
Pygorithm

Release 1.0

Omkar Pathak

Mar 02, 2018

1	Introduction	1
2	Quick Links	3
2.1	Binary Conversions	3
2.2	Data Structures	7
2.3	Dynamic Programming	22
2.4	Fibonacci	23
2.5	Geometry	24
2.6	Greedy Algorithms	52
2.7	Math	53
2.8	Path Finding Algorithms	55
2.9	Searching	57
2.10	Sorting	61
2.11	Strings	65
3	Quick Start Guide	69
4	Getting Started	71
	Python Module Index	73

CHAPTER 1

Introduction

Pygorithm: A fun way to learn algorithms on the go! Just import the module and start learning, it's that easy.

A Python module written in pure python and for purely educational purposes. Get the code, time complexities and much more by just importing the required algorithm. A good way to start learning Python programming. Learn implementation of all major algorithms in Python. No need to surf the internet to get the required code. Just install this module and get going.

- [Source Code](#)
- [Documentation](#)

2.1 Binary Conversions

A place for implementation of base conversions

2.1.1 Features

- To see all the available functions in a module, you can just type `help()` with the module name as argument. For example,

```
>>> from pygorithm import binary
>>> help(binary)
Help on package pygorithm.binary in pygorithm:

NAME
pygorithm.binary - Collection of binary conversions and algorithms

MODULE REFERENCE
https://docs.python.org/3.5/library/pygorithm.binary.html

The following documentation is automatically generated from the Python
source files. It may be incomplete, incorrect or include features that
are considered implementation detail and may vary between Python
implementations. When in doubt, consult the module reference at the
location listed above.

PACKAGE CONTENTS
ascii
base10
```

```
base16
base2
binary_utils

DATA
__all__ = ['ascii', 'base2', 'base10', 'base16']
```

2.1.2 ASCII Conversions

- Functions and their uses

ASCII

Conversions from ASCII to:

- base2
- base16

Author: Ian Doarn

`pygorithm.binary.ascii.to_base16` (*string*, *visualize=False*)

Convert ascii to hexadecimal :param string: string to convert :param visualize: Show process :param as_string: return value as string not array :return: hex representation of given string

`pygorithm.binary.ascii.to_base2` (*string*, *visualize=False*, *as_string=False*)

Convert ascii string to binary :param string: Ascii string :param visualize: Show process :param as_string: join strings with a space as one large value :return: array of binary numbers, or entire string

2.1.3 Base2 Coverions

- Functions and their uses

Binary: Base2

Conversions from base2 to:

- base10
- base16
- ASCII

Author: Ian Doarn

`pygorithm.binary.base2.to_ascii` (*b*, *visualize=False*)

Convert binary to ASCII

Parameters

- **b** – binary number or array
- **visualize** – Show process

Returns ASCII String

`pygorithm.binary.base2.to_base10` (*n*, *visualize=False*)

Convert given number to a list for every number do the following formula

$x * 2 + \text{number}$

repeat for each result! Example:

binary number = 100110

$0 \times 2 + 1 = 1$
 $1 \times 2 + 0 = 2$
 $2 \times 2 + 0 = 4$
 $4 \times 2 + 1 = 9$
 $9 \times 2 + 1 = 19$
 $19 \times 2 + 0 = 38$

Parameters

- **n** – binary number
- **visualize** – Show process

Returns decimal number

`pygorithm.binary.base2.to_base16(n, visualize=False)`

Convert binary numbers to hexadecimal numbers

Parameters

- **n** – binary number
- **visualize** – Visualise the process

Returns hexadecimal number

2.1.4 Base10 Coverions

- Functions and their uses

Binary: Base10

Conversions from base10 to:

- base2
- base16

Author: Ian Doarn

`pygorithm.binary.base10.to_base2(n, visualize=False)`

Divide each number by 2 using the % operator.

Reverse the resulting list of numbers and return the result

Parameters

- **n** – decimal number
- **visualize** – Show process

Returns binary number

`pygorithm.binary.base10.to_base16(n, visualize=False)`

Convert decimal number to hexadecimal

Divide the number by 16 and add the remainder to a list, round down the value after division and repeat till our value is 0

Reverse the results list, get each values respective hex value using HEX_VALUES map

Parameters

- **n** – decimal number
- **visualize** – Show process

Returns hexadecimal number

2.1.5 Base16 Conversions

- Functions and their uses

Binary: Base16

Conversions from base16 to:

- base2
- base10
- ASCII

Author: Ian Doarn

```
pygorithm.binary.base16.to_base2(h, visualize=False)
Convert hexadecimal to binary number
```

Parameters

- **h** – hexadecimal number
- **visualize** – Show process

Returns binary number

```
pygorithm.binary.base16.to_base10(h, visualize=False)
Convert hexadecimal number to decimal number
```

Send hex to a list and reverse. Evaluate each hex value via HEX_LETTER_VALUES map. Enumerate the list, using the equation: $value * 16 ^ index$

value is the hex char value: F -> 15 index is its position in the list: ['1', 'A', 'F'] F's index = 2

Continue this for each hex letter until we reach the end of the list, summing all evaluated values.

Parameters

- **h** – hexadecimal number
- **visualize** – Show process

Returns decimal number

```
pygorithm.binary.base16.to_ascii(h_array, visualize=False)
Convert base16 array to ASCII string
```

Input must be a list of strings: Example:

```
array = [ '74', '68', '65', '20', '71', '75', '69', '63', '6B', '20', '62', '72', '6F', '77', '6E', '20', '66', '6F',
          '78', '20', '6A', '75', '6D', '70', '73', '20', '6F', '76', '65', '72', '20', '74', '68', '65', '20', '6C', '61',
          '7A', '79', '20', '64', '6F', '67'
        ]
```

result -> the quick brown fox jumps over the lazy dog

Parameters

- **h_array** – hex value array
- **visualize** – Show process

Returns ASCII string

2.2 Data Structures

Implementing Data Structures purely in Python.

2.2.1 Quick Start Guide

```
# import the required data structure
from pygorithm.data_structures import stack

# create a stack with default stack size 10
myStack = stack.Stack()
myStack.push(2)

# print the contents of stack
print(myStack)
```

2.2.2 Features

- **Data Structures implementations available:**

- **Stack**

- * Stack (data_structures.stack.Stack)
- * Infix to Postfix conversion (data_structures.stack.InfixToPostfix)

- **Queue**

- * Queue (data_structures.queue.Queue)
- * Deque (data_structures.queue.Deque)

- **Linked List**

- * Singly Linked List (data_structures.linked_list.SinglyLinkedList)
- * Doubly Linked List (data_structures.linked_list.DoublyLinkedList)

- **Tree**

- * Binary Tree (data_structures.tree.BinaryTree)
- * Binary Search Tree (data_structures.tree.BinarySearchTree)

- **Graph**

- * Graph (data_structures.graph.Graph)
- * Topological Sort (data_structures.graph.TopologicalSort)
- * Check cycle in Directed Graph (data_structures.graph.CheckCycleDirectedGraph)
- * Check cycle in Undirected Graph (data_structures.graph.CheckCycleUndirectedGraph)

- **Heap**

- * Heap (data_structures.heap.Heap)

- **QuadTree**

- * QuadTree (data_structures.quadtree.QuadTree)

- Get the code used for any of the implementation

```
from pygorithm.data_structures.stack import Stack

myStack = Stack()
print(myStack.get_code())
```

- To see all the available functions in a module, you can just type `help()` with the module name as argument. For example,

```
>>> from pygorithm import data_structures
>>> help(data_structures)
Help on package pygorithm.data_structures in pygorithm:

NAME
pygorithm.data_structures

PACKAGE CONTENTS
graph
heap
linked_list
modules
queue
stack
tree
```

2.2.3 Stack

Author: OMKAR PATHAK Created On: 3rd August 2017

Stack

```
class pygorithm.data_structures.stack.Stack (limit=10)
    Stack object

    push (data)
        pushes an item into the stack returns -1 if the stack is empty

    pop ()
        pops the topmost item from the stack returns -1 if the stack is empty

    peek ()
        returns the topmost element of the stack returns -1 if the stack is empty

    is_empty ()
        checks if the stack is empty returns boolean value, True or False

    size ()
        returns the current size of the stack

    static get_code ()
        returns the code for current class
```

Infix to Postfix

```

class pygorithm.data_structures.stack.InfixToPostfix (expression=None,
                                                    stack=None)
    get the postfix of the given infix expression

    infix_to_postfix ()
        function to generate postfix expression from infix expression

    static get_code ()
        returns the code of the current class

```

2.2.4 Queue

Author: OMKAR PATHAK Created On: 3rd August 2017

Queue

```

class pygorithm.data_structures.queue.Queue (limit=10)
    Queue implementation

    size ()
        returns the current size of the queue

    is_empty ()
        checks if the queue is empty

    enqueue (data)
        inserts an item into the queue

    dequeue ()
        pops an item from the queue which was first inserted

    get_code ()
        Return source code for Queue class :return:

```

Deque

```

class pygorithm.data_structures.queue.Deque (limit=10)
    Deque implementation

    is_empty ()
        checks whether the deque is empty

    is_full ()
        checks whether the deque is full

    insert_rear (data)
        inserts an element at the rear end of the deque

    insert_front (data)
        inserts an element at the front end of the deque

    delete_rear ()
        deletes an element from the rear end of the deque

    delete_front ()
        deletes an element from the front end of the deque

```

```
static get_code ()  
    returns the code of the current class
```

2.2.5 Linked Lists

Author: OMKAR PATHAK Created On: 5th August 2017

Linked l_list and Node can be accommodated in separate classes for convenience

Node

```
class pygorithm.data_structures.linked_list.Node (data, next_node=None)  
    Node class for creating a node for linked list. Each node has its data and a pointer that points to next node in the  
    Linked l_list  
  
    static get_code ()  
        return the code for the current class
```

Singly Linked List

```
class pygorithm.data_structures.linked_list.SinglyLinkedList  
    Defining the head of the linked list  
  
    get_data ()  
        prints the elements in the linked list  
  
    insert_at_start (data)  
        insert an item at the beginning of the linked list  
  
    insert_after (next_node_data, data)  
        insert an item after an element in the linked list  
  
    insert_at_end (data)  
        insert an item at the end of the linked list  
  
    delete (data)  
        to delete specified element from the linked list  
  
    static get_code ()  
        return the code for the current class
```

Doubly Linked List

```
class pygorithm.data_structures.linked_list.DoublyLinkedList  
    DoublyLinkedList Class  
  
    get_data ()  
        prints the elements in the linked list  
  
    insert_at_start (data)  
        insert an element at the beginning of the linked list  
  
    insert_at_end (data)  
        insert an element at the end of the linked list  
  
    delete (data)  
        to delete specified element from the linked list
```

```

static get_code ()
    returns the code of the current class

```

2.2.6 Tree

Author: OMKAR PATHAK Created On: 9th August 2017

Node class to create a node for binary tree

Node

```

class pygorithm.data_structures.tree.Node (data=None)
    Node class for creating a node for tree. Each node has its data and a pointer that points to next node in the
    Linked List

    set_left (node)
        for setting the left child of the node

    set_right (node)
        for setting the right child of the node

    get_left ()
        for getting the left child of the node

    get_right ()
        for getting the right child of the node

    set_data (data)
        for setting the data of the node

    get_data ()
        for getting the data of the node

    static get_code ()
        returns the code of the current class

```

Binary Tree

```

class pygorithm.data_structures.tree.BinaryTree
    BinaryTree class to create a binary tree

    insert (data)
        insert data to root or create a root node

    inorder (root)
        in this we traverse first to the leftmost node, then print its data and then traverse for rightmost node :param
        root: Node object

    preorder (root)
        in this we first print the root node and then traverse towards leftmost node and then to the rightmost node
        :param root: Node object

    postorder (root)
        in this we first traverse to the leftmost node and then to the rightmost node and then print the data :param
        root: Node object

    number_of_nodes (root)
        counting number of nodes

```

```
static get_code ()  
    returns the code of the current class
```

Binary Search Tree Node

```
class pygorithm.data_structures.tree.BSTNode (data)  
    class for creating a node for binary Search tree  
  
    set_left (node)  
        for setting the left child of the node  
  
    set_right (node)  
        for setting the right child of the node  
  
    get_left ()  
        returns the left child of the current node  
  
    get_right ()  
        returns the right child of the current node  
  
    set_data (data)  
        for setting the data of a node  
  
    get_data ()  
        returns the data of the current node  
  
    insert (data)  
        For inserting the data in the Tree  
  
    static min_val_bst_node (bst_node)  
        for returning the node with the lowest value  
  
    delete (data)  
        For deleting the bst_node  
  
    find (data)  
        This function checks whether the specified data is in tree or not  
  
    inorder (root)  
        in this we first traverse to the leftmost node and then print the data and then move to the rightmost child  
        :param root: Node object  
  
    preorder (root)  
        in this we first print the root node and then traverse towards leftmost node and then to the rightmost node  
        :param root: Node object  
  
    postorder (root)  
        in this we first traverse to the leftmost node and then to the rightmost node and then print the data :param  
        root: Node object  
  
    static get_code ()  
        returns the code of current class
```

Binary Search Tree

```
class pygorithm.data_structures.tree.BinarySearchTree  
  
    insert (data)  
        inserts a node in the tree
```


delete (*data*)
deletes the node with the specified data from the tree

preorder ()
finding the preorder of the tree

inorder ()
finding the inorder of the tree

postorder ()
finding the postorder of the tree

number_of_nodes (*root*)
counting number of nodes

static get_code ()
returns the code of the current class

2.2.7 Graph

Author: OMKAR PATHAK Created On: 12th August 2017

Graph

class pygorithm.data_structures.graph.**Graph**
Graph object Creates the graph

print_graph ()
Prints the contents of the graph

add_edge (*from_vertex, to_vertex*)
Adds an edge in the graph

get_code ()
returns the code for the current class

Weighted Graph

class pygorithm.data_structures.graph.**WeightedGraph**
WeightedGraph object A graph with a numerical value (weight) on edges

add_edge (*u, v, weight*)

Parameters

- **u** – from vertex - type : integer
- **v** – to vertex - type : integer
- **weight** – weight of the edge - type : numeric

print_graph ()
Print the graph :return: None

kruskal_mst ()
Kruskal algorithm for finding the minimum spanning tree of a weighted graph. This version use a union-find data structure. More detailed info here: https://en.wikipedia.org/wiki/Kruskal%27s_algorithm Author: Michele De Vita <mik3dev@gmail.com>

```
static kruskal_time_complexity ()  
    Return time complexity of kruskal :return: string  
classmethod kruskal_code ()  
    Returns the code for current class
```

Weighted Undirected Graph

```
class pygorithm.data_structures.graph.WeightedUndirectedGraph  
    WeightedUndirectedGraph object A graph with a numerical value (weight) on edges, which is the same for both  
    directions in an undirected graph.  
  
    add_edge (u, v, weight)  
        Adds the specified edge to this graph. If the edge already exists, this will only modify the weight (not  
        create duplicates). :param u: from vertex :param v: to vertex :param weight: weight of the edge - type :  
        numeric  
  
    get_edge_weight (u, v)  
        Gets the weight between u and v if such an edge exists, or None if it does not. :param u: one edge :param  
        v: the other edge :return: numeric or None  
  
    remove_edge (edge, other_edge_or_none=None)  
        Removes the specified edge from the grid entirely or, if specified, the connection with one other edge.  
        Behavior is undefined if the connection does not exist. :param edge: the edge to remove :param  
        other_edge_or_none: an edge connected to edge or none  
  
    gridify (size, weight)  
        Constructs connections from a square grid starting at (0, 0) until (size-1, size-1) with connections between  
        adjacent and diagonal nodes. Diagonal nodes have a weight of  $\text{weight} \times \sqrt{2}$  :param size: the size of  
        the square grid to construct - type : integer :param weight: the weight between orthogonal nodes. - type:  
        numeric :return: None
```

Topological Sort

```
class pygorithm.data_structures.graph.TopologicalSort  
  
    topological_sort ()  
        function for sorting graph elements using topological sort  
  
    get_code ()  
        returns the code for the current class
```

Check Cycle in Directed Graph

```
class pygorithm.data_structures.graph.CheckCycleDirectedGraph  
    Class to check cycle in directed graph  
  
    print_graph ()  
        for printing the contents of the graph  
  
    add_edge (from_vertex, to_vertex)  
        function to add an edge in the graph  
  
    check_cycle ()  
        This function will return True if graph is cyclic else return False
```

```
static get_code ()
    returns the code for the current class
```

Check Cycle in Undirected Graph

```
class pygorithm.data_structures.graph.CheckCycleUndirectedGraph
    Class to check cycle in undirected graph

print_graph ()
    for printing the contents of the graph

add_edge (from_vertex, to_vertex)
    for adding the edge between two vertices

check_cycle ()
    This function will return True if graph is cyclic else return False

static get_code ()
    returns the code for the current class
```

2.2.8 Heap

Author: ALLSTON MICKEY Contributed: OMKAR PATHAK Created On: 11th August 2017

Heap

```
class pygorithm.data_structures.heap.Heap (limit=10)
    min-heap implementation as queue

static parent_idx (idx)
    retrieve the parent

static left_child_idx (idx)
    retrieve the left child

static right_child_idx (idx)
    retrieve the right child

insert (data)
    inserting an element in the heap

heapify_up ()
    Start at the end of the tree (last enqueued item).

    Compare the rear item to its parent, swap if the parent is larger than the child (min-heap property). Repeat
    until the min-heap property is met.

    Best Case:  $O(1)$ , item is inserted at correct position, no swaps needed Worst Case:  $O(\log n)$ , item needs to
    be swapped throughout all levels of tree

pop ()
    Removes the lowest value element (highest priority, at root) from the heap

favorite (parent)
    Determines which child has the highest priority by 3 cases
```

heapify_down()

Select the root and sift down until min-heap property is met.

While a favorite child exists, and that child is smaller than the parent, swap them (sift down).

Best Case: $O(1)$, item is inserted at correct position, no swaps needed Worst Case: $O(\log n)$, item needs to be swapped throughout all levels of tree

get_code()

returns the code for the current class

2.2.9 Trie

Author: MrDupin

Trie

```
class pygorithm.data_structures.trie.Trie
```

insert (*word*)

Inserts a word in the trie. Starting from the root, move down the trie following the path of characters in the word. If the nodes for the word characters end, add them. When the last char is added, mark it as a word-ending node.

search (*word*)

Searches for given word in trie. We want to find the last node for the word. If we can't, then it means the word is not in the trie.

find_words (*prefix*)

Find all words with the given prefix

find_final_node (*word*)

Returns the last node in given word. The process goes like this: Start from the root. For every char in word, go down one level. If we can't go down a level, then the word doesn't exist. If we do, and the current char is the last char of the word and the node we are currently at is a word, then we have found the given word.

build_word_list (*v*, *cWord*)

Recursively builds the list of words.

- *v*: Node to check
- *cWord* : The word built up to *v*

2.2.10 QuadTree

Author: Timothy Moore Created On: 31th August 2017

Defines a two-dimensional quadtree of arbitrary depth and bucket size.

QuadTreeEntity

```
class pygorithm.data_structures.quadtree.QuadTreeEntity (aabb)
```

This is the minimum information required for an object to be usable in a quadtree as an entity. Entities are the things that you are trying to compare in a quadtree.

Variables `aabb` – the axis-aligned bounding box of this entity

`__init__` (`aabb`)

Create a new quad tree entity with the specified `aabb`

Parameters `aabb` (`pygorithm.geometry.rect2.Rect2`) – axis-aligned bounding box

`__repr__` ()

Create an unambiguous representation of this entity.

Example:

```
from pygorithm.geometry import (vector2, rect2)
from pygorithm.data_structures import quadtree

_ent = quadtree.QuadTreeEntity(rect2.Rect2(5, 5))

# prints quadtreeentity(aabb=rect2(width=5, height=5, mincorner=vector2(x=0, y=0))
print(repr(_ent))
```

Returns unambiguous representation of this quad tree entity

Return type string

`__str__` ()

Create a human readable representation of this entity

Example:

```
from pygorithm.geometry import (vector2, rect2)
from pygorithm.data_structures import quadtree

_ent = quadtree.QuadTreeEntity(rect2.Rect2(5, 5))

# prints entity(at rect(5x5 at <0, 0>))
print(str(_ent))
```

Returns human readable representation of this entity

Return type string

`__weakref__`

list of weak references to the object (if defined)

QuadTree

`class` `pygorithm.data_structures.quadtree.QuadTree` (`bucket_size`, `max_depth`, `location`, `depth=0`, `entities=None`)

A quadtree is a sorting tool for two-dimensional space, most commonly used to reduce the number of required collision calculations in a two-dimensional scene. In this context, the scene is stepped without collision detection, then a quadtree is constructed from all of the boundaries

Caution: Just because a quad tree has split does not mean entities will be empty. Any entities which overlay any of the lines of the split will be included in the parent of the quadtree.

Tip: It is important to tweak bucket size and depth to the problem, but a common error is too small a bucket size. It is typically not reasonable to have a bucket size smaller than 16; A good starting point is 64, then modify as appropriate. Larger buckets reduce the overhead of the quad tree which could easily exceed the improvement from reduced collision checks. The max depth is typically just a sanity check since depth greater than 4 or 5 would either indicate a badly performing quadtree (too dense objects, use an r-tree or kd-tree) or a very large world (where an iterative quadtree implementation would be appropriate).

Variables

- **bucket_size** – maximum number objects per bucket (before `max_depth`)
- **max_depth** – maximum depth of the quadtree
- **depth** – the depth of this node (0 being the topmost)
- **location** – where this quad tree node is situated
- **entities** – the entities in this quad tree and in NO OTHER related quad tree
- **children** – either None or the 4 *QuadTree* children of this node

`__init__` (*bucket_size, max_depth, location, depth=0, entities=None*)
Initialize a new quad tree.

Warning: Passing entities to this quadtree will NOT cause it to split automatically! You must call `think()` for that. This allows for more predictable performance per line.

Parameters

- **bucket_size** (*int*) – the number of entities in this quadtree
- **max_depth** (*int*) – the maximum depth for automatic splitting
- **location** (*pygorithm.geometry.rect2.Rect2*) – where this quadtree is located
- **depth** (*int*) – the depth of this node
- **entities** (list of *QuadTreeEntity* or None for empty list) – the entities to initialize this quadtree with

think (*recursive=False*)
Call `split()` if appropriate

Split this quad tree if it has not split already and it has more entities than `bucket_size` and `depth` is less than `max_depth`.

If `recursive` is True, `think` is called on the `children` with `recursive` set to True after splitting.

Parameters recursive (*bool*) – if `think(True)` should be called on `children` (if there are any)

split ()
Split this quadtree.

Caution: A call to split will always split the tree or raise an error. Use `think()` if you want to ensure the quadtree is operating efficiently.

Caution: This function will not respect `bucket_size` or `max_depth`.

Raises `ValueError` – if `children` is not empty

get_quadrant (*entity*)

Calculate the quadrant that the specified entity belongs to.

Touching a line is considered overlapping a line. Touching is determined using `math.isclose()`

Quadrants are:

- -1: None (it overlaps 2 or more quadrants)
- 0: Top-left
- 1: Top-right
- 2: Bottom-right
- 3: Bottom-left

Caution: This function does not verify the entity is contained in this quadtree.

This operation takes $O(1)$ time.

Parameters *entity* (`QuadTreeEntity`) – the entity to place

Returns quadrant

Return type int

insert_and_think (*entity*)

Insert the entity into this or the appropriate child.

This also acts as thinking (recursively). Using `insert_and_think()` iteratively is slightly less efficient but has more predictable performance than initializing with a large number of entities then thinking is slightly faster but may hang. Both may exceed recursion depth if `max_depth` is too large.

Parameters *entity* (`QuadTreeEntity`) – the entity to insert

retrieve_collidables (*entity*, *predicate=None*)

Find all entities that could collide with the specified entity.

Warning: If *entity* is, itself, in the quadtree, it will be returned. The predicate may be used to prevent this using your preferred equality method.

The predicate takes 1 positional argument (the entity being considered) and returns *False* if the entity should never be returned, even if it might collide with the entity. It should return *True* otherwise.

Parameters

- **entity** (`QuadTreeEntity`) – the entity to find collidables for

- **predicate** (`types.FunctionType` or `None`) – the predicate

Returns potential collidables (never `None`)

Return type list of `QuadTreeEntity`

find_entities_per_depth()

Calculate the number of nodes and entities at each depth level in this quad tree. Only returns for depth levels at or equal to this node.

This is implemented iteratively. See `__str__()` for usage example.

Returns dict of depth level to number of entities

Return type dict int: int

find_nodes_per_depth()

Calculate the number of nodes at each depth level.

This is implemented iteratively. See `__str__()` for usage example.

Returns dict of depth level to number of nodes

Return type dict int: int

sum_entities (*entities_per_depth=None*)

Sum the number of entities in this quad tree and all lower quad trees.

If *entities_per_depth* is not `None`, that array is used to calculate the sum of entities rather than traversing the tree. Either way, this is implemented iteratively. See `__str__()` for usage example.

Parameters **entities_per_depth** (*dict int: (int, int)* or `None`) – the result of `find_entities_per_depth()`

Returns number of entities in this and child nodes

Return type int

calculate_avg_ents_per_leaf()

Calculate the average number of entities per leaf node on this and child quad trees.

In the ideal case, the average entities per leaf is equal to the bucket size, implying maximum efficiency. Note that, as always with averages, this might be misleading if this tree has reached its max depth.

This is implemented iteratively. See `__str__()` for usage example.

Returns average number of entities at each leaf node

Return type `numbers.Number`

calculate_weight_misplaced_ents (*sum_entities=None*)

Calculate a rating for misplaced entities.

A misplaced entity is one that is not on a leaf node. That weight is multiplied by 4*remaining maximum depth of that node, to indicate approximately how many additional calculations are required.

The result is then divided by the total number of entities on this node (either calculated using `sum_entities()` or provided) to get the approximate cost of the misplaced nodes in comparison with the placed nodes. A value greater than 1 implies a different tree type (such as r-tree or kd-tree) should probably be used.

This is implemented iteratively. See `__str__()` for usage example.

Parameters **sum_entities** (*int* or `None`) – the number of entities on this node

Returns weight of misplaced entities

Return type numbers.Number

`__repr__()`

Create an unambiguous representation of this quad tree.

This is implemented iteratively.

Example:

```
from pygorithm.geometry import (vector2, rect2)
from pygorithm.data_structures import quadtree

# create a tree with a up to 2 entities in a bucket that
# can have a depth of up to 5.
_tree = quadtree.QuadTree(1, 5, rect2.Rect2(100, 100))

# add a few entities to the tree
_tree.insert_and_think(quadtree.QuadTreeEntity(rect2.Rect2(2, 2, vector2.
↳Vector2(5, 5))))
_tree.insert_and_think(quadtree.QuadTreeEntity(rect2.Rect2(2, 2, vector2.
↳Vector2(95, 5))))

# prints quadtree(bucket_size=1, max_depth=5, location=rect2(width=100,
↳height=100, mincorner=vector2(x=0, y=0)), depth=0, entities=[],
↳children=[quadtree(bucket_size=1, max_depth=5, location=rect2(width=50.0,
↳height=50.0, mincorner=vector2(x=0, y=0)), depth=1,
↳entities=[quadtreeentity(aabb=rect2(width=2, height=2,
↳mincorner=vector2(x=5, y=5)))]), quadtree(bucket_size=1, max_
↳depth=5, location=rect2(width=50.0, height=50.0, mincorner=vector2(x=50.0,
↳y=0)), depth=1, entities=[quadtreeentity(aabb=rect2(width=2, height=2,
↳mincorner=vector2(x=95, y=5)))]), children=None), quadtree(bucket_size=1,
↳max_depth=5, location=rect2(width=50.0, height=50.0, mincorner=vector2(x=50.
↳0, y=50.0)), depth=1, entities=[], children=None), quadtree(bucket_size=1,
↳max_depth=5, location=rect2(width=50.0, height=50.0, mincorner=vector2(x=0,
↳y=50.0)), depth=1, entities=[], children=None)])
```

Returns unambiguous, recursive representation of this quad tree

Return type string

`__str__()`

Create a human-readable representation of this quad tree

Caution: Because of the complexity of quadtrees it takes a fair amount of calculation to produce something somewhat legible. All returned statistics have paired functions. This uses only iterative algorithms to calculate statistics.

Example:

```
from pygorithm.geometry import (vector2, rect2)
from pygorithm.data_structures import quadtree

# create a tree with a up to 2 entities in a bucket that
# can have a depth of up to 5.
_tree = quadtree.QuadTree(2, 5, rect2.Rect2(100, 100))

# add a few entities to the tree
```

```
_tree.insert_and_think(quadtree.QuadTreeEntity(rect2.Rect2(2, 2, vector2.  
↳Vector2(5, 5)))  
_tree.insert_and_think(quadtree.QuadTreeEntity(rect2.Rect2(2, 2, vector2.  
↳Vector2(95, 5)))  
  
# prints quadtree(at rect(100x100 at <0, 0>) with 0 entities here (2 in_  
↳total); (nodes, entities) per depth: [ 0: (1, 0), 1: (4, 2) ] (allowed max_  
↳depth: 5, actual: 1), avg ent/leaf: 0.5 (target 1), misplaced weight 0.0 (0_  
↳best, >1 bad)  
print(_tree)
```

Returns human-readable representation of this quad tree

Return type string

static get_code()

Get the code for the QuadTree class

Returns code for QuadTree

Return type string

__weakref__

list of weak references to the object (if defined)

2.3 Dynamic Programming

A place for implementation of greedy algorithms

2.3.1 Features

- To see all the available functions in a module, you can just type `help()` with the module name as argument. For example,

```
>>> from pygorithm import greedy_algorithm  
>>> help(greedy_algorithm)  
Help on package pygorithm.dynamic_programming in pygorithm:  
  
NAME  
pygorithm.dynamic_programming - Collection for dynamic programming algorithms  
  
PACKAGE CONTENTS  
binary_knapsack  
lis  
  
DATA  
__all__ = ['binary_knapsack', 'lis']
```

2.3.2 Binary (0/1) Knapsack

- Functions and their uses

Author: Omkar Pathak Created At: 25th August 2017

`pygorithm.dynamic_programming.binary_knapsack.knapsack(w, value, weight)`

The knapsack problem or rucksack problem is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items.

Parameters

- **w** – maximum weight capacity
- **value** – an array of values of items in the knapsack
- **weight** – an array of weights of items in the knapsack

`pygorithm.dynamic_programming.binary_knapsack.get_code()`

returns the code for the knapsack function

2.3.3 Longest Increasing Subsequence

- Functions and their uses

Author: Omkar Pathak Created At: 25th August 2017

`pygorithm.dynamic_programming.lis.longest_increasing_subsequence(_list)`

The Longest Increasing Subsequence (LIS) problem is to find the length of the longest subsequence of a given sequence such that all elements of the subsequence are sorted in increasing order. For example, the length of LIS for [10, 22, 9, 33, 21, 50, 41, 60, 80] is 6 and LIS is [10, 22, 33, 50, 60, 80].

Parameters `_list` – an array of elements

Returns returns a tuple of maximum length of lis and an array of the elements of lis

`pygorithm.dynamic_programming.lis.get_code()`

returns the code for the longest_increasing_subsequence function

2.4 Fibonacci

Learning fibonacci implementations in few ways!

2.4.1 Quick Start Guide

```
from pygorithm.fibonacci import recursion as fib_recursion

sequence = fib_recursion.get_sequence(10)
print(sequence) # [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

2.4.2 Features

- **Fibonacci implementations available:**
 - Generator
 - Golden ratio
 - Memorization (which saves some recursions to avoid computation of same series again and again)

– Recursion

- Get the code used for any of the implementation

```
from pygorithm.fibonacci import recursion as fib_recursion

code = fib_recursion.get_code()
print(code)
```

- To see all the available functions in a module, you can just type `help()` with the module name as argument. For example,

```
>>> from pygorithm import fibonacci
>>> help(fibonacci)
Help on package pygorithm.fibonacci in pygorithm:

NAME
pygorithm.fibonacci - Collection of fibonacci methods and functions

PACKAGE CONTENTS
generator
goldenratio
memoization
modules
recursion
```

2.4.3 Implementations API

- Functions and their uses

get_sequence (*number*)

- **number** : arbitrary integer, that need to be calculated in Fibonacci number type
- **Return Value** : return Fibonacci value by specified number as integer

get_code ()

- **Return Value** : returns the code for the `get_sequence()` function

2.5 Geometry

Some geometrical shapes and operations

2.5.1 Quick Start Guide

```
# import the required shapes and structures
from pygorithm.geometry import polygon2
from pygorithm.geometry import vector2

# create a regular polygon
poly1 = polygon2.Polygon2.from_regular(5, 5)

# create a polygon from tuple (x, y) - note that the polygon must be convex
# and the points must be clockwise
```

```

poly2 = polygon2.Polygon2(points=[ (0, 0), (1, 0), (1, 1), (0, 1) ])

# create a polygon from vector2s.
poly3 = polygon2.Polygon2(points=[ vector2.Vector2(0, 0),
                                   vector2.Vector2(1, 1),
                                   vector2.Vector2(2, 0) ])

# create a polygon by rotating another polygon
poly4 = poly3.rotate(0.2)
poly5 = poly3.rotate(degrees = 30)

# check intersection
intrs, mtv = polygon2.Polygon2.find_intersection(poly1, poly2, (0, 0), (1, 0))

if intrs:
    mtv_dist = mtv[0]
    mtv_vec = mtv[1]
    print('They intersect. The best way to push poly1 is {} units along {}'.
          ↪format(mtv_dist, mtv_vec))
else:
    print('No intersection')

```

2.5.2 Features

- **Structures available:**
 - Vector2 (vector2)
 - Line2 (line2)
 - AxisAlignedLine (axisall)
- **Shapes available:**
 - Concave Polygons (polygon2)
 - Rectangles (rect2)
- **Algorithms available:**
 - Separating Axis Theorem (polygon2)
 - Broad-phase (rect2)
 - Extrapolated intersection (extrapolated_intersection)

2.5.3 Vector2

class pygorithm.geometry.vector2.**Vector2**(*args, **kwargs)

Define a simple two-dimensional, mutable vector.

Important: Equality is not overridden on vectors, because it is expected that vectors will be used mutably by directly modifying x and y. However, all functions on vectors are immutable (they return a copy)

Variables

- **x** (`numbers.Number`) – The first component of this vector.
- **y** (`numbers.Number`) – The second component of this vector.

`__init__` (**args, **kwargs*)

Create a new `Vector2` from the two components.

Accepts a pair of unnamed parameters, a pair of named `x`, `y` parameters, another `Vector2`, or a tuple with 2 numerics. Examples of each:

```
from pygorithm.geometry import vector2

# A pair of unnamed parameters
vec1 = vector2.Vector2(0, 5)

# A pair of named parameters
vec2 = vector2.Vector2(x = 0, y = 5)

# Another vector2
vec3 = vector2.Vector2(vec2)

# A tuple with two numerics
vec4 = vector2.Vector2( (0, 5) )
```

Parameters

- **args** – unnamed arguments (purpose guessed by order)
- **kwargs** – named arguments (purpose known by name)

`__add__` (*other*)

Adds the two vectors component wise.

Example:

```
from pygorithm.geometry import vector2

vec1 = vector2.Vector2(0, 3)
vec2 = vector2.Vector2(2, 4)

vec3 = vec1 + vec2

# prints <2, 7>
print(vec3)
```

Parameters *other* (`pygorithm.geometry.vector2.Vector2`) – the vector to add to this one

Returns a new vector that is the sum of self and other

Return type `pygorithm.geometry.vector2.Vector2`

`__sub__` (*other*)

Subtract the two vectors component wise.

Example:

```
from pygorithm.geometry import vector2
```

```

vec1 = vector2.Vector2(5, 5)
vec2 = vector2.Vector2(2, 3)

vec3 = vec1 - vec2
vec4 = vec2 - vec1

# prints <3, 2>
print(vec3)

# prints <2, 3>
print(vec4)

```

Parameters other (*pygorithm.geometry.vector2.Vector2*) – the vector to subtract from this one

Returns a new vector two that is the difference of self and other

Return type *pygorithm.geometry.vector2.Vector2*

__mul__ (*scale_factor*)

Scale the vector by the specified factor.

Caution: This will never perform a dot product. If *scale_factor* is a *Vector2*, an exception is thrown.

Example:

```

from pygorithm.geometry import vector2

vec1 = vector2.Vector2(4, 8)

vec2 = vec1 * 0.5

# prints <2, 4>
print(vec2)

```

Param *scale_factor* the amount to scale this vector by

Returns a new vector that is self scaled by *scale_factor*

Return type *pygorithm.geometry.vector2.Vector2*

Raises TypeError – if *scale_factor* is a *Vector2*

__rmul__ (*scale_factor*)

Scale the vector by the specified factor.

Caution: This will never perform a dot product. If *scale_factor* is a *Vector2*, an exception is thrown.

Example:

```

from pygorithm.geometry import vector2

vec1 = vector2.Vector2(4, 8)

```

```
vec2 = 2 * vec1

# prints <8, 16>
print (vec2)
```

Param `scale_factor` the amount to scale this vector by

Returns a new vector that is self scaled by `scale_factor`

Return type `pygorithm.geometry.vector2.Vector2`

Raises **TypeError** – if `scale_factor` is a `Vector2`

__repr__ ()

Create an unambiguous representation of this vector

Example:

```
from pygorithm.geometry import vector2

vec = vector2.Vector2(3, 5)

# prints vector2(x=3, y=5)
print (repr(vec))
```

Returns an unambiguous representation of this vector

Return type string

__str__ ()

Create a human-readable representation of this vector.

Rounds to 3 decimal places if there are more.

Example:

```
from pygorithm.geometry import vector2

vec = vector2.Vector2(7, 11)

# prints <7, 11>
print (str(vec))

# also prints <7, 11>
print (vec)
```

Returns a human-readable representation of this vector

Return type string

__weakref__

list of weak references to the object (if defined)

dot (*other*)

Calculate the dot product between this vector and other.

The dot product of two vectors is calculated as so:


```

Let v1 be a vector such that v1 = <v1_x, v1_y>
Let v2 be a vector such that v2 = <v2_x, v2_y>

v1 . v2 = v1_x * v2_x + v1_y * v2_y

```

Example:

```

from pygorithm.geometry import vector2

vec1 = vector2.Vector2(3, 5)
vec2 = vector2.Vector2(7, 11)

dot_12 = vec1.dot(vec2)

# prints 76
print(dot_12)

```

Parameters *other* (`pygorithm.geometry.vector2.Vector2`) – the other vector

Returns the dot product of self and other

Return type `numbers.Number`

cross (*other*)

Calculate the z-component of the cross product between this vector and other.

The cross product of two vectors is calculated as so:

```

Let v1 be a vector such that v1 = <v1_x, v1_y>
Let v2 be a vector such that v2 = <v2_x, v2_y>

v1 x v2 = v1.x * v2.y - v1.y * v2.x

```

Caution: This is the special case of a cross product in 2 dimensions returning 1 value. This is really a vector in the z direction!

rotate (**args, **kwargs*)

The named argument “degrees” or “radians” may be passed in to rotate this vector by the specified amount in degrees (or radians), respectively. If both are omitted, the first unnamed argument is assumed to be the amount to rotate in radians.

Additionally, the named argument “about” may be passed in to specify about what the vector should be rotated. If omitted then the first unconsumed unnamed argument is assumed to be the vector. If there are no unconsumed unnamed arguments then the origin is assumed.

Examples:

```

from pygorithm.geometry import vector2
import math

vec1 = vector2.Vector2(1, 0)

vec2 = vec1.rotate(math.pi * 0.25)

# prints <0.707, 0.707>
print(vec2)

```

```
vec3 = vec1.rotate(degrees = 45)

# prints <0.707, 0.707>
print (vec3)

# The following operations are all identical

vec4 = vec1.rotate(math.pi, vector2.Vector2(1, 1))
vec5 = vec1.rotate(radians = math.pi, about = vector2.Vector2(1, 1))
vec6 = vec1.rotate(degrees = 180, about = vector2.Vector2(1, 1))
vec7 = vec1.rotate(vector2.Vector2(1, 1), degrees = 180)

# prints <1, 2>
print (vec4)
```

Parameters

- **args** – the unnamed arguments (purpose guessed by position)
- **kwargs** – the named arguments (purpose known by name)

Returns the new vector formed by rotating this vector

Return type *pygorithm.geometry.vector2.Vector2*

normalize()

Create the normalized version of this vector

The normalized version will go in the same direction but will have magnitude of 1.

Note: This will never return self, even if this vector is already normalized.

Example:

```
from pygorithm.geometry import vector2

vec1 = vector2.Vector2(2, 0)

vec2 = vec1.normalize()

# prints <1, 0>
print (vec2)
```

Returns a new normalized version of this vector

Return type *pygorithm.geometry.vector2.Vector2*

magnitude_squared()

Calculate the square of the magnitude of this vector.

Example:

```
from pygorithm.geometry import vector2

vec1 = vector2.Vector2(5, 12)
magn_sq = vec1.magnitude_squared()
```

```
# prints 169 (13^2)
print (magn_sq)
```

Returns square of the magnitude of this vector

Return type `numbers.Number`

magnitude()

Calculate the magnitude of this vector

Note: It is substantially faster to operate on magnitude squared where possible.

Example:

```
from pygorithm.geometry import vector2

vec1 = vector2.Vector2(3, 4)
magn = vec1.magnitude()

# prints 5
print (magn)
```

Returns magnitude of this vector

Return type `numbers.Number`

2.5.4 Line2

class `pygorithm.geometry.line2.Line2` (*start*, *end*)

Define a two-dimensional directed line segment defined by two points. This class is mostly used as a way to cache information that is regularly required when working on geometrical problems.

Caution: Lines should be used as if they were completely immutable to ensure correctness. All attributes of `Line2` can be reconstructed from the two points, and thus cannot be changed on their own and must be recalculated if there were any changes to *start* or *end*.

Tip: To prevent unnecessary recalculations, many functions on lines accept an ‘offset’ argument, which is used to perform calculations on lines that are simply shifts of other lines.

Note: The minimum x is guaranteed to be on either (or both) of the start and end. However, minimum x and minimum y might not come from the same point. The same is true for the maximum x and maximum y.

Variables

- **start** (`pygorithm.geometry.vector2.Vector2`) – the start of this line
- **end** (`pygorithm.geometry.vector2.Vector2`) – the end of this line

`__init__` (*start, end*)

Create a new line from start to end.

Parameters

- **start** (*pygorithm.geometry.vector2.Vector2*) – the start point
- **end** (*pygorithm.geometry.vector2.Vector2*) – the end point

Raises `ValueError` – if start and end are at the same point

delta

Get the vector from start to end, lazily initialized.

Returns delta from start to end

Return type *pygorithm.geometry.vector2.Vector2*

axis

Get the normalized delta vector, lazily initialized

Returns normalized delta

Return type *pygorithm.geometry.vector2.Vector2*

normal

Get normalized normal vector to axis, lazily initialized.

Get the normalized normal vector such that the normal vector is 90 degrees counter-clockwise from the axis.

Returns normalized normal to axis

Return type *pygorithm.geometry.vector2.Vector2*

magnitude_squared

Get the square of the magnitude of delta, lazily initialized.

Returns square of magnitude of delta

Return type `numbers.Number`

magnitude

Get the magnitude of delta, lazily initialized.

Note: It is substantially faster to operate on squared magnitude, where possible.

Returns magnitude of delta

Return type `numbers.Number`

min_x

Get the minimum x that this line contains, lazily initialized.

Returns minimum x this line contains

Return type `numbers.Number`

min_y

Get the minimum y that this line contains, lazily initialized.

Returns minimum x this line contains

Return type `numbers.Number`

max_x

Get the maximum x that this line contains, lazily initialized.

Returns maximum x this line contains

Return type `numbers.Number`

max_y

Get the maximum y that this line contains, lazily initialized.

Returns maximum x this line contains

Return type `numbers.Number`

slope

Get the slope of this line, lazily initialized.

Caution: The slope may be 0 (horizontal line) or positive or negative infinity (vertical lines). It may be necessary to handle these lines separately, typically through checking the *horizontal* and *vertical* properties.

Returns the slope of this line (rise over run).

Return type `numbers.Number`

y_intercept

Get the y-intercept of this line, lazily initialized.

This does not take into account any offset of the line and may return `None` if this is a vertical line.

Caution: This function will return a y-intercept for non-vertical line segments that do not reach $x=0$.

Caution: The y-intercept will change based on the offset in a somewhat complex manner. `calculate_y_intercept()` accepts an offset parameter.

Returns the y-intercept of this line when unshifted

Return type `numbers.Number` or `None`

horizontal

Get if this line is horizontal, lazily initialized.

A line is horizontal if it has a slope of 0. This also means that `start.y == end.y`

Returns if this line is horizontal

Return type `bool`

vertical

Get if this line is vertical, lazily initialized.

A line is vertical if it has a slope of `+inf` or `-inf`. This also means that `start.x == end.x`.

Returns if this line is vertical

Return type `bool`

__repr__()

Get an unambiguous representation of this line

Example:

```
from pygorithm.geometry import (vector2, line2)

vec1 = vector2.Vector2(1, 1)
vec2 = vector2.Vector2(3, 4)

line = line2.Line2(vec1, vec2)

# prints line2(start=vector2(x=1, y=1), end=vector2(x=3, y=4))
print(repr(line))
```

Returns unambiguous representation of this line

Return type string

__str__()

Get a human-readable representation of this line

Example:

```
from pygorithm.geometry import (vector2, line2)

vec1 = vector2.Vector2(1, 1)
vec2 = vector2.Vector2(3, 4)

line = line2.Line2(vec1, vec2)

# prints <1, 1> -> <3, 4>
print(str(line))

# same as above
print(line)
```

Returns human-readable representation of this line

Return type string

calculate_y_intercept(offset)

Calculate the y-intercept of this line when it is at the specified offset.

If the offset is None this is exactly equivalent to y_intercept

Parameters **offset** (*pygorithm.geometry.vector2.Vector2* or None) – the offset of this line for this calculations

Returns the y-intercept of this line when at offset

Return type numbers.Number

static are_parallel(line1, line2)

Determine if the two lines are parallel.

Two lines are parallel if they have the same or opposite slopes.

Parameters

- **line1** (*pygorithm.geometry.line2.Line2*) – the first line

- **line2** (*pygorithm.geometry.line2.Line2*) – the second line

Returns if the lines are parallel

Return type bool

static contains_point (*line, point, offset=None*)

Determine if the line contains the specified point.

Optionally, specify an offset for the line. Being on the line is determined using *math.isclose*.

Parameters

- **line** (*pygorithm.geometry.line2.Line2*) – the line
- **point** (*pygorithm.geometry.vector2.Vector2*) – the point
- **offset** (*pygorithm.geometry.vector2.Vector2* or None) – the offset of the line or None for the origin

Returns if the point is on the line

Return type bool

static find_intersection (*line1, line2, offset1=None, offset2=None*)

Find the intersection between the two lines.

The lines may optionally be offset by a fixed amount. This will incur a minor performance penalty which is less than that of recreating new lines.

Two lines are considered touching if they only share exactly one point and that point is an edge of one of the lines.

If two lines are parallel, their intersection could be a line.

Tip: This will never return True, True

Parameters

- **line1** (*pygorithm.geometry.line2.Line2*) – the first line
- **line2** (*pygorithm.geometry.line2.Line2*) – the second line
- **offset1** (*pygorithm.geometry.vector2.Vector2* or None) – the offset of line 1
- **offset2** (*pygorithm.geometry.vector2.Vector2* or None) – the offset of line 2

Returns (touching, overlapping, intersection_location)

Return type (bool, bool, *pygorithm.geometry.line2.Line2* or *pygorithm.geometry.vector2.Vector2* or None)

__weakref__

list of weak references to the object (if defined)

2.5.5 Axis-Aligned Line

class *pygorithm.geometry.axisall.AxisAlignedLine* (*axis, point1, point2*)

Define an axis aligned line.

This class provides functions related to axis aligned lines as well as acting as a convenient container for them. In this context, an axis aligned line is a two-dimensional line that is defined by an axis and length on that axis, rather than two points. When working with two lines defined as such that have the same axis, many calculations are simplified.

Note: Though it requires the same amount of memory as a simple representation of a 2 dimensional line (4 numerics), it cannot describe all types of lines. All lines that can be defined this way intersect (0, 0).

Note: *min* and *max* are referring to nearness to negative and positive infinity, respectively. The absolute value of *min* may be larger than that of *max*.

Note: AxisAlignedLines are an intermediary operation, so offsets should be baked into them.

Variables

- **axis** (*pygorithm.geometry.vector2.Vector2*) – the axis this line is on
- **min** (*numbers.Number*) – the point closest to negative infinity
- **max** (*numbers.Number*) – the point closest to positive infinity

__init__ (*axis, point1, point2*)

Construct an axis aligned line with the appropriate min and max.

Parameters

- **axis** (*pygorithm.geometry.vector2.Vector2*) – axis this line is on (for book-keeping only, may be None)
- **point1** (*numbers.Number*) – one point on this line
- **point2** (*numbers.Number*) – a different point on this line

static intersects (*line1, line2*)

Determine if the two lines intersect

Determine if the two lines are touching, if they are overlapping, or if they are disjoint. Lines are touching if they share only one end point, whereas they are overlapping if they share infinitely many points.

Note: It is rarely faster to check intersection before finding intersection if you will need the minimum translation vector, since they do mostly the same operations.

Tip: This will never return `True, True`

Parameters

- **line1** (*pygorithm.geometry.axisall.AxisAlignedLine*) – the first line
- **line2** (*pygorithm.geometry.axisall.AxisAlignedLine*) – the second line

Returns (touching, overlapping)

Return type (bool, bool)

static find_intersection (*line1*, *line2*)

Calculate the MTV between line1 and line2 to move line1

Determine if the two lines are touching and/or overlapping and then returns the minimum translation vector to move line 1 along axis. If the result is negative, it means line 1 should be moved in the opposite direction of the axis by the magnitude of the result.

Returns *true*, (*None*, *touch_point_numeric*, *touch_point_numeric*) if the lines are touching and not overlapping.

Note: Ensure your program correctly handles *true*, (*None*, *numeric*, *numeric*)

Parameters

- **line1** (*pygorithm.geometry.axisall.AxisAlignedLine*) – the first line
- **line2** (*pygorithm.geometry.axisall.AxisAlignedLine*) – the second line

Returns (touching, (mtv against 1, intersection min, intersection max))

Return type (bool, (numbers.Number or None, numbers.Number, numbers.Number) or None)

static contains_point (*line*, *point*)

Determine if the line contains the specified point.

The point must be defined the same way as min and max.

Tip: It is not possible for both returned booleans to be *True*.

Parameters

- **line** (*pygorithm.geometry.axisall.AxisAlignedLine*) – the line
- **point** (numbers.Number) – the point

Returns (if the point is an edge of the line, if the point is contained by the line)

Return type (bool, bool)

__repr__ ()

Create an unambiguous representation of this axis aligned line.

Example:

```
from pygorithm.geometry import axisall

aal = axisall.AxisAlignedLine(None, 3, 5)

# prints AxisAlignedLine(axis=None, min=3, max=5)
print(repr(aal))
```

Returns un-ambiguous representation of this line

Return type string

`__str__()`

Create a human-readable representation of this axis aligned line.

Example:

```
from pygorithm.geometry import axisall

aal = axisall.AxisAlignedLine(None, 0.7071234, 0.7071234)

# prints axisall(along None from 0.707 to 0.707)
print(aal)
```

Returns human-readable representation of this line

Return type string

`__weakref__`

list of weak references to the object (if defined)

2.5.6 Concave Polygon

class `pygorithm.geometry.polygon2.Polygon2` (*points*, *suppress_errors=False*)

Define a concave polygon defined by a list of points such that each adjacent pair of points form a line, the line from the last point to the first point form a line, and the lines formed from the smaller index to the larger index will walk clockwise around the polygon.

Note: Polygons should be used as if they were completely immutable to ensure correctness. All attributes of `Polygon2` can be reconstructed from the points array, and thus cannot be changed on their own and must be recalculated if there were any changes to *points*.

Note: To reduce unnecessary recalculations, Polygons notably do not have an easily modifiable position. However, where relevant, the class methods will accept offsets to the polygons. In all of these cases the offset may be `None` for a minor performance improvement.

Note: Unfortunately, operations on rotated polygons require recalculating the polygon based on its rotated points. This should be avoided unless necessary through the use of Axis-Aligned Bounding Boxes and similar tools.

Caution: The length of *normals* is not necessarily the same as *points* or *lines*. It is only guaranteed to have no two vectors that are the same or opposite directions, and contain either the vector in the same direction or opposite direction of the normal vector for every line in the polygon.

Variables

- **points** (list of `pygorithm.geometry.vector2.Vector2`) – the ordered list of points on this polygon
- **lines** (list of `pygorithm.geometry.line2.Line2`) – the ordered list of lines on this polygon

- **normals** (list of `pygorithm.geometry.vector2.Vector2`) – the unordered list of unique normals on this polygon
- **center** (`pygorithm.geometry.vector2.Vector2`) – the center of this polygon when unshifted.

`__init__` (`points`, `suppress_errors=False`)
Create a new polygon from the set of points

Caution: A significant amount of calculation is performed when creating a polygon. These should be reused whenever possible. This cost can be alleviated somewhat by suppressing certain expensive sanity checks, but the polygon can behave very unexpectedly (and potentially without explicit errors) if the errors are suppressed.

The center of the polygon is calculated as the average of the points.

The lines of the polygon are constructed using `line2`.

The normals of the lines are calculated using `line2`.

A simple linear search is done to check for repeated points.

The area is calculated to check for clockwise order using the *Shoelace Formula* <https://en.wikipedia.org/wiki/Shoelace_formula>

The polygon is proven to be convex by ensuring the cross product of the line from the point to previous point and point to next point is positive or 0, for all points.

Parameters

- **points** (list of `pygorithm.geometry.vector2.Vector2` or list of (`numbers.Number`, `numbers.Number`)) – the ordered set of points on this polygon
- **suppress_errors** (`bool`) – True to not do somewhat expensive sanity checks

Raises

- **ValueError** – if there are less than 3 points (not suppressable)
- **ValueError** – if there are any repeated points (suppressable)
- **ValueError** – if the points are not clockwise oriented (suppressable)
- **ValueError** – if the polygon is not convex (suppressable)

classmethod from_regular (`sides`, `length`, `start_rads=None`, `start_degs=None`, `center=None`)
Create a new regular polygon.

Hint: If no rotation is specified there is always a point at (`length`, `0`)

If no center is specified, the center will be calculated such that all the vertexes positive and the bounding box includes (0, 0). This operation requires O(n) time (where n is the number of sides)

May specify the angle of the first point. For example, if the coordinate system is x to the right and y upward, then if the starting offset is 0 then the first point will be at the right and the next point counter-clockwise.

This would make for the regular quad (`sides=4`) to look like a diamond. To make the bottom side a square, the whole polygon needs to be rotated 45 degrees, like so:

```

from pygorithm.geometry import (vector2, polygon2)
import math

# This is a diamond shape (rotated square) (0 degree rotation assumed)
diamond = polygon2.Polygon2.from_regular(4, 1)

# This is a flat square
square = polygon2.Polygon2.from_regular(4, 1, start_degs = 45)

# Creating a flat square with radians
square2 = polygon2.Polygon2.from_regular(4, 1, math.pi / 4)

```

Uses the *definition of a regular polygon* <https://en.wikipedia.org/wiki/Regular_polygon> to find the angle between each vertex in the polygon. Then converts the side length to circumradius using the formula explained *here* <<http://mathworld.wolfram.com/RegularPolygon.html>>

Finally, each vertex is found using $\langle \text{radius} * \cos(\text{angle}), \text{radius} * \sin(\text{angle}) \rangle$

If the center is not specified, the minimum of the bounding box of the polygon is calculated while the vertices are being found, and the inverse of that value is offset to the rest of the points in the polygon.

Parameters

- **sides** (`numbers.Number`) – the number of sides in the polygon
- **length** (`numbers.Number`) – the length of any side of the polygon
- **start_rads** (`numbers.Number` or `None`) – the starting radians or `None`
- **start_degs** (`numbers.Number` or `None`) – the starting degrees or `None`
- **center** (`pygorithm.geometry.vector2.Vector2`) – the center of the polygon

Returns the new regular polygon

Return type `pygorithm.geometry.polygon2.Polygon2`

Raises

- **ValueError** – if `sides < 3` or `length <= 0`
- **ValueError** – if `start_rads` is not `None` and `start_degs` is not `None`

classmethod from_rotated (`original`, `rotation`, `rotation_degrees=None`)

Create a regular polygon that is a rotation of a different polygon.

The rotation must be in radians, or null and `rotation_degrees` must be specified. Positive rotations are clockwise.

Examples:

```

from pygorithm.goemetry import (vector2, polygon2)
import math

poly = polygon2.Polygon2.from_regular(4, 1)

# the following are equivalent (within rounding)
rotated1 = polygon2.Polygon2.from_rotated(poly, math.pi / 4)
rotated2 = polygon2.Polygon2.from_rotated(poly, None, 45)

```

Uses the *2-d rotation matrix* <https://en.wikipedia.org/wiki/Rotation_matrix> to rotate each point.

Parameters

- **original** (*pygorithm.geometry.polygon2.Polygon2*) – the polygon to rotate
- **rotation** (*numbers.Number*) – the rotation in radians or None
- **rotation_degrees** (*numbers.Number*) – the rotation in degrees or None

Returns the rotated polygon

Return type *pygorithm.geometry.polygon2.Polygon2*

Raises

- **ValueError** – if rotation is not None and rotation_degrees is not None
- **ValueError** – if rotation is None and rotation_degrees is None

area

Get the area of this polygon. Lazily initialized.

Uses the *Shoelace Formula* <https://en.wikipedia.org/wiki/Shoelace_formula> to calculate the signed area, allowing this to also test for correct polygon orientation.

Returns area of this polygon

Return type *numbers.Number*

Raises **ValueError** – if the polygon is not in clockwise order

static project_onto_axis (*polygon, offset, axis*)

Find the projection of the polygon along the axis.

Uses the *dot product* <https://en.wikipedia.org/wiki/Dot_product> of each point on the polygon to project those points onto the axis, and then finds the extremes of the projection.

Parameters

- **polygon** (*pygorithm.geometry.polygon2.Polygon2*) – the polygon to project
- **offset** (*pygorithm.geometry.vector2.Vector2*) – the offset of the polygon
- **axis** (*pygorithm.geometry.vector2.Vector2*) – the axis to project onto

Returns the projection of the polygon along the axis

Return type *pygorithm.geometry.axisall.AxisAlignedLine*

static contains_point (*polygon, offset, point*)

Determine if the polygon at offset contains point.

Distinguish between points that are on the edge of the polygon and points that are completely contained by the polygon.

Tip: This can never return True, True

This finds the cross product of this point and the two points comprising every line on this polygon. If any are 0, this is an edge. Otherwise, they must all be negative (when traversed clockwise).

Parameters

- **polygon** (*pygorithm.geometry.polygon2.Polygon2*) – the polygon

- **offset** (*pygorithm.geometry.vector2.Vector2* or None) – the offset of the polygon
- **point** (*pygorithm.geometry.vector2.Vector2*) – the point to check

Returns on edge, contained

Return type bool, bool

static find_intersection (*poly1, poly2, offset1, offset2, find_mtv=True*)

Find if the polygons are intersecting and how to resolve it.

Distinguish between polygons that are sharing 1 point or a single line (touching) as opposed to polygons that are sharing a 2-dimensional amount of space.

The resulting MTV should be applied to the first polygon (or its offset), or its negation can be applied to the second polygon (or its offset).

The MTV will be non-null if overlapping is True and find_mtv is True.

Note: There is only a minor performance improvement from setting find_mtv to False. It is rarely an improvement to first check without finding mtv and then to find the mtv.

Caution: The first value in the mtv could be negative (used to inverse the direction of the axis)

This uses the ‘Separating Axis Theorem <<http://www.dyn4j.org/2010/01/sat/>> to calculate intersection.

Parameters

- **poly1** (*pygorithm.geometry.polygon2.Polygon2*) – the first polygon
- **poly2** (*pygorithm.geometry.polygon2.Polygon2*) – the second polygon
- **offset1** (*pygorithm.geometry.vector2.Vector2* or None) – the offset of the first polygon
- **offset2** (*pygorithm.geometry.vector2.Vector2* or None) – the offset of the second polygon
- **find_mtv** (*bool*) – if False, the mtv is always None and there is a small performance improvement

Returns (touching, overlapping, (mtv distance, mtv axis))

Return type (bool, bool, (numbers.Number, *pygorithm.geometry.vector2.Vector2*) or None)

__repr__ ()

Creates an unambiguous representation of this polygon, only showing the list of points.

Returns unambiguous representation of this polygon

Return type string

__str__ ()

Creates a human-readable representation of this polygon and includes a link to visualize it

Returns human-readable representation

Return type string

`__weakref__`

list of weak references to the object (if defined)

2.5.7 Axis-Aligned Rectangle

class `pygorithm.geometry.rect2.Rect2` (*width, height, mincorner=None*)

A rectangle. Uses SAT collision against polygons and broad-phase collision against other rectangles.

Rectangles are fast to construct and have very fast rectangle-rectangle collision detection.

Rect2 is designed to have almost exactly the opposite performance characteristics as `Polygon2` when doing collision against `Polygon2`s: Fast to construct and complex on first call with many operations incurring expensive recalculations.

Caution: Collision detection against a polygon will cause initialization of the polygon representation of a rectangle. This has the noticeable performance characteristics that are seen whenever a polygon is constructed (see `Polygon2`). This operation occurs only if width and height were modified.

Variables `mincorner` (`pygorithm.geometry.vector2.Vector2`) – the position of this polygon

`__init__` (*width, height, mincorner=None*)

Create a new rectangle of width and height.

If `mincorner` is `None`, the origin is assumed.

Parameters

- **width** (`numbers.Number`) – width of this rect
- **height** (`numbers.Number`) – height of this rect
- **mincorner** (`pygorithm.geometry.vector2.Vector2` or `None`) – the position of this rect

Raises `ValueError` – if width or height are not strictly positive

polygon

Get the polygon representation of this rectangle, without the offset. Lazily initialized and up-to-date with width and height.

Caution: This does not include the `mincorner` (which should be passed as offset for polygon operations)

Returns polygon representation of this rectangle

Return type `pygorithm.geometry.polygon2.Polygon2`

width

Get or set the width of this rect.

Caution: Setting the width of the rectangle will remove the polygon caching required for rectangle-polygon collision.

Returns width of this rect

Return type `numbers.Number`

Raises **ValueError** – if trying to set `width <= 1e-07`

height

Get or set the height of this rect

Caution: Setting the height of the rectangle will remove the cached operations required for rectangle-polygon collision.

Returns height of this rect

Return type `numbers.Number`

Raises **ValueError** – if trying to set `height <= 1e-07`

area

Get the area of this rect

Returns area of this rect

Return type `numbers.Number`

static project_onto_axis (*rect*, *axis*)

Project the rect onto the specified axis.

Tip: This function is extremely fast for vertical or horizontal axes.

Parameters

- **rect** (`pygorithm.geometry.rect2.Rect2`) – the rect to project
- **axis** (`pygorithm.geometry.vector2.Vector2`) – the axis to project onto (normalized)

Returns the projection of the rect along axis

Return type `pygorithm.geometry.axisall.AxisAlignedLine`

static contains_point (*rect*, *point*)

Determine if the rect contains the point

Distinguish between points that are on the edge of the rect and those that are not.

Tip: This will never return `True, True`

Parameters

- **rect** (`pygorithm.geometry.rect2.Rect2`) – the rect
- **point** (`pygorithm.geometry.vector2.Vector2`) – the point

Returns point on edge, point inside

Return type `bool, bool`

classmethod `_find_intersection_rects` (*rect1*, *rect2*, *find_mtv=True*)

Find the intersection between two rectangles.

Not intended for direct use. See `find_intersection()`

Parameters

- **rect1** (`pygorithm.geometry.rect2.Rect2`) – first rectangle
- **rect2** (`pygorithm.geometry.rect2.Rect2`) – second rectangle
- **find_mtv** (`bool`) – False to never find mtv (may allow small performance improvement)

Returns (touching, overlapping, (mtv distance, mtv axis))

Return type (`bool`, `bool`, (`numbers.Number`, `pygorithm.geometry.vector2.Vector2`) or `None`)

classmethod `_find_intersection_rect_poly` (*rect*, *poly*, *offset*, *find_mtv=True*)

Find the intersection between a rect and polygon.

Not intended for direct use. See `find_intersection()`

Parameters

- **rect** (`pygorithm.geometry.rect2.Rect2`) – rectangle
- **poly** (`pygorithm.geometry.polygon2.Polygon2`) – polygon
- **offset** (`pygorithm.geometry.vector2.Vector2`) – offset for the polygon
- **find_mtv** (`bool`) – False to never find mtv (may allow small performance improvement)

Returns (touching, overlapping, (mtv distance, mtv axis))

Return type (`bool`, `bool`, (`numbers.Number`, `pygorithm.geometry.vector2.Vector2`) or `None`)

`__weakref__`

list of weak references to the object (if defined)

classmethod `_find_intersection_poly_rect` (*poly*, *offset*, *rect*, *find_mtv=True*)

Find the intersection between a polygon and rect.

Not intended for direct use. See `find_intersection()`

Parameters

- **poly** (`pygorithm.geometry.polygon2.Polygon2`) – polygon
- **offset** (`pygorithm.geometry.vector2.Vector2`) – offset for the polygon
- **rect** (`pygorithm.geometry.rect2.Rect2`) – rectangle
- **find_mtv** (`bool`) – False to never find mtv (may allow small performance improvement)

Returns (touching, overlapping, (mtv distance, mtv axis))

Return type (`bool`, `bool`, (`numbers.Number`, `pygorithm.geometry.vector2.Vector2`) or `None`)

classmethod `find_intersection` (**args*, ***kwargs*)

Determine the state of intersection between a rect and a polygon.

For Rect-Polygon intersection:

Must be passed in 3 arguments - a *Rect2*, a *Polygon2*, and a *Vector2*. The vector must come immediately after the polygon, but the rect can be either the first or last unnamed argument. If it is the first argument, the mtv is against the rectangle. If it is the last argument, the mtv is against the polygon.

For Rect-Rect intersection:

Must be passed in 2 arguments (both rects).

Note: The first argument is checked with `isinstance(arg, Rect2)`. If this is `False`, the first argument is assumed to be a `Polygon2`. If you want to use a compatible rectangle class for which this check would fail, you can call `_find_intersection_rect_poly()` directly or pass the polygon first and invert the resulting mtv (if one is found). If two unnamed arguments are provided, they are assumed to be both rects without further checks.

Examples:

```
from pygorithm.geometry import (vector2, polygon2, rect2)

octogon = polygon2.Polygon2.from_regular(8, 1)
oct_offset = vector2.Vector2(0.5, 0)

unit_square = rect2.Rect2(1, 1)

# find mtv for square against octogon
touching, overlapping, mtv = rect2.Rect2.find_intersection(unit_square,
↳octogon, oct_offset)

# find mtv for octogon against square
touching, overlapping, mtv = rect2.Rect2.find_intersection(octogon, oct_
↳offset, unit_square)

# find intersection but skip mtv (two options)
touching, overlapping, alwaysNone = rect2.Rect2.find_intersection(unit_square,
↳ octogon, oct_offset, find_mtv=False)
touching, overlapping, alwaysNone = rect2.Rect2.find_intersection(octogon,
↳oct_offset, unit_square, find_mtv=False)

big_square = rect2.Rect2(2, 2, vector2.Vector2(-1.5, 0))

# find mtv for square against big square
touching, overlapping, mtv = rect2.Rect2.find_intersection(unit_square, big_
↳square)

# find mtv for big square against square
touching, overlapping, mtv = rect2.Rect2.find_intersection(big_square, unit_
↳square)
```

Parameters

- **find_mtv** (*bool*) – if mtv should be found where possible (default `True`)
- **args** (*list*) – 2 arguments for rect-rect, 3 arguments for rect-polygon (see above)

Returns (`touching`, `overlapping`, (`mtv distance`, `mtv axis`))

Return type (`bool`, `bool`, (`numbers.Number`, `pygorithm.geometry.vector2.Vector2`) or `None`)

__repr__()

Create an unambiguous representation of this rectangle.

Example:

```

from pygorithm.geometry import (vector2, rect2)

unit_square = rect2.Rect2(1, 1, vector2.Vector2(3, 4))

# prints rect2(width=1, height=1, mincorner=vector2(x=3, y=4))
print(repr(unit_square))

```

Returns unambiguous representation of this rectangle**Return type** string**__str__()**

Create a human readable representation of this rectangle

Example:

```

from pygorithm.geometry import (vector2, rect2)

unit_square = rect2.Rect2(1, 1, vector2.Vector2(3, 4))
ugly_rect = rect2.Rect2(0.7071234, 0.7079876, vector2.Vector2(0.56789123, 0.
↳876543))

# prints rect(1x1 at <3, 4>)
print(str(unit_square))

# prints rect(0.707x0.708 at <0.568, 0.877>)
print(str(ugly_rect))

```

Returns human-readable representation of this rectangle**Return type** string

2.5.8 Extrapolated Intersection

Author: Timothy Moore Created On: 4th September 2017

Contains various approaches to determining if a polygon will intersect another polygon as one or both polygons go along a single direction at a constant speed.

This problem could be thought of as one of extrapolation - given these initial conditions, extrapolate to determine if intersections will occur.

Note: Touching is not considered intersecting in this module, unless otherwise stated. Touching is determined using *math.isclose*

`pygorithm.geometry.extrapolated_intersection.calculate_one_moving_point_and_one_stationary_`

Determine if the point moving at velocity will intersect the line.

The line is positioned at offset. Given a moving point and line segment, determine if the point will ever intersect the line segment.

Caution: Points touching at the start are considered to be intersection. This is because there is no way to get the “direction” of a stationary point like you can a line or polygon.

Parameters

- **point** (`pygorithm.geometry.vector2.Vector2`) – the starting location of the point
- **velocity** (`pygorithm.geometry.vector2.Vector2`) – the velocity of the point
- **line** (`pygorithm.geometry.line2.Line2`) – the geometry of the stationary line
- **offset** (`pygorithm.geometry.vector2.Vector2`) – the offset of the line

Returns if the point will intersect the line, distance until intersection

Return type `bool`, `numbers.Number` or `None`

`pygorithm.geometry.extrapolated_intersection.calculate_one_moving_line_and_one_stationary_`

Determine if the moving line will intersect the stationary line.

Given two line segments, one moving and one not, determine if they will ever intersect.

Parameters

- **line1** (`pygorithm.geometry.line2.Line2`) – the geometry of the moving line
- **offset1** (`pygorithm.geometry.vector2.Vector2`) – the starting location of the moving line
- **velocity1** (`pygorithm.geometry.vector2.Vector2`) – the velocity of the moving line
- **_line2** (`pygorithm.geometry.line2.Line2`) – the geometry of the second line
- **offset2** (`pygorithm.geometry.vector2.Vector2`) – the location of the second line

Returns if the lines will ever intersect, distance until intersection

Return type `bool`, `numbers.Number` or `None`

`pygorithm.geometry.extrapolated_intersection.calculate_one_moving_and_one_stationary` (*poly1*, *poly1_offset*, *poly1_velocity*, *poly2*, *poly2_offset*)

Determine if the moving polygon will intersect the stationary polygon.

This is the simplest question. Given two polygons, one moving and one not, determine if the two polygons will ever intersect (assuming they maintain constant velocity).

Parameters

- **poly1** (*pygorithm.geometry.polygon2.Polygon2*) – the geometry of the polygon that is moving
- **poly1_offset** (*pygorithm.geometry.vector2.Vector2*) – the starting location of the moving polygon
- **poly1_velocity** (*pygorithm.geometry.vector2.Vector2*) – the velocity of the moving polygon
- **poly2** (*pygorithm.geometry.polygon2.Polygon2*) – the geometry of the stationary polygon
- **poly2_offset** (*pygorithm.geometry.vector2.Vector2*) – the offset of the stationary polygon

Returns if they will intersect

Return type bool

`pygorithm.geometry.extrapolated_intersection.calculate_one_moving_one_stationary_distance`

Determine if the moving polygon will intersect the stationary polygon within some distance.

This is a step up, and very similar to the actual problem many any-angle pathfinding algorithms run into. Given two polygons, 1 moving and 1 stationary, determine if the first polygon will intersect the second polygon before moving a specified total distance.

Parameters

- **poly1** (*pygorithm.geometry.polygon2.Polygon2*) – the geometry of the polygon that is moving
- **poly1_offset** (*pygorithm.geometry.vector2.Vector2*) – the starting location of the moving polygon
- **poly1_velocity** (*pygorithm.geometry.vector2.Vector2*) – the velocity of the moving polygon
- **poly2** (*pygorithm.geometry.polygon2.Polygon2*) – the geometry of the stationary polygon
- **poly2_offset** (*pygorithm.geometry.vector2.Vector2*) – the offset of the stationary polygon
- **max_distance** (*numbers.Number*) – the max distance that poly1 can go

Returns if they will intersect

Return type bool

`pygorithm.geometry.extrapolated_intersection.calculate_one_moving_one_stationary_along_path`

Determine if the moving polygon will intersect the stationary polygon as it moves from the start to the end.

This is a rewording of `calculate_one_moving_one_stationary_distancelimit()` that is more common. Given two polygons, 1 moving and 1 stationary, where the moving polygon is going at some speed from one point to another, determine if the two polygons will intersect.

Parameters

- **poly1** (`pygorithm.geometry.polygon2.Polygon2`) – the geometry of the polygon that is moving
- **poly1_start** (`pygorithm.geometry.vector2.Vector2`) – where the moving polygon begins moving from
- **poly1_end** (`pygorithm.geometry.vector2.Vector2`) – where the moving polygon stops moving
- **poly2** (`pygorithm.geometry.polygon2.Polygon2`) – the geometry of the stationary polygon
- **poly2_offset** (`pygorithm.geometry.vector2.Vector2`) – the location of the second polygon

Returns if they will intersect

Return type bool

`pygorithm.geometry.extrapolated_intersection.calculate_one_moving_many_stationary` (`poly1`, `poly1_offset`, `poly1_velocity`, `other_poly_offset_tuples`)

Determine if the moving polygon will intersect anything as it moves at a constant direction and speed forever.

This is the simplest arrangement of this problem with a collection of stationary polygons. Given many polygons of which 1 is moving, determine if the moving polygon intersects the other polygons now or at some point in the future if it moves at some constant direction and speed forever.

This does not verify the stationary polygons are not intersecting.

Parameters

- **poly1** (`pygorithm.geometry.polygon2.Polygon2`) – the geometry of the polygon that is moving
- **poly1_offset** (`pygorithm.geometry.vector2.Vector2`) – the starting location of the moving polygon
- **poly1_velocity** (`pygorithm.geometry.vector2.Vector2`) – the velocity of the moving polygon
- **other_poly_offset_tuples** (list of (`pygorithm.geometry.polygon2.Polygon2`, `pygorithm.geometry.vector2.Vector2`)) – list of (polygon, offset) of the stationary polygons

Returns if an intersection will occur

Return type bool

```
pygorithm.geometry.extrapolated_intersection.calculate_one_moving_many_stationary_distance:
```

Determine if the moving polygon will intersect anything as it moves in a constant direction and speed for a certain distance.

This does not verify the stationary polygons are not intersecting.

Parameters

- **poly1** (*pygorithm.geometry.polygon2.Polygon2*) – the geometry of the polygon that is moving
- **poly1_offset** (*pygorithm.geometry.vector2.Vector2*) – the starting location of the moving polygon
- **poly1_velocity** (*pygorithm.geometry.vector2.Vector2*) – the velocity of the moving polygon
- **max_distance** (*numbers.Number*) – the max distance the polygon will go
- **other_poly_offset_tuples** (list of (*pygorithm.geometry.polygon2.Polygon2*, *pygorithm.geometry.vector2.Vector2*)) – list of (polygon, offset) of the stationary polygons

Returns if an intersection will occur

Return type bool

```
pygorithm.geometry.extrapolated_intersection.calculate_one_moving_many_stationary_along_pat
```

Determine if a polygon that moves from one point to another will intersect anything.

This is the question that the Theta* family of pathfinding algorithms require. It is simply a rewording of *calculate_one_moving_many_stationary_distance*limit()

This does not verify the stationary polygons are not intersecting.

Parameters

- **poly1** (*pygorithm.geometry.polygon2.Polygon2*) – the geometry of the polygon that is moving
- **poly1_start** (*pygorithm.geometry.vector2.Vector2*) – where the polygon begins moving from
- **poly1_end** (*pygorithm.geometry.vector2.Vector2*) – where the polygon stops moving at
- **other_poly_offset_tuples** (list of (*pygorithm.geometry.polygon2.Polygon2*, *pygorithm.geometry.vector2.Vector2*)) – list of (polygon, offset) of the stationary polygons

Returns if an intersection will occur

Return type bool

```
pygorithm.geometry.extrapolated_intersection.calculate_two_moving (poly1,  
                                                                    poly1_offset,  
                                                                    poly1_vel,  
                                                                    poly2,  
                                                                    poly2_offset,  
                                                                    poly2_vel)
```

Determine if two moving polygons will intersect at some point.

This is the simplest question when there are multiple moving polygons. Given two polygons moving at a constant velocity and direction forever, determine if an intersection will occur.

It should be possible for the reader to extrapolate from this function and the process for stationary polygons to create similar functions to above where all or some polygons are moving.

Parameters

- **poly1** (*pygorithm.geometry.polygon2.Polygon2*) – the first polygon
- **poly1_offset** (*pygorithm.geometry.vector2.Vector2*) – where the first polygon starts at
- **poly1_vel** (*pygorithm.geometry.vector2.Vector2*) – the velocity of the first polygon
- **poly2** (*pygorithm.geometry.polygon2.Polygon2*) – the second polygon
- **poly2_offset** (*pygorithm.geometry.vector2.Vector2*) – where the second polygon starts at
- **poly2_vel** (*pygorithm.geometry.vector2.Vector2*) – the velocity of the second polygon

Returns if an intersection will occur

Return type bool

2.6 Greedy Algorithms

A place for implementation of greedy algorithms

2.6.1 Features

- To see all the available functions in a module, you can just type `help()` with the module name as argument. For example,

```
>>> from pygorithm import greedy_algorithm  
>>> help(greedy_algorithm)  
Help on package pygorithm.greedy_algorithm in pygorithm:  
  
NAME  
pygorithm.greedy_algorithm - Collection for greedy algorithms  
  
PACKAGE CONTENTS  
activity_selection  
fractional_knapsack  
  
DATA  
__all__ = ['fractional_knapsack', 'activity_selection']
```


2.6.2 Activity Selection Problem

- Functions and their uses

Author: OMKAR PATHAK Created On: 26th August 2017

`pygorithm.greedy_algorithm.activity_selection.activity_selection` (*start_times*,
finish_times)

The activity selection problem is a combinatorial optimization problem concerning the selection of non-conflicting activities to perform within a given time frame, given a set of activities each marked by a start time (si) and finish time (fi). The problem is to select the maximum number of activities that can be performed by a single person or machine, assuming that a person can only work on a single activity at a time.

Parameters

- **start_times** – An array that contains start time of all activities
- **finish_times** – An array that contains finish time of all activities

`pygorithm.greedy_algorithm.activity_selection.get_code` ()
returns the code for the activity_selection function

2.6.3 Fractional Knapsack

- Functions and their uses

Author: SHARAD BHAT Created On: 22nd August 2017

`pygorithm.greedy_algorithm.fractional_knapsack.knapsack` (*w*,
item_values,
item_weights)

The knapsack problem or rucksack problem is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items.

Parameters

- **w** – maximum weight capacity
- **item_values** – a list of values of items in the knapsack
- **item_weights** – a list of weights of items in the knapsack

`pygorithm.greedy_algorithm.fractional_knapsack.get_code` ()
returns the code for the knapsack function

2.7 Math

Some of the mathematical algorithms and their implementations

2.7.1 Quick Start Guide

```
# import the required math
from pygorithm.math import lcm

# find the lcm for all the elements in the list
```

```
ans = lcm.lcm([3, 12, 16])

#print the result
print(ans)
```

2.7.2 Features

- **Algorithms available:**
 - LCM (lcm)
 - Sieve of Eratostenes (sieve_of_eratosthenes)
 - Factorial
 - Binary To decimal conversion
 - Decimal to binary conversion
 - Hex To decimal conversion
 - Decimal to hex conversion
- To see all the available functions in a module there is a *modules()* function available. For example,

```
>>> from pygorithm.math import modules
>>> modules.modules()
['lcm', 'sieve_of_eratosthenes']
```

- Get the code used for any of the algorithm

```
from pygorithm.math import lcm

# for printing the source code of LCM function
print(lcm.get_code())
```

2.7.3 LCM

- Functions and their uses

`lcm.lcm (List)`

- **List** : list or array of which LCM is to be found
- **Return Value** : returns the integer value of LCM

`lcm.get_code ()`

- **Return Value** : returns the code for the `lcm.lcm ()` function

2.7.4 Sieve of Eratostenes

- Functions and their uses

`sieve_of_eratosthenes.sieve_of_eratosthenes (n)`

- **n** : upper limit upto which prime numbers are to be found
- **Return Value** : returns the *list* of all primes upto n

```
sieve_of_eratostenes.get_code()
```

- **Return Value** : returns the code for the `sieve_of_eratostenes.sieve_of_eratostenes()` function

2.7.5 Factorial

- Functions and their uses

```
factorial.factorial(number)
```

- **number** : integer number of which factorial is to be found
- **Return Value** : returns the integer of factorial of the number

```
factorial.get_code()
```

- **Return Value** : returns the code for the `factorial.factorial()` function

2.7.6 Conversion

- Functions and their uses

```
conversion.decimal_to_binary(number)
```

- **number** : decimal number in string or integer format
- **Return Value** : returns the string of equivalent binary number

```
conversion.binary_to_decimal(number)
```

- **number** : binary number in string or integer format
- **Return Value** : returns the integer of equivalent decimal number

```
conversion.decimal_to_hex(number)
```

- **number** : decimal number in string or integer format
- **Return Value** : returns the string of equivalent hex number

```
conversion.hex_to_decimal(number)
```

- **number** : hex number in string or integer format
- **Return Value** : returns the integer of equivalent decimal number

2.8 Path Finding Algorithms

Some pathfinding algorithms and their implementations

2.8.1 Quick Start Guide

```
# import the required pathing algorithm
from pygorithm.pathing import dijkstra

# import a graph data structure
from pygorithm.data_structures import graph
```

```
# initialize the graph with nodes from (0, 0) to (4, 4)
# with weight corresponding to distance (orthogonal
# is 1, diagonal is sqrt(2))
my_graph = graph.WeightedUndirectedGraph()
my_graph.gridify(5, 1)

# make the graph more interesting by removing along the
# x=2 column except for (2,4)
my_graph.remove_edge((2, 0))
my_graph.remove_edge((2, 1))
my_graph.remove_edge((2, 2))
my_graph.remove_edge((2, 3))

# calculate a path
my_path = dijkstra.find_path(my_graph, (0, 0), (3, 0))

# print path
print(' -> '.join(my_path))
# (0, 0) -> (1, 1) -> (0, 2) -> (1, 3) -> (2, 4) -> (3, 3) -> (3, 2) -> (3, 1) -> (3, 0)
```

2.8.2 Features

- **Algorithms available:**
 - Dijkstra (dijkstra)
 - Unidirectional AStar (astar)
 - BiDirectional AStar (astar)
- To see all the available functions in a module there is a *modules()* function available. For example,

```
>>> from pygorithm.pathfinding import modules
>>> modules.modules()
['dijkstra', 'astar']
```

- Get the code used for any of the algorithm

```
from pygorithm.pathing import dijkstra

# for printing the source code of Dijkstra object
print(dijkstra.Dijkstra.get_code())
```

2.8.3 Dijkstra

- Functions and their uses

`dijkstra.Dijkstra.find_path` (*pygorithm.data_structures.WeightedUndirectedGraph*, *vertex*, *vertex*)

- **pygorithm.data_structures.WeightedUndirectedGraph** : acts like an object with *graph* (see *WeightedUndirectedGraph*)
- **vertex** : any hashable type for the start of the path
- **vertex** : any hashable type for the end of the path

- **Return Value** : returns a *List* of vertexes (of the same type as the graph) starting with from and going to to. This algorithm does *not* respect weights.

`dijkstra.get_code()`

- **Return Value** : returns the code for the `Dijkstra` object

2.8.4 Unidirectional AStar

- Functions and their uses

`astar.OneDirectionalAStar.find_path` (`pygorithm.data_structures.WeightedUndirectedGraph`,
`vertex`, `vertex`, `function`)

- **pygorithm.data_structures.WeightedUndirectedGraph** : acts like an object with `graph` and `get_edge_weight` (see `WeightedUndirectedGraph`)
- **vertex** : any hashable type for the start of the path
- **vertex** : any hashable type for the end of the path
- **function** : `function(graph, vertex, vertex)` returns numeric - a heuristic function for distance between two vertices
- **Return Value** : returns a *List* of vertexes (of the same type of the graph) starting from from and going to to. This algorithm respects weights, but is only guaranteed to be optimal if the heuristic is admissable. An admissable function will never *overestimate* the cost from one node to another (in other words, it is optimistic).

2.8.5 BiDirectional AStar

- Functions and their uses

`astar.BiDirectionalAStar.find_path` (`pygorithm.data_structures.WeightedUndirectedGraph`,
`vertex`, `vertex`, `function`)

- **pygorithm.data_structures.WeightedUndirectedGraph** : acts like an object with `graph` and `get_edge_weight` (see `WeightedUndirectedGraph`)
- **vertex** : any hashable type for the start of the path
- **vertex** : any hashable type for the end of the path
- **function** : `function(graph, vertex, vertex)` returns numeric - a heuristic function for distance between two vertices
- **Return Value** : returns a *List* of vertexes (of the same type of the graph) starting from from and going to to. This algorithm respects weights, but is only guaranteed to be optimal if the heuristic is admissable. An admissable function will never *overestimate* the cost from one node to another (in other words, it is optimistic).

2.9 Searching

Learning searching algorithms easily!

2.9.1 Quick Start Guide

```
# import the required searching algorithm
from pygorithm.searching import binary_search

my_list = [12, 4, 2, 14, 3, 7, 5]
```

```
# pre-requisite for binary search is that the list should be sorted
my_list.sort()

# to search an element in the above list
index = binary_search.search(my_list, 7)
print(index)
```

2.9.2 Features

- **Searching algorithms available:**
 - Linear Search (`linear_search`)
 - Binary Search (`binary_search`)
 - Breadth First Search (`breadth_first_search`)
 - Depth First Search (`depth_first_search`)
- To see all the available functions in a module, you can just type `help()` with the module name as argument. For example,

```
>>> from pygorithm import searching
>>> help(searching)
Help on package pygorithm.searching in pygorithm:

NAME
pygorithm.searching - Collection of searching algorithms

PACKAGE CONTENTS
binary_search
breadth_first_search
depth_first_search
linear_search
modules
quick_select
```

- For Searching: Remember `search()` function in `binary_search` module takes two parameters as a sorted list and the target element to be searched.

```
# import the required searching algorithm
from pygorithm.searching import binary_search

my_list = [12, 4, 2, 14, 3, 7, 5]

# pre-requisite for binary search is that the list should be sorted
my_list.sort()

# to search an element in the above list
index = binary_search.search(my_list, 7)
print(index)
```

- Get time complexities of all the searching algorithms

```
from pygorithm.searching import binary_search
```

```
# for printing time complexities of binary_search
print(binary_search.time_complexities())
```

- Get the code used for any of the algorithm

```
from pygorithm.searching import binary_search

# for printing the source code of bubble_sort
print(binary_search.get_code())
```

2.9.3 Binary Search

- Functions and their uses

`binary_search.search(_list, target)`

- **_list** : Sorted list in which the target is to be searched
- **target** : target to be searched in the list
- **Return Value** : returns the position (index) of the target if target found, else returns False

`binary_search.time_complexities()`

- **Return Value** : returns time complexities (Best, Average, Worst)

`binary_search.get_code()`

- **Return Value** : returns the code for the `binary_search.search()` function

2.9.4 Linear Search

- Functions and their uses

`linear_search.search(_list, target)`

- **_list** : the list in which item is to searched
- **target** : target to be searched in the list
- **Return Value** : returns the position (index) of the target if target found, else returns False

`linear_search.time_complexities()`

- **Return value** : returns time complexities (Best, Average, Worst)

`linear_search.get_code()`

- **Return Value** : returns the code for the `linear_search.search()` function

2.9.5 Breadth First Search

- Functions and their uses

`breadth_first_search.search(graph, startVertex)`

- **graph** : takes the graph data structures with edges and vertices
- **startVertex** : it tells the function the vertex to start with
- **Return Value** : returns the *set* of bfs for the graph

`breadth_first_search.time_complexities()`

- **Return Value** : returns time complexities

`breadth_first_search.get_code()`

- **Return Value** : returns the code for the `breadth_first_search.search()` function

2.9.6 Depth First Search

- Functions and their uses

`breadth_first_search.search(graph, start, path)`

- **graph** : takes the graph data structures with edges and vertices
- **start** : it tells the function the vertex to start with
- **path** : returns the list containing the required dfs
- **Return Value** : returns the *list* of dfs for the graph

`breadth_first_search.time_complexities()`

- **Return Value** : returns time complexities

`breadth_first_search.get_code()`

- **Return Value** : returns the code for the `depth_first_search.search()` function

2.9.7 Quick Select Search

- Functions and their uses

`quick_select.search(array, n)`

- **array** : an unsorted array
- **n** : nth number to be searched in the given *array*
- **Return Value** : returns the nth element

`quick_select.time_complexities()`

- **Return Value** : returns time complexities

`quick_select.get_code()`

- **Return Value** : returns the code for the `quick_select.search()` function

2.9.8 Interpolation Search

- Functions and their uses

`interpolation_search.search(_list, target)`

- **_list** : *Sorted* list in which the target is to be searched
- **target** : target to be searched in the list
- **Return Value** : returns the position (index) of the target if target found, else returns False

`interpolation_search.time_complexities()`

- **Return Value** : returns time complexities (Best, Average, Worst)

```
interpolation_search.get_code()
```

- **Return Value** : returns the code for the `interpolation_search.search()` function

2.10 Sorting

Just sort the way you want.

2.10.1 Quick Start Guide

```
# import the required sort
from pygorithm.sorting import bubble_sort

my_list = [12, 4, 2, 14, 3, 7, 5]

# to sort the _list
sorted_list = bubble_sort.sort(my_list)
```

2.10.2 Features

- To see all the available functions in a module, you can just type `help()` with the module name as argument. For example,

```
>>> from pygorithm import sorting
>>> help(sorting)
Help on package pygorithm.sorting in pygorithm:

NAME
    pygorithm.sorting - Collection of sorting methods

PACKAGE CONTENTS
    bubble_sort
    bucket_sort
    counting_sort
    heap_sort
    insertion_sort
    merge_sort
    modules
    quick_sort
    selection_sort
    shell_sort
```

- For sorting: Remember `sort()` function takes its parameter as a `_list` only.

```
# import the required sort
from pygorithm.sorting import bubble_sort

my_list = [12, 4, 2, 14, 3, 7, 5]

# to sort the _list
sorted_list = bubble_sort.sort(my_list)
```

- Get time complexities of all the sorting algorithms

```
from pygorithm.sorting import bubble_sort
# for printing time complexities of bubble_sort
print(bubble_sort.time_complexities())
```

- Get the code used for any of the algorithm

```
from pygorithm.sorting import bubble_sort
# for printing the source code of bubble_sort
print(bubble_sort.get_code())
```

2.10.3 Bubble Sort

- Functions and their uses

`bubble_sort.sort(_list)`

- `_list` : list or array to be sorted
- **Return Value** : returns the sorted *list*

`bubble_sort.time_complexities()`

- **Return Value** : returns time complexities (Best, Average, Worst)

`bubble_sort.get_code()`

- **Return Value** : returns the code for the `bubble_sort.sort()` function
- For improved Bubble sort

`bubble_sort.improved_sort(_list)`

- `_list` : list or array to be sorted
- **Return Value** : returns the sorted *list*

2.10.4 Bucket Sort

- Functions and their uses

`bucket_sort.sort(_list, bucketSize)`

- `_list` : list or array to be sorted
- `bucketSize` : size of the bucket. Default is 5
- **Return Value** : returns the sorted *list*

`bucket_sort.time_complexities()`

- **Return Value** : returns time complexities (Best, Average, Worst)

`bucket_sort.get_code()`

- **Return Value** : returns the code for the `bucket_sort.sort()` function

2.10.5 Counting Sort

- Functions and their uses

`counting_sort.sort(_list)`

- **_list** : *list* or *array* to be sorted
- **Return Value** : returns the sorted *list*

`counting_sort.time_complexities()`

- **Return Value** : returns time complexities (Best, Average, Worst)

`counting_sort.get_code()`

- **Return Value** : returns the code for the `counting_sort.sort()` function

2.10.6 Heap Sort

- Functions and their uses

`heap_sort.sort(_list)`

- **_list** : *list* or *array* to be sorted
- **Return Value** : returns the sorted *list*

`heap_sort.time_complexities()`

- **Return Value** : returns time complexities (Best, Average, Worst)

`heap_sort.get_code()`

- **Return Value** : returns the code for the `heap_sort.sort()` function

2.10.7 Insertion Sort

- Functions and their uses

`insertion_sort.sort(_list)`

- **_list** : *list* or *array* to be sorted
- **Return Value** : returns the sorted *list*

`insertion_sort.time_complexities()`

- **Return Value** : returns time complexities (Best, Average, Worst)

`insertion_sort.get_code()`

- **Return Value** : returns the code for the `insertion_sort.sort()` function

2.10.8 Merge Sort

- Functions and their uses

`merge_sort.sort(_list)`

- **_list** : *list* or *array* to be sorted
- **Return Value** : returns the sorted *list*

`merge_sort.time_complexities()`

- **Return Value** : returns time complexities (Best, Average, Worst)

`merge_sort.get_code()`

- **Return Value** : returns the code for the `merge_sort.sort()` function

2.10.9 Quick Sort

- Functions and their uses

`quick_sort.sort(_list)`

- **_list** : *list* or *array* to be sorted
- **Return Value** : returns the sorted *list*

`quick_sort.time_complexities()`

- **Return Value** : returns time complexities (Best, Average, Worst)

`quick_sort.get_code()`

- **Return Value** : returns the code for the `quick_sort.sort()` function

2.10.10 Selection Sort

- Functions and their uses

`selection_sort.sort(_list)`

- **_list** : *list* or *array* to be sorted
- **Return Value** : returns the sorted *list*

`selection_sort.time_complexities()`

- **Return Value** : returns time complexities (Best, Average, Worst)

`selection_sort.get_code()`

- **Return Value** : returns the code for the `selection_sort.sort()` function

2.10.11 Shell Sort

- Functions and their uses

`shell_sort.sort(_list)`

- **_list** : *list* or *array* to be sorted
- **Return Value** : returns the sorted *list*

`shell_sort.time_complexities()`

- **Return Value** : returns time complexities (Best, Average, Worst)

`shell_sort.get_code()`

- **Return Value** : returns the code for the `shell_sort.sort()` function

2.11 Strings

A place for implementation of string algorithms

2.11.1 Features

- To see all the available functions in a module, you can just type `help()` with the module name as argument. For example,

```
>>> from pygorithm import strings
>>> help(strings)
Help on package pygorithm.strings in pygorithm:

NAME
pygorithm.strings - Collection of string methods and functions

PACKAGE CONTENTS
anagram
isogram
manacher_algorithm
palindrome
pangram
```

2.11.2 Anagram

- Functions and their uses

Author: OMKAR PATHAK Created On: 17th August 2017

`pygorithm.strings.anagram.is_anagram(word, _list)`

ANAGRAM An anagram is direct word switch or word play, the result of rearranging the letters of a word or phrase to produce a new word or phrase, using all the original letters exactly once we are taking a word and a list. We return the anagrams of that word from the given list and return the list of anagrams else return empty list.

Parameters

- **word** – word
- **_list** – list of words

Returns anagrams

`pygorithm.strings.anagram.get_code()`

returns the code for the `is_anagram` function :return: source code

2.11.3 Isogram

- Functions and their uses

Author: OMKAR PATHAK Created On: 17th August 2017

`pygorithm.strings.isogram.is_isogram(word)`

An isogram (also known as a “nonpattern word”) is a logological term for a word or phrase without a repeating letter

Parameters `word` – word to check

Returns `bool`

```
pygorithm.strings.isogram.get_code()
    returns the code for the is_isogram function :return: source code
```

2.11.4 Palindrome

- Functions and their uses

Author: OMKAR PATHAK Created On: 17th August 2017

```
pygorithm.strings.palindrome.is_palindrome(string)
    Checks the string for palindrome
```

Parameters `string` – string to check

Returns `true` if string is a palindrome `false` if not

```
pygorithm.strings.palindrome.get_code()
    returns the code for the is_palindrome function :return: source code
```

2.11.5 Pangram

- Functions and their uses

Author: OMKAR PATHAK Created On: 17th August 2017

```
pygorithm.strings.pangram.is_pangram(sentence)
    A sentence containing every letter of the alphabet.
```

Parameters `sentence` – Sentence to check

Returns `bool`

```
pygorithm.strings.pangram.get_code()
    returns the code for the is_pangram function :return: source code
```

2.11.6 Manacher's Algorithm

- Functions and their uses

Author: OMKAR PATHAK Created at: 27th August 2017

```
pygorithm.strings.manacher_algorithm.manacher(string)
```

Computes length of the longest palindromic substring centered on each char in the given string. The idea behind this algorithm is to reuse previously computed values whenever possible (palindromes are symmetric).

Example (interleaved string): i 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 s # a # b # c # q # q # q # q # q # x # y # P 0 1 0 1 0 1 2 3 4 5 6 5 4 ?

^ ^ ^ ^

mirror center current right

We're at index 15 wondering shall we compute (costly) or reuse. The mirror value for 15 is 9 (center is in 12). $P[\text{mirror}] = 3$ which means a palindrome of length 3 is centered at this index. A palindrome of same length would be placed in index 15, if $15 + 3 \leq 18$ (right border of large parlindrome centered in 12). This condition is satisfied, so we can reuse value from index 9 and avoid costly computation.

`pygorithm.strings.manacher_algorithm.get_code()`
returns the code for the manacher's algorithm :return: source code

Quick Start Guide

- Download and install the Python package. [Installation instructions](#)
- Just import the required algorithm and start learning

```
from pygorithm.sorting import bubble_sort

# This will print the code for bubble sort
print(bubble_sort.get_code())

my_list = [12, 4, 2, 14, 3, 7, 5]

# to sort the list
sorted_list = bubble_sort.sort(my_list)
```


CHAPTER 4

Getting Started

- For getting started, first download the package using Python package manager

```
pip3 install pygorithm
```

- For Python 2, you can use pip instead.
- Or you can download the source code from [here](#), and then just install the package using

```
python setup.py install
```


p

- pygorithm.binary.ascii, 4
- pygorithm.binary.base10, 5
- pygorithm.binary.base16, 6
- pygorithm.binary.base2, 4
- pygorithm.data_structures.graph, 13
- pygorithm.data_structures.heap, 15
- pygorithm.data_structures.linked_list,
10
- pygorithm.data_structures.quadtree, 16
- pygorithm.data_structures.queue, 9
- pygorithm.data_structures.stack, 8
- pygorithm.data_structures.tree, 11
- pygorithm.data_structures.trie, 16
- pygorithm.dynamic_programming.binary_knapsack,
22
- pygorithm.dynamic_programming.lis, 23
- pygorithm.geometry.extrapolated_intersection,
47
- pygorithm.greedy_algorithm.activity_selection,
53
- pygorithm.greedy_algorithm.fractional_knapsack,
53
- pygorithm.strings.anagram, 65
- pygorithm.strings.isogram, 65
- pygorithm.strings.manacher_algorithm,
66
- pygorithm.strings.palindrome, 66
- pygorithm.strings.pangram, 66

Symbols

- `__add__()` (pygorithm.geometry.vector2.Vector2 method), 26
`__init__()` (pygorithm.data_structures.quadtree.QuadTree method), 18
`__init__()` (pygorithm.data_structures.quadtree.QuadTreeEntity method), 17
`__init__()` (pygorithm.geometry.axisall.AxisAlignedLine method), 36
`__init__()` (pygorithm.geometry.line2.Line2 method), 31
`__init__()` (pygorithm.geometry.polygon2.Polygon2 method), 39
`__init__()` (pygorithm.geometry.rect2.Rect2 method), 43
`__init__()` (pygorithm.geometry.vector2.Vector2 method), 26
`__mul__()` (pygorithm.geometry.vector2.Vector2 method), 27
`__repr__()` (pygorithm.data_structures.quadtree.QuadTree method), 21
`__repr__()` (pygorithm.data_structures.quadtree.QuadTreeEntity method), 17
`__repr__()` (pygorithm.geometry.axisall.AxisAlignedLine method), 37
`__repr__()` (pygorithm.geometry.line2.Line2 method), 33
`__repr__()` (pygorithm.geometry.polygon2.Polygon2 method), 42
`__repr__()` (pygorithm.geometry.rect2.Rect2 method), 46
`__repr__()` (pygorithm.geometry.vector2.Vector2 method), 28
`__rmul__()` (pygorithm.geometry.vector2.Vector2 method), 27
`__str__()` (pygorithm.data_structures.quadtree.QuadTree method), 21
`__str__()` (pygorithm.data_structures.quadtree.QuadTreeEntity method), 17
`__str__()` (pygorithm.geometry.axisall.AxisAlignedLine method), 37
`__str__()` (pygorithm.geometry.line2.Line2 method), 34
`__str__()` (pygorithm.geometry.polygon2.Polygon2 method), 42
`__str__()` (pygorithm.geometry.rect2.Rect2 method), 47
`__str__()` (pygorithm.geometry.vector2.Vector2 method), 28
`__sub__()` (pygorithm.geometry.vector2.Vector2 method), 26
`__weakref__` (pygorithm.data_structures.quadtree.QuadTree attribute), 22
`__weakref__` (pygorithm.data_structures.quadtree.QuadTreeEntity attribute), 17
`__weakref__` (pygorithm.geometry.axisall.AxisAlignedLine attribute), 38
`__weakref__` (pygorithm.geometry.line2.Line2 attribute), 35
`__weakref__` (pygorithm.geometry.polygon2.Polygon2 attribute), 42
`__weakref__` (pygorithm.geometry.rect2.Rect2 attribute), 45
`__weakref__` (pygorithm.geometry.vector2.Vector2 attribute), 28
`_find_intersection_poly_rect()` (pygorithm.geometry.rect2.Rect2 class method), 45
`_find_intersection_rect_poly()` (pygorithm.geometry.rect2.Rect2 class method), 45
`_find_intersection_rects()` (pygorithm.geometry.rect2.Rect2 class method), 44
A
`activity_selection()` (in module pygorithm.greedy_algorithm.activity_selection), 53
`add_edge()` (pygorithm.data_structures.graph.CheckCycleDirectedGraph method), 14
`add_edge()` (pygorithm.data_structures.graph.CheckCycleUndirectedGraph method), 15
`add_edge()` (pygorithm.data_structures.graph.Graph method), 13

add_edge() (pygorithm.data_structures.graph.WeightedGraph method), 13
 add_edge() (pygorithm.data_structures.graph.WeightedUndirectedGraph method), 14
 are_parallel() (pygorithm.geometry.line2.Line2 static method), 34
 area (pygorithm.geometry.polygon2.Polygon2 attribute), 41
 area (pygorithm.geometry.rect2.Rect2 attribute), 44
 astar.BiDirectionalAStar.find_path() (built-in function), 57
 astar.OneDirectionalAStar.find_path() (built-in function), 57
 axis (pygorithm.geometry.line2.Line2 attribute), 32
 AxisAlignedLine (class in pygorithm.geometry.axisall), 35

B

binary_search.get_code() (built-in function), 59
 binary_search.search() (in module pygorithm.searching), 59
 binary_search.time_complexities() (built-in function), 59
 BinarySearchTree (class in pygorithm.data_structures.tree), 12
 BinaryTree (class in pygorithm.data_structures.tree), 11
 breadth_first_search.get_code() (built-in function), 60
 breadth_first_search.search() (built-in function), 59, 60
 breadth_first_search.time_complexities() (built-in function), 59, 60
 BSTNode (class in pygorithm.data_structures.tree), 12
 bubble_sort.get_code() (built-in function), 62
 bubble_sort.improved_sort() (built-in function), 62
 bubble_sort.sort() (built-in function), 62
 bubble_sort.time_complexities() (built-in function), 62
 bucket_sort.get_code() (built-in function), 62
 bucket_sort.sort() (built-in function), 62
 bucket_sort.time_complexities() (built-in function), 62
 build_word_list() (pygorithm.data_structures.trie.Trie method), 16

C

calculate_avg_ents_per_leaf() (pygorithm.data_structures.quadtree.QuadTree method), 20
 calculate_one_moving_and_one_stationary() (in module pygorithm.geometry.extrapolated_intersection), 48
 calculate_one_moving_line_and_one_stationary_line() (in module pygorithm.geometry.extrapolated_intersection), 48
 calculate_one_moving_many_stationary() (in module pygorithm.geometry.extrapolated_intersection), 50
 calculate_one_moving_many_stationary_along_path() (in module pygorithm.geometry.extrapolated_intersection), 51
 calculate_one_moving_many_stationary_distancelimit() (in module pygorithm.geometry.extrapolated_intersection), 50
 calculate_one_moving_one_stationary_along_path() (in module pygorithm.geometry.extrapolated_intersection), 50
 calculate_one_moving_one_stationary_distancelimit() (in module pygorithm.geometry.extrapolated_intersection), 49
 calculate_one_moving_point_and_one_stationary_line() (in module pygorithm.geometry.extrapolated_intersection), 47
 calculate_two_moving() (in module pygorithm.geometry.extrapolated_intersection), 51
 calculate_weight_misplaced_ents() (pygorithm.data_structures.quadtree.QuadTree method), 20
 calculate_y_intercept() (pygorithm.geometry.line2.Line2 method), 34
 check_cycle() (pygorithm.data_structures.graph.CheckCycleDirectedGraph method), 14
 check_cycle() (pygorithm.data_structures.graph.CheckCycleUndirectedGraph method), 15
 CheckCycleDirectedGraph (class in pygorithm.data_structures.graph), 14
 CheckCycleUndirectedGraph (class in pygorithm.data_structures.graph), 15
 contains_point() (pygorithm.geometry.axisall.AxisAlignedLine static method), 37
 contains_point() (pygorithm.geometry.line2.Line2 static method), 35
 contains_point() (pygorithm.geometry.polygon2.Polygon2 static method), 41
 contains_point() (pygorithm.geometry.rect2.Rect2 static method), 44
 conversion.binary_to_decimal() (built-in function), 55
 conversion.decimal_to_binary() (built-in function), 55
 conversion.decimal_to_hex() (built-in function), 55
 conversion.hex_to_decimal() (built-in function), 55
 counting_sort.get_code() (built-in function), 63
 counting_sort.sort() (built-in function), 63
 counting_sort.time_complexities() (built-in function), 63
 cross() (pygorithm.geometry.vector2.Vector2 method), 29

D

- delete() (pygorithm.data_structures.linked_list.DoublyLinkedList method), 10
- delete() (pygorithm.data_structures.linked_list.SinglyLinkedList method), 10
- delete() (pygorithm.data_structures.tree.BinarySearchTree method), 12
- delete() (pygorithm.data_structures.tree.BSTNode method), 12
- delete_front() (pygorithm.data_structures.queue.Deque method), 9
- delete_rear() (pygorithm.data_structures.queue.Deque method), 9
- delta (pygorithm.geometry.line2.Line2 attribute), 32
- Deque (class in pygorithm.data_structures.queue), 9
- dequeue() (pygorithm.data_structures.queue.Queue method), 9
- dijkstra.Dijkstra.find_path() (built-in function), 56
- dijkstra.get_code() (built-in function), 57
- dot() (pygorithm.geometry.vector2.Vector2 method), 28
- DoublyLinkedList (class in pygorithm.data_structures.linked_list), 10

E

- enqueue() (pygorithm.data_structures.queue.Queue method), 9

F

- factorial.factorial() (built-in function), 55
- factorial.get_code() (built-in function), 55
- favorite() (pygorithm.data_structures.heap.Heap method), 15
- find() (pygorithm.data_structures.tree.BSTNode method), 12
- find_entities_per_depth() (pygorithm.data_structures.quadtree.QuadTree method), 20
- find_final_node() (pygorithm.data_structures.trie.Trie method), 16
- find_intersection() (pygorithm.geometry.axisall.AxisAlignedLine static method), 37
- find_intersection() (pygorithm.geometry.line2.Line2 static method), 35
- find_intersection() (pygorithm.geometry.polygon2.Polygon2 static method), 42
- find_intersection() (pygorithm.geometry.rect2.Rect2 class method), 45
- find_nodes_per_depth() (pygorithm.data_structures.quadtree.QuadTree method), 20
- find_words() (pygorithm.data_structures.trie.Trie method), 16
- from_regular() (pygorithm.geometry.polygon2.Polygon2 class method), 39

- from_rotated() (pygorithm.geometry.polygon2.Polygon2 class method), 40

G

- get_code() (built-in function), 24
- get_code() (in module pygorithm.dynamic_programming.binary_knapsack), 23
- get_code() (in module pygorithm.dynamic_programming.lis), 23
- get_code() (in module pygorithm.greedy_algorithm.activity_selection), 53
- get_code() (in module pygorithm.greedy_algorithm.fractional_knapsack), 53
- get_code() (in module pygorithm.strings.anagram), 65
- get_code() (in module pygorithm.strings.isogram), 66
- get_code() (in module pygorithm.strings.manacher_algorithm), 66
- get_code() (in module pygorithm.strings.palindrome), 66
- get_code() (in module pygorithm.strings.pangram), 66
- get_code() (pygorithm.data_structures.graph.CheckCycleDirectedGraph static method), 14
- get_code() (pygorithm.data_structures.graph.CheckCycleUndirectedGraph static method), 15
- get_code() (pygorithm.data_structures.graph.Graph method), 13
- get_code() (pygorithm.data_structures.graph.TopologicalSort method), 14
- get_code() (pygorithm.data_structures.heap.Heap method), 16
- get_code() (pygorithm.data_structures.linked_list.DoublyLinkedList static method), 10
- get_code() (pygorithm.data_structures.linked_list.Node static method), 10
- get_code() (pygorithm.data_structures.linked_list.SinglyLinkedList static method), 10
- get_code() (pygorithm.data_structures.quadtree.QuadTree static method), 22
- get_code() (pygorithm.data_structures.queue.Deque static method), 9
- get_code() (pygorithm.data_structures.queue.Queue method), 9
- get_code() (pygorithm.data_structures.stack.InfixToPostfix static method), 9
- get_code() (pygorithm.data_structures.stack.Stack static method), 8
- get_code() (pygorithm.data_structures.tree.BinarySearchTree static method), 13
- get_code() (pygorithm.data_structures.tree.BinaryTree static method), 11
- get_code() (pygorithm.data_structures.tree.BSTNode static method), 12

get_code() (pygorithm.data_structures.tree.Node static method), 11
 get_data() (pygorithm.data_structures.linked_list.DoublyLinkedList method), 10
 get_data() (pygorithm.data_structures.linked_list.SinglyLinkedList method), 10
 get_data() (pygorithm.data_structures.tree.BSTNode method), 12
 get_data() (pygorithm.data_structures.tree.Node method), 11
 get_edge_weight() (pygorithm.data_structures.graph.WeightedUndirectedGraph method), 14
 get_left() (pygorithm.data_structures.tree.BSTNode method), 12
 get_left() (pygorithm.data_structures.tree.Node method), 11
 get_quadrant() (pygorithm.data_structures.quadtree.QuadTree method), 19
 get_right() (pygorithm.data_structures.tree.BSTNode method), 12
 get_right() (pygorithm.data_structures.tree.Node method), 11
 get_sequence() (built-in function), 24
 Graph (class in pygorithm.data_structures.graph), 13
 gridify() (pygorithm.data_structures.graph.WeightedUndirectedGraph method), 14

H

Heap (class in pygorithm.data_structures.heap), 15
 heap_sort.get_code() (built-in function), 63
 heap_sort.sort() (built-in function), 63
 heap_sort.time_complexities() (built-in function), 63
 heapify_down() (pygorithm.data_structures.heap.Heap method), 15
 heapify_up() (pygorithm.data_structures.heap.Heap method), 15
 height (pygorithm.geometry.rect2.Rect2 attribute), 44
 horizontal (pygorithm.geometry.line2.Line2 attribute), 33

I

infix_to_postfix() (pygorithm.data_structures.stack.InfixToPostfix method), 9
 InfixToPostfix (class in pygorithm.data_structures.stack), 9
 inorder() (pygorithm.data_structures.tree.BinarySearchTree method), 13
 inorder() (pygorithm.data_structures.tree.BinaryTree method), 11
 inorder() (pygorithm.data_structures.tree.BSTNode method), 12
 insert() (pygorithm.data_structures.heap.Heap method), 15
 insert() (pygorithm.data_structures.tree.BinarySearchTree method), 12
 insert() (pygorithm.data_structures.tree.BinaryTree method), 11
 insert() (pygorithm.data_structures.tree.BSTNode method), 12
 insert() (pygorithm.data_structures.trie.Trie method), 16
 insert_after() (pygorithm.data_structures.linked_list.SinglyLinkedList method), 10
 insert_and_think() (pygorithm.data_structures.quadtree.QuadTree method), 19
 insert_at_end() (pygorithm.data_structures.linked_list.DoublyLinkedList method), 10
 insert_at_end() (pygorithm.data_structures.linked_list.SinglyLinkedList method), 10
 insert_at_start() (pygorithm.data_structures.linked_list.DoublyLinkedList method), 10
 insert_at_start() (pygorithm.data_structures.linked_list.SinglyLinkedList method), 10
 insert_front() (pygorithm.data_structures.queue.Deque method), 9
 insert_rear() (pygorithm.data_structures.queue.Deque method), 9
 insertion_sort.get_code() (built-in function), 63
 insertion_sort.sort() (built-in function), 63
 insertion_sort.time_complexities() (built-in function), 63
 interpolation_search.get_code() (built-in function), 61
 interpolation_search.search() (in module pygorithm.searching), 60
 interpolation_search.time_complexities() (built-in function), 60
 intersects() (pygorithm.geometry.axisall.AxisAlignedLine static method), 36
 is_anagram() (in module pygorithm.strings.anagram), 65
 is_empty() (pygorithm.data_structures.queue.Deque method), 9
 is_empty() (pygorithm.data_structures.queue.Queue method), 9
 is_empty() (pygorithm.data_structures.stack.Stack method), 8
 is_full() (pygorithm.data_structures.queue.Deque method), 9
 is_isogram() (in module pygorithm.strings.isogram), 65
 is_palindrome() (in module pygorithm.strings.palindrome), 66
 is_pangram() (in module pygorithm.strings.pangram), 66

K

knapsack() (in module pygorithm.dynamic_programming.binary_knapsack), 22
 knapsack() (in module pygorithm.greedy_algorithm.fractional_knapsack), 53

- kruskal_code() (pygorithm.data_structures.graph.WeightedGraph class method), 14
- kruskal_mst() (pygorithm.data_structures.graph.WeightedGraph method), 13
- kruskal_time_complexity() (pygorithm.data_structures.graph.WeightedGraph static method), 13
- ## L
- lcm.get_code() (built-in function), 54
- lcm.lcm() (built-in function), 54
- left_child_idx() (pygorithm.data_structures.heap.Heap static method), 15
- Line2 (class in pygorithm.geometry.line2), 31
- linear_search.get_code() (built-in function), 59
- linear_search.search() (built-in function), 59
- linear_search.time_complexities() (built-in function), 59
- longest_increasing_subsequence() (in module pygorithm.dynamic_programming.lis), 23
- ## M
- magnitude (pygorithm.geometry.line2.Line2 attribute), 32
- magnitude() (pygorithm.geometry.vector2.Vector2 method), 31
- magnitude_squared (pygorithm.geometry.line2.Line2 attribute), 32
- magnitude_squared() (pygorithm.geometry.vector2.Vector2 method), 30
- manacher() (in module pygorithm.strings.manacher_algorithm), 66
- max_x (pygorithm.geometry.line2.Line2 attribute), 32
- max_y (pygorithm.geometry.line2.Line2 attribute), 33
- merge_sort.get_code() (built-in function), 64
- merge_sort.sort() (built-in function), 63
- merge_sort.time_complexities() (built-in function), 63
- min_val_bst_node() (pygorithm.data_structures.tree.BSTNode static method), 12
- min_x (pygorithm.geometry.line2.Line2 attribute), 32
- min_y (pygorithm.geometry.line2.Line2 attribute), 32
- ## N
- Node (class in pygorithm.data_structures.linked_list), 10
- Node (class in pygorithm.data_structures.tree), 11
- normal (pygorithm.geometry.line2.Line2 attribute), 32
- normalize() (pygorithm.geometry.vector2.Vector2 method), 30
- number_of_nodes() (pygorithm.data_structures.tree.BinarySearchTree method), 13
- number_of_nodes() (pygorithm.data_structures.tree.BinaryTree method), 11
- ## P
- parent_idx() (pygorithm.data_structures.heap.Heap static method), 15
- peek() (pygorithm.data_structures.stack.Stack method), 8
- polygon (pygorithm.geometry.rect2.Rect2 attribute), 43
- Polygon2 (class in pygorithm.geometry.polygon2), 38
- pop() (pygorithm.data_structures.heap.Heap method), 15
- pop() (pygorithm.data_structures.stack.Stack method), 8
- postorder() (pygorithm.data_structures.tree.BinarySearchTree method), 13
- postorder() (pygorithm.data_structures.tree.BinaryTree method), 11
- postorder() (pygorithm.data_structures.tree.BSTNode method), 12
- preorder() (pygorithm.data_structures.tree.BinarySearchTree method), 13
- preorder() (pygorithm.data_structures.tree.BinaryTree method), 11
- preorder() (pygorithm.data_structures.tree.BSTNode method), 12
- print_graph() (pygorithm.data_structures.graph.CheckCycleDirectedGraph method), 14
- print_graph() (pygorithm.data_structures.graph.CheckCycleUndirectedGraph method), 15
- print_graph() (pygorithm.data_structures.graph.Graph method), 13
- print_graph() (pygorithm.data_structures.graph.WeightedGraph method), 13
- project_onto_axis() (pygorithm.geometry.polygon2.Polygon2 static method), 41
- project_onto_axis() (pygorithm.geometry.rect2.Rect2 static method), 44
- push() (pygorithm.data_structures.stack.Stack method), 8
- pygorithm.binary.ascii (module), 4
- pygorithm.binary.base10 (module), 5
- pygorithm.binary.base16 (module), 6
- pygorithm.binary.base2 (module), 4
- pygorithm.data_structures.graph (module), 13
- pygorithm.data_structures.heap (module), 15
- pygorithm.data_structures.linked_list (module), 10
- pygorithm.data_structures.quadtree (module), 16
- pygorithm.data_structures.queue (module), 9
- pygorithm.data_structures.stack (module), 8
- pygorithm.data_structures.tree (module), 11
- pygorithm.data_structures.trie (module), 16
- pygorithm.dynamic_programming.binary_knapsack (module), 22
- pygorithm.dynamic_programming.lis (module), 23
- pygorithm.geometry.extrapolated_intersection (module), 47
- pygorithm.greedy_algorithm.activity_selection (module), 53

pygorithm.greedy_algorithm.fractional_knapsack (module), 53
 pygorithm.strings.anagram (module), 65
 pygorithm.strings.isogram (module), 65
 pygorithm.strings.manacher_algorithm (module), 66
 pygorithm.strings.palindrome (module), 66
 pygorithm.strings.pangram (module), 66

Q

QuadTree (class in pygorithm.data_structures.quadtree), 17
 QuadTreeEntity (class in pygorithm.data_structures.quadtree), 16
 Queue (class in pygorithm.data_structures.queue), 9
 quick_select.get_code() (built-in function), 60
 quick_select.search() (built-in function), 60
 quick_select.time_complexities() (built-in function), 60
 quick_sort.get_code() (built-in function), 64
 quick_sort.sort() (built-in function), 64
 quick_sort.time_complexities() (built-in function), 64

R

Rect2 (class in pygorithm.geometry.rect2), 43
 remove_edge() (pygorithm.data_structures.graph.WeightedUndirectedGraph method), 14
 retrieve_collidables() (pygorithm.data_structures.quadtree.QuadTree method), 19
 right_child_idx() (pygorithm.data_structures.heap.Heap static method), 15
 rotate() (pygorithm.geometry.vector2.Vector2 method), 29

S

search() (pygorithm.data_structures.trie.Trie method), 16
 selection_sort.get_code() (built-in function), 64
 selection_sort.sort() (built-in function), 64
 selection_sort.time_complexities() (built-in function), 64
 set_data() (pygorithm.data_structures.tree.BSTNode method), 12
 set_data() (pygorithm.data_structures.tree.Node method), 11
 set_left() (pygorithm.data_structures.tree.BSTNode method), 12
 set_left() (pygorithm.data_structures.tree.Node method), 11
 set_right() (pygorithm.data_structures.tree.BSTNode method), 12
 set_right() (pygorithm.data_structures.tree.Node method), 11
 shell_sort.get_code() (built-in function), 64
 shell_sort.sort() (built-in function), 64
 shell_sort.time_complexities() (built-in function), 64
 sieve_of_eratostenes.get_code() (built-in function), 54

sieve_of_eratostenes.sieve_of_eratostenes() (built-in function), 54
 SinglyLinkedList (class in pygorithm.data_structures.linked_list), 10
 size() (pygorithm.data_structures.queue.Queue method), 9
 size() (pygorithm.data_structures.stack.Stack method), 8
 slope (pygorithm.geometry.line2.Line2 attribute), 33
 split() (pygorithm.data_structures.quadtree.QuadTree method), 18
 Stack (class in pygorithm.data_structures.stack), 8
 sum_entities() (pygorithm.data_structures.quadtree.QuadTree method), 20

T

think() (pygorithm.data_structures.quadtree.QuadTree method), 18
 to_ascii() (in module pygorithm.binary.base16), 6
 to_ascii() (in module pygorithm.binary.base2), 4
 to_base10() (in module pygorithm.binary.base16), 6
 to_base10() (in module pygorithm.binary.base2), 4
 to_base16() (in module pygorithm.binary.ascii), 4
 to_base16() (in module pygorithm.binary.base10), 5
 to_base16() (in module pygorithm.binary.base2), 5
 to_base2() (in module pygorithm.binary.ascii), 4
 to_base2() (in module pygorithm.binary.base10), 5
 to_base2() (in module pygorithm.binary.base16), 6
 topological_sort() (pygorithm.data_structures.graph.TopologicalSort method), 14
 TopologicalSort (class in pygorithm.data_structures.graph), 14
 Trie (class in pygorithm.data_structures.trie), 16

V

Vector2 (class in pygorithm.geometry.vector2), 25
 vertical (pygorithm.geometry.line2.Line2 attribute), 33

W

WeightedGraph (class in pygorithm.data_structures.graph), 13
 WeightedUndirectedGraph (class in pygorithm.data_structures.graph), 14
 width (pygorithm.geometry.rect2.Rect2 attribute), 43

Y

y_intercept (pygorithm.geometry.line2.Line2 attribute), 33