# PyDSL

*Release 0.01*

**Ian Wu**

**Sep 30, 2020**

# CONTENTS

CHAPTER

ONE

# INTRODUCTION

PyDSL is a data structures library, written in Python for Python. It aims to provide an efficient implementation and a convenient interface for each data structure, as well as a selection of useful methods and algorithms to manipulate them.

PyDSL is still very much in development - see the TODO section. We welcome open-source contributions to help improve this library.

## 1.1 Installation Guide

Prerequisites: - Python3.6 or later

Installation using pip:

```
pip install PyDSALib==0.1
```

Clone from GitHub:

```
git clone https://github.com/HerrHruby/PyDSL.git
```

## 1.2 Tutorials

## 1.3 TODO List

- URGENT: Unit testing for ALL modules
- Improve documentation: - Include module docstrings for all modules - Provide examples of use for every module - Formatting
- Create tutorials
- Rewrite binary_tree to use stack instead of recursion, to improve scalability
- Create a vector (in the maths sense) class for efficient vector manipulation?
- Implement delete method for AVL_tree
- Graph pretty printer (quite challenging)

CHAPTER

TWO

# DATA STRUCTURES

## 2.1 Queue

**class** PyDSL.queue.**Queue**
Bases: *PyDSL.linked_list.LinkedList*

Queue class (first-in first-out). Extends LinkedList

**head**
Node object. The head of the linked list

**size**
the length of the linked list

**tail**
Node object. The tail of the linked list

**dequeue**()
Remove and return item from the queue

**enqueue**(*item*)
Insert item into the queue

**peek**()
Return the next item from the queue

## 2.2 Deque

**class** PyDSL.deque.**Deque**
Bases: *PyDSL.linked_list.LinkedList*

Node class for deque. Extends LinkedList

**head**
Node object. The head of the deque

**tail**
Node object. The tail of the deque

**size**
the length of the deque

**left_dequeue**()
Remove an item from the left of the deque and return it

**left_enqueue**(*item*)
Add an item to the left of the deque

**right_dequeue**()
Remove an item from the right of the deque and return it

**right_enqueue**(*item*)
Add an item to the right of the deque

## 2.3 Stack

**class** PyDSL.stack.**Stack**
Bases: *PyDSL.linked_list.LinkedList*

Stack class (last-in first-out). Extends LinkedList

**head**
Node object. The head of the linked list

**size**
the length of the linked list

**peek**()
Return the next item from the stack

**pop**()
Remove and return item

**push**(*item*)
Add item to the stack

## 2.4 Linked List

Linked list class

**class** PyDSL.linked_list.**LinkedList**
Bases: object

Node class for linked list

**head**
Node object. The head of the linked list

**size**
the length of the linked list

**get_size**()
Get the size of the LinkedList

**insert**(*item*, *index*)
Insert a node at an index

**is_empty**()
Check if the linked list is empty

**remove**(*item*)
Remove a node with matching data

**search**(*item*)
Search for a node with matching data. Return True if found, False if not

**class** PyDSL.linked_list.**Node**(*init_data*)
Bases: object

Node class for linked list

**data**
the data contained inside the node

**next**
Node object. The next node in the linked list

**previous**
Node object. The previous node in the linked list

**get_data**()
Get data of node

**get_next**()
Get the next node

**get_prev**()
Get the previous node

**set_data**(*input_data*)
Set data for the node

**set_next**(*input_next*)
Set the next node

**set_prev**(*input_prev*)
Set the previous node

## 2.5 Binary Heap

**class** PyDSL.heap.**HeapNode**(*data*)
Bases: object

HeapNode class. Forms the elements of the MaxHeap and MinHeap classes

**data**
the data contained in the HeapNode. Can be a float/int or a tuple, with the zeroth element of the

**tuple acting as the key**

**get_data**()
Get the data contained in the HeapNode

**set_data**(*item*)
Set the data contained in the HeapNode

**class** PyDSL.heap.**MaxHeap**
Bases: object

Class for a max heap - a heap that prioritises the biggest item

**heap_list**
the list of HeapNodes. The first element is always an empty node that is ignored by operations

**heap_size**
the number of elements in the heap. Starts at 0

**build**(*new_list*)
Construct a MaxHeap given a list of numbers or tuples

**delete_item**(*item_index*)
Delete a HeapNode in a specific position (index) within heap_list

**heap_change_key**(*val*, *new_key*)
For heaps containing HeapNodes of tuples - change the keys of items with values val

**insert**(*item*)
Insert an item into the heap

**pop**()
Remove and return the HeapNode of highest priority

**class** PyDSL.heap.**MinHeap**
Bases: object

Class for a min heap - a heap that prioritises the smallest item

**heap_list**
the list of HeapNodes. The first element is always an empty node that is ignored by operations

**heap_size**
the number of elements in the heap. Starts at 0

**build**(*new_list*)
Construct a MaxHeap given a list of numbers or tuples

**delete_item**(*item_index*)
Delete a HeapNode in a specific position (index) within heap_list

**heap_change_key**(*val*, *new_key*)
For heaps containing HeapNodes of tuples - change the keys of items with values val

**insert**(*item*)
Insert an item into the heap

**pop**()
Remove and return the HeapNode of highest priority

## 2.6 Binary Search Tree

**class** PyDSL.binary_tree.**BinaryTree**
Bases: object

Binary tree class

**root**
the root node of the tree (default = None)

**size**
the number of elements in the binary tree (default = 0)

**delete**(*key*)
Deletes a node with matching key

**get_height**(*key*)
Calculates and returns the height of a node with matching key. Root height = 0

**get_node**(*key*)
Returns the value of the node with a matching key

**get_size**()
Getter method for size

**insert_node**(*key*, *val*)
Inserts a new node into the tree at the correct location

**class** PyDSL.binary_tree.**TreeNode**(*key*, *val*, *left_child=None*, *right_child=None*, *parent=None*)
Bases: object

Node class for binary tree

**key**
key of the node

**val**
value of the node

**left_child**
TreeNode object. The left child of the current node

**right_child**
TreeNode object. The right child of the current node

**parent**
TreeNode object. The parents of the current node

**find_min**()
Finds the node with minimum key in the tree

**get_key**()
Getter method for key

**get_left_child**()
Getter method for the left child

**get_right_child**()
Getter method for the right child

**get_successor**()
Calls find_min() to get the successor node

**get_val**()
Getter method for val

**has_left_child_only**()
Check if the current node has a left child only

**has_right_child_only**()
Check if the current node has a right child only

**is_leaf**()
Check if the current node is a leaf

**set_key**(*item*)
Setter method for key

**set_val**(*item*)
Setter method for val

**splice**()
Splices out a node

## 2.7 AVL Tree

**class** PyDSL.AVL_tree.**AVLNode**(*key*, *val*, *left_child=None*, *right_child=None*, *parent=None*)

Bases: *PyDSL.binary_tree.TreeNode*

Node class for AVL (self-balancing) tree. Extends the TreeNode class

**key**
key of the node

**val**
value of the node

**left_child**
TreeNode object. The left child of the current node

**right_child**
TreeNode object. The right child of the current node

**parent**
TreeNode object. The parents of the current node

**balance_factor**
the balance factor of the node

**class** PyDSL.AVL_tree.**AVLTree**

Bases: *PyDSL.binary_tree.BinaryTree*

AVL tree class. Extends BinaryTree

**root**
the root node of the tree (default = None)

**size**
the number of elements in the binary tree (default = 0)

**insert_node**(*key*, *val*)
Insert a new node into the tree at the correct location. Overrides insert_node method in BinaryTree

**rebalance**(*node*)
Rebalance the tree

**rotate_left**(*org_root*)
Implement left rotation of a node

**rotate_right**(*org_root*)
Implement right rotation of a node

**update_balance**(*node*)
Update the balance factor of every node, and rebalance the tree if necessary

## 2.8 Graph

**class** PyDSL.graph.**Graph**
Bases: object

Graph class

**node_list**
a list of all nodes belonging to the graph

**size**
the number of nodes in the graph

**add_edge**(*key*, *edge*, *weight*)
Add an edge between two nodes (from key to edge) with specified weight

**add_node**(*key*)
Add a node to the graph

**add_undirected_edge**(*key*, *edge*, *weight*)
Add an undirected edge between two nodes (from key to edge) with specified weight

**get_node**(*key*)
Get the node in the graph with corresponding key

**get_nodes**()
Get the keys of all the nodes in the graph

**class** PyDSL.graph.**Node**(*key*)
Bases: object

Node class for graphs

**key**
key of the node

**connections**
a dictionary containing the neighbours of the node and the weight of the edge as a key-val pair

**colour**
flag for search algorithms. Default = 'white'

**add_nbr**(*nbr*, *weight*)
Add a neighbour, and specify the weight of the edge

**get_colour**()
Get the colour of the node

**get_key**()
Get the key of the node

**get_nbrs**()
Get the keys of all the neighbours in a list

**get_weight**(*nbr*)
Get the weight of the edge connecting the node and a neighbour

**set_colour**(*col*)
Set the colour of the node

CHAPTER

THREE

# ALGORITHMS

## 3.1 Binary Tree Algorithms

PyDSL.binary_tree_algorithms.**bfs**(*tree*)
Perform breadth-first search of the tree.

**Parameters** **tree** – the tree to perform BFS on

**Returns** A list of nodes in order of the search

PyDSL.binary_tree_algorithms.**inorder**(*tree*)
Perform inorder tree traversal.

**Parameters** **tree** – the tree to perform inorder traversal on

**Returns** A list of nodes in order of the search

PyDSL.binary_tree_algorithms.**postorder**(*tree*)
Perform postorder tree traversal.

**Parameters** **tree** – the tree to perform postorder traversal on

**Returns** A list of nodes in order of the search

PyDSL.binary_tree_algorithms.**preorder**(*tree*)
Perform preorder tree traversal.

**Parameters** **tree** – the tree to perform preorder traversal on

**Returns** A list of nodes in order of the search

## 3.2 Graph Algorithms

**class** PyDSL.graph_algorithms.**DistanceNode**(*key*)
Bases: *PyDSL.graph.Node*

Node class for graph algorithms requiring distance and predecessor attributes. Extends the graph node class

**key**
key of the node

**connections**
a dictionary containing the neighbours of the node and the weight of the edge as a key-val pair

**colour**
flag for search algorithms. Default = 'white'

**distance**
the distance attribute. Default = 0

**pred**
the predecessor node. Default = None

**get_distance**()
Get the distance of the node

**get_pred**()
Get the predecessor node of the node

**set_distance**(*dist*)
Set the distance of the node

**set_pred**(*node*)
Set the predecessor node of the node

PyDSL.graph_algorithms.**bfs**(*graph*)
Perform breadth-first search on the graph.

**Parameters** **graph** – the graph to perform BFS on

**Returns** Returns a list of spanning trees (graphs)

PyDSL.graph_algorithms.**dfs**(*graph*)
Perform depth-first search on the graph.

**Parameters** **graph** – the graph to perform DFS on

**Returns** Returns a list of spanning trees (graphs)

PyDSL.graph_algorithms.**dijkstra**(*graph*, *start_node*)
Perform Dijkstra's Algorithm on the graph, given a starting node.

**Parameters**

- **graph** – the graph to perform Dijkstra on
- **start_node** – the starting node, relative to which distances are found

**Returns** Returns a graph of DistanceNode objects, with the predecessor of each node corresponding to the shortest path from said node to the starting node

PyDSL.graph_algorithms.**prim**(*graph*)
Performs Prim's Algorithm on the graph.

**Parameters** **graph** – the graph to find the MST of a graph

**Returns** A list of minimum spanning trees (graphs)

PyDSL.graph_algorithms.**topological_sort**(*graph*)
Performs topological sort on a directed acyclic graph.

**Parameters** **graph** – the graph to perform topological sort on

**Returns** A list of sorted nodes

CHAPTER

# FOUR

# TOOLS

## 4.1 Pretty Printer

PyDSL.pretty_printer.**pretty_print_graph**(*graph*)

PyDSL.pretty_printer.**pretty_print_tree**(*tree*, *layer_height=2*, *print_fn=<function <lambda>>*)

CHAPTER

FIVE

# INDICES AND TABLES

- genindex
- search

# PYTHON MODULE INDEX

## p

# INDEX

## I



## K



## L



## M



## N



## P

## Q



## R



## S



## T



## U



## V