2, 7],
     [0, 8], [1, 6], [2, 5, 10], [3, 11],
     [4, 9], [8, 13], [6], [7, 15],
     [13], [9, 12, 14], [13, 15], [11, 14] ]

print(DFS(G)) #[0, 1, 5, 6, 2, 3, 7, 11, 15]
```

mathslib.algorithms.**convex_hull_gift_wrapping**(*pts*)

Implementation of the Convex Hull Gift Wrapping Algorithm

Parameters

pts – A list containing 2D points

Returns

A list of points consisting of the convex hull starting from leftmost point going around

mathslib.algorithms.**convex_hull_DC**(*pts*)

Implementation of the Convex Hull Divide and conquer Algorithm

Parameters

pts – A list containing 2D points

Returns

A list of points consisting of the convex hull starting from leftmost point going around

4.1.7 mathslib.simple module

Various simple functions

Author: Igor van Loo

mathslib.simple.**n_choose_r**(*n*, *r*)

n choose r function

Parameters

- **n** – An integer
- **r** – An integer

Returns

n choose r

```
print(n_choose_r(50, 30)) #47129212243960
```

mathslib.simple.**number_to_base**(*n*, *b*)

Changes n from base 10 to base b

Parameters

- **n** – An integer, number to be changed
- **b** – An integer, base in question

Returns

n in base b

```
print(number_to_base(10, 2)) #[1, 0, 1, 0]
print(number_to_base(10, 3)) #[1, 0, 1]
```

mathslib.simple.**extended_euclidean_algorithm**(*a*, *b*)

Standard Extended Euclidean Algorithm

Parameters

- **a** – An integer
- **b** – An integer

Returns

A tuple (g, s, t) where gcd(a, b) = g = as + bt

```
print(extended_euclidean_algorithm(240, 46)) #(2, -9, 47)
```

mathslib.simple.**lcm**(*a_list*)

Finds the lcm of a list of numbers

Parameters

alist – A list containing integers

Returns

The lcm of all numbers in the list

```
print(lcm([2, 3])) #6
print(lcm([2, 4, 5, 7])) #140
print(lcm([8345, 23579, 174])) #34237415370
```

mathslib.simple.**mod_division**(*a*, *b*, *m*)

Finds a/b mod m

Parameters

- **a** – An integer, the numerator
- **b** – An integer, the denominator
- **m** – An integer, the modulus

Returns

a/b mod m

```
print(mod_division(8, 4, 5)) #2
```

mathslib.simple.**bisect**(*alist*, *goal*)

This function is equivalent to bisect_right from the bisect module

Parameters

- **alist** – A list
- **goal** – A number

Returns
index i of A such that A[i - 1] < g <= A[i]

```
print(bisect([2, 3, 5, 7], 6)) #3 since A[2] = 5 < 6 <= A[3] = 7
```

mathslib.simple.**is_clockwise**(*a*, *b*, *c*)

Finds if 3 points a going to b going to c are in clockwise order. It is used in convex hull algorithm

Parameters

- **a** – A tuple, representing a point in 2D
- **b** – A tuple, representing a point in 2D
- **c** – A tuple, representing a point in 2D

Returns
True if point are in clockwise direction, otherwise False

4.1.8 Module contents

Library of Mathematical functions and Algorithms

Author: Igor van Loo

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

mathslib, 26
mathslib.algorithms, 21
mathslib.fib, 20
mathslib.linalg, 17
mathslib.numtheory, 9
mathslib.primes, 14
mathslib.simple, 24

INDEX

A

argmax() (*in module mathslib.linalg*), 19

B

BFS() (*in module mathslib.algorithms*), 23
bisect() (*in module mathslib.simple*), 25

C

chinese_remainder_theorem() (*in module mathslib.numtheory*), 13
concatenate() (*in module mathslib.linalg*), 19
continued_fraction() (*in module mathslib.numtheory*), 9
convex_hull_DC() (*in module mathslib.algorithms*), 24
convex_hull_gift_wrapping() (*in module mathslib.algorithms*), 24
count_k_free() (*in module mathslib.numtheory*), 11

D

determinant() (*in module mathslib.linalg*), 18
DFS() (*in module mathslib.algorithms*), 23
dijkstras_algorithm() (*in module mathslib.algorithms*), 21
divisor() (*in module mathslib.numtheory*), 9
divisors_of() (*in module mathslib.numtheory*), 9

E

extended_euclidean_algorithm() (*in module mathslib.simple*), 25

F

fermat_primality_test() (*in module mathslib.primes*), 16
fib_till() (*in module mathslib.fib*), 20
fibonacci() (*in module mathslib.fib*), 20
fillmatrix() (*in module mathslib.linalg*), 19
floyd_warshall_algorithm() (*in module mathslib.algorithms*), 22
frobenius_number() (*in module mathslib.numtheory*), 14

G

gauss_jordan_elimination() (*in module mathslib.linalg*), 17
generalised_CRT() (*in module mathslib.numtheory*), 14

I

identity() (*in module mathslib.linalg*), 18
inverse() (*in module mathslib.linalg*), 18
is_clockwise() (*in module mathslib.simple*), 26
is_prime() (*in module mathslib.primes*), 15

K

k_powerful() (*in module mathslib.numtheory*), 12
k_smooth_numbers() (*in module mathslib.numtheory*), 12
knap_sack() (*in module mathslib.algorithms*), 22
knap_sack_values() (*in module mathslib.algorithms*), 23

L

lcm() (*in module mathslib.simple*), 25
legendre_factorial() (*in module mathslib.numtheory*), 12
legendre_symbol() (*in module mathslib.numtheory*), 13

M

mathslib
    module, 26
mathslib.algorithms
    module, 21
mathslib.fib
    module, 20
mathslib.linalg
    module, 17
mathslib.numtheory
    module, 9
mathslib.primes
    module, 14
mathslib.simple
    module, 24

matrix_addition() (*in module mathslib.linalg*), 18
matrix_mul() (*in module mathslib.linalg*), 19
miller_primality_test() (*in module mathslib.primes*), 16
mobius() (*in module mathslib.numtheory*), 11
mobius_k_sieve() (*in module mathslib.numtheory*), 11
mod_division() (*in module mathslib.simple*), 25
module
    mathslib, 26
    mathslib.algorithms, 21
    mathslib.fib, 20
    mathslib.linalg, 17
    mathslib.numtheory, 9
    mathslib.primes, 14
    mathslib.simple, 24

N

n_choose_r() (*in module mathslib.simple*), 24
number_to_base() (*in module mathslib.simple*), 24

O

overall_fraction() (*in module mathslib.numtheory*), 10

P

phi() (*in module mathslib.numtheory*), 10
phi_sieve() (*in module mathslib.numtheory*), 10
phi_sum() (*in module mathslib.numtheory*), 10
ppt() (*in module mathslib.numtheory*), 11
prime_factors() (*in module mathslib.primes*), 15
prime_sieve() (*in module mathslib.primes*), 14
primepi() (*in module mathslib.primes*), 15
primepi_sieve() (*in module mathslib.primes*), 15
prims_algorithm() (*in module mathslib.algorithms*), 21

S

solve() (*in module mathslib.linalg*), 17
sum_of_primes() (*in module mathslib.primes*), 16

T

tonelli_shanks() (*in module mathslib.numtheory*), 13

Z

zeckendorf_representation() (*in module mathslib.fib*), 20