Pypleine Documentation

Contents

Module Pypeline

Sub-modules

- Pypeline.Pypes
- Pypeline.profile
- Pypeline.utils

Module Pypeline.Pypes

Sub-modules

- Pypeline.Pypes.AVLTree
- Pypeline.Pypes.BinarySearchTree

- Pypeline.Pypes.DoublyLinkedList
- Pypeline.Pypes.Heap
- Pypeline.Pypes.LinkedList
- Pypeline.Pypes.MaxHeap
- Pypeline.Pypes.MinHeap
- Pypeline.Pypes.Node
- Pypeline.Pypes.Queue
- Pypeline.Pypes.RedBlackBST
- Pypeline.Pypes.Sorting
- Pypeline.Pypes.Stack
- Pypeline.Pypes.StringSearch
- Pypeline.Pypes.TernarySearchTree
- Pypeline.Pypes.Trie

Module `Pypeline.Pypes.AVLTree`

Implementation of the AVL tree !!

Classes

Class `AVL`

```
class AVL
```

Implementation of the AVL Tree (https://en.wikipedia.org/wiki/AVL_tree)

Attributes ——= **`root`** : The root node of the AVL Tree

Constructor of the AVL class

Methods

Method `calcBalance`

```
def calcBalance(
    self,
    node: Optional[Pypeline.Pypes.AVLTree.Node]
) -> int
```

Method used to determine whether a particular node is balanced

if return value > 1, it means left heavy situations -> right rotation

if return value < -1, it means right heavy situations -> left rotation

otherwise it is balanced

Parameters ——=

node: the node that will have its level of balance calculated

Method `calcHeight`

```
def calcHeight(
    self,
    node: Optional[Pypeline.Pypes.AVLTree.Node]
) -> int
```

Method used to determine the height of a particular node

Parameters ——=

node: the node for which the height is to be calculated

Method `insert`

```
def insert(
    self,
    data: Any
) -> NoneType
```

Public method used to insert data into the AVL tree

A check is necessary to ensure the AVL property is not violated. This check is performed within the associated helper method ___insertNode

Parameters ——=

data: the data to be added to the tree

Method `remove`

```
def remove(
    self,
    data: Any
) -> NoneType
```

Public method used to remove some piece of data from the tree

Parameters ——=

data: the data to be removed to the tree

Method `rotateLeft`

```
def rotateLeft(
    self,
    node: Pypeline.Pypes.AVLTree.Node
) -> Pypeline.Pypes.AVLTree.Node
```

Method used to perform a left rotation from a particular node.

Rotations to the right and the left are symmetrical operations. Rotation operations are quite fast as it it just updating references O(1) time complexity.

Parameters ——=

node: the node that will be reference rotatation point

Method `rotateRight`

```
def rotateRight(
    self,
    node: Pypeline.Pypes.AVLTree.Node
) -> Pypeline.Pypes.AVLTree.Node
```

Method used to perform a right rotation from a particular node.

Rotations to the right and the left are symmetrical operations. Rotation operations are quite fast as it it just updating references O(1) time complexity.

Parameters ——=

node: the node that will be reference rotatation point

Method `traverse`

```
def traverse(
    self
) -> NoneType
```

Method used to perform an inOrderTraversal

Class `Node`

```
class Node(
    data: Any
)
```

This is a custom node class for the AVL Tree

Attributes ——= **`data`** : The data to be stored in the node

`rightChild` The right child of the current node
`leftChild` The left child of the current node
`height` The length of longest path from current not to lead. Used to check if the tree is balanced.

Constructor of the Node class

Parameters ——=

data: the data to be stored in the Node

Module `Pypeline.Pypes.BinarySearchTree`

Implementation of the binary search tree

Classes

Class `BST`

```
class BST
```

Implementation of the Binary Search Tree (https://en.wikipedia.org/wiki/Binary_search_tree)

All items on the left are smaller than given node & all on the right are larger than the given node

Attributes ——= **`root`** : The root node of the BST Tree

`size` counter for the number of items inside the tree

Constructor of the BST class

Methods

Method `getMaxValue`

```
def getMaxValue(
    self
) -> Union[int, float, NoneType]
```

Public method used to get the largest value in the BST

Method `getMinValue`

```
def getMinValue(
    self
) -> Union[int, float, NoneType]
```

Public method used to get the smallest value in the BST

Method `getSize`

```
def getSize(
    self
) -> int
```

Method used to return the size of the BST

Method `inOrderTraversal`

```
def inOrderTraversal(
    self
) -> NoneType
```

Public method used to perform an inOrderTraversal

Method `insert`

```
def insert(
    self,
    data: Union[int, float]
) -> NoneType
```

Public method used to insert data into the BST tree

Parameters ——=

data: the data to be added to the tree

Method `postOrderTraversal`

```
def postOrderTraversal(
    self
) -> NoneType
```

Public method used to perform an postOrderTraversal

Method `preOrderTraversal`

```
def preOrderTraversal(
    self
) -> NoneType
```

Public method used to perform an preOrderTraversal

Method `remove`

```
def remove(
    self,
    data: Union[int, float]
) -> NoneType
```

Public method used to remove data from the BST tree

Parameters ——=

data: the data to be removed from the tree

Class `Node`

```
class Node(
    data: Union[int, float]
)
```

This is a custom node class for the Binary Search Tree

Attributes ——= **`data`** : The data to be stored in the node

`rightChild` The right child of the current node
`leftChild` The left child of the current node

Constructor of the Node class

Parameters ——=

data: the data to be stored in the Node

Module `Pypeline.Pypes.DoublyLinkedList`

Implementation of the Doubly Linked List

@author Chenghao Gong gongc12

Classes

Class `DoublyLinkedList`

```
class DoublyLinkedList
```

The constructor for the doubly linked list

Attributes —-= **`head`** : The head node in the linked list

Methods

Method `append`

```
def append(
    self,
    new_data
)
```

Method `contains`

```
def contains(
    self,
    element
)
```

Method `get`

```
def get(
    self,
    index
)
```

Method `getFirst`

```
def getFirst(
    self
)
```

Method `getLast`

```
def getLast(
    self
)
```

Method `insertAfter`

```
def insertAfter(
    self,
    prev_node,
    new_data
)
```

Method `printList`

```
def printList(
    self,
    node
)
```

Method `push`

```
def push(
    self,
    new_data
)
```

Method `remove`

```
def remove(
    self,
    index
)
```

Method `sizeOf`

```
def sizeOf(
    self
)
```

Class `Node`

```
class Node(
    data
)
```

This is a custom node class for the Doubly Linked List

Attributes ——= **`data`** : The data to be stored in the node

`next` The next node in the linked list
`prev` The previous node in the linked list

Module `Pypeline.Pypes.Heap`

Implementation of the heap

Classes

Class `Heap`

```
class Heap(
    size: int = 10
)
```

Implementation of a Heap (https://en.wikipedia.org/wiki/Heap_(data_structure))

Attributes ——= **`heap`** : the underlying list data sturcture for the heap

`currentPosition` used to keep track of the index in the heap

Constructor of the Heap class

Parameters ——=

size: desired size of the Heap

Class variables

Variable `HEAP_SIZE` Type: `ClassVar[int]`

Methods

Method `heapSort`

```
def heapSort(
    self
) -> NoneType
```

Public method used to sort the heap

Method `insert`

```
def insert(
    self,
    item: int
) -> NoneType
```

Public method used to insert an item into the Heap

Parameters —=

item: the data to be added to the heap

Method `isFull`

```
def isFull(
    self
) -> bool
```

Public method used to determine if the heap is full

Module `Pypeline.Pypes.LinkedList`

Implementation of the linked list

Classes

Class `LinkedList`

```
class LinkedList
```

Implementation of the linked list (https://en.wikipedia.org/wiki/Linked_list)

Methods

Method `getSize`

```
def getSize(
    self
) -> int
```

Method `getSize2`

```
def getSize2(
    self
) -> int
```

Method `insert`

```
def insert(
    self,
    data: Any
) -> NoneType
```

Method `insertEnd`

```
def insertEnd(
    self,
    data: Any
) -> NoneType
```

Method `remove`

```
def remove(
    self,
    data: Any
) -> NoneType
```

Method `to_array`

```
def to_array(
    self
) -> Any
```

Method `to_list`

```
def to_list(
    self
) -> List[Any]
```

Method `traverseList`

```
def traverseList(
    self
) -> NoneType
```

Class `Node`

```
class Node(
    data: Any
)
```

This is a custom node class for the Linked List

Attributes ——= **`data`** : The data to be stored in the node

`nextNode` Reference to the next node

Module `Pypeline.Pypes.MaxHeap`

Implementation of the Max Heap

Classes

Class `MaxHeap`

```
class MaxHeap(
    li: List[Union[float, int]] = []
)
```

Implementation of the Max Heap (https://en.wikipedia.org/wiki/Min-max_heap)

For any given node C, if P is a parent node of C, then the key (the value) of P is greater than or equal to the key of C

Methods

Method `heapify`

```
def heapify(
    self,
    li: List[Union[float, int]]
) -> NoneType
```

Method `heappop`

```
def heappop(
    self
) -> Any
```

Method `heappush`

```
def heappush(
    self,
    val: Union[float, int]
) -> NoneType
```

Method `merge`

```
def merge(
    self,
    li: List[Union[float, int]]
) -> NoneType
```

Method `to_array`

```
def to_array(
    self
) -> Any
```

Module `Pypeline.Pypes.MinHeap`

Implementation of the Min Heap

Classes

Class `Backwards`

```
class Backwards(
    ...
)
```

int([x]) -> integer int(x, base=10) -> integer

Convert a number or string to an integer, or return 0 if no arguments are given. If x is a number, return x.__int__(). For floating point numbers, this truncates towards zero.

If x is not a number or if base is given, then x must be a string, bytes, or bytearray instance representing an integer literal in the given base. The literal can be preceded by '+' or '-' and be surrounded by whitespace. The base defaults to 10. Valid bases are 0 and 2-36. Base 0 means to interpret the base from the string as an integer literal.

```
>>> int('0b100', base=0)
4
```

Ancestors (in MRO)

- builtins.int

Class `MinHeap`

```
class MinHeap(
    li: List[int] = []
)
```

Implementation of the Min Heap (https://en.wikipedia.org/wiki/Heap_(data_structure))

For any given node C, if P is a parent node of C, the key of P is less than or equal to the key of C

Methods

Method `heapify`

```
def heapify(
    self,
    li: List[int]
) -> NoneType
```

Method `heappop`

```
def heappop(
    self
) -> float
```

Method `heappush`

```
def heappush(
    self,
    val: int
) -> NoneType
```

Method `merge`

```
def merge(
    self,
    li: List[int]
) -> NoneType
```

Method `to_array`

```
def to_array(
    self
) -> Any
```

Method `to_list`

```
def to_list(
    self
) -> List[int]
```

Class `Restore`

```
class Restore(
    ...
)
```

int([x]) -> integer int(x, base=10) -> integer

Convert a number or string to an integer, or return 0 if no arguments are given. If x is a number, return x.__int__(). For floating point numbers, this truncates towards zero.

If x is not a number or if base is given, then x must be a string, bytes, or bytearray instance representing an integer literal in the given base. The literal can be preceded by '+' or '-' and be surrounded by whitespace. The base defaults to 10. Valid bases are 0 and 2-36. Base 0 means to interpret the base from the string as an integer literal.

```
>>> int('0b100', base=0)
4
```

Ancestors (in MRO)

- builtins.int

Module `Pypeline.Pypes.Node`

Implementation of the node object

Classes

Class `Node`

```
class Node(
    val: Any = None
)
```

Base for node object containing node value and visited state. val -> Any data type visited -> bool

Methods

Method `getVal`

```
def getVal(
    self
) -> Any
```

Method `isVisited`

```
def isVisited(
    self
) -> bool
```

Method `setVal`

```
def setVal(
    self,
    val
) -> NoneType
```

Method `setVisited`

```
def setVisited(
    self
) -> NoneType
```

Method `swapVisited`

```
def swapVisited(
    self
) -> NoneType
```

Module `Pypeline.Pypes.Queue`

Implementation of the Queue Data Stucture

FIFO Structure -> First in First Out

Classes

Class `Queue`

```
class Queue(
    maxlen: int = 1000000000.0
)
```

Methods

Method `dequeue`

```
def dequeue(
    self
) -> Any
```

Method `enqueue`

```
def enqueue(
    self,
    data: Any
) -> NoneType
```

Method `isEmpty`

```
def isEmpty(
    self
) -> bool
```

Method `peek`

```
def peek(
    self
) -> Any
```

Method `sizeQueue`

```
def sizeQueue(
    self
) -> int
```

Method `to_array`

```
def to_array(
    self
) -> Any
```

Method `to_list`

```
def to_list(
    self
) -> List[int]
```

Module `Pypeline.Pypes.RedBlackBST`

Functions

Function `cmp`

```
def cmp(
    v1,
    v2
)
```

A helper function to compare the two values

Parameters —=

v1: first value v2: second value

Function `flipColors`

```
def flipColors(
    h
)
```

A helper function to flip the colors of the tree by changing the color of each node

Parameters —=

h: a node in the tree

Function `isRed`

```
def isRed(
    x
)
```

A helper function to determine if the link is red or not

Parameters —=

x: a node in the tree

Function `rotateLeft`

```
def rotateLeft(
    h
)
```

A helper function to restore the red-black tree order by changing the color of each node and added their sizes together

Parameters —=

h: a node in the tree

Function `rotateRight`

```
def rotateRight(
    h
)
```

A helper function to restore the red-black tree order by changing the color of each node and added their sizes together

Parameters —=

h: a node in the tree

Function `size`

```
def size(
    root
)
```

A helper function to determine the size of tree

Parameters —=

root: a root in the tree

Classes

Class `Node`

```
class Node(
    key,
    val,
    size,
    color
)
```

This is a custom node class for the Red Black BST

Attributes —= **`key`** : the key used for comparison

`size` the size of the tree
`color` True indicates Red and False indicate a black link
`val` the data stored in the node
`left` Left node child
`right` Right node child

Class `redBlackBST`

```
class redBlackBST
```

Implementation of the Red Black BST

Attributes —= **`root`** : value is set to None first unless Put is called

Methods

Method `actualPut`

```
def actualPut(
    self,
    h,
    key,
    val
) -> Pypeline.Pypes.RedBlackBST.Node
```

A method that puts the node into the BST calling different helper function to ensures the Red black order is preserved

Parameters —=

key: the key used for comparison val: the value of the node being put h: the node we want to put at

Method `get`

```
def get(
    self,
    key
)
```

Method `inorder`

```
def inorder(
    self,
    root
)
```

Method `put`

```
def put(
    self,
    key,
    val
)
```

A method that puts the node into the BST calling another helper method to finish the heavy lifting work

Parameters —=

key: the key used for comparison val: the value of the node being put

Module `Pypeline.Pypes.Sorting`

Sub-modules

- Pypeline.Pypes.Sorting.Helper
- Pypeline.Pypes.Sorting.Mergesort
- Pypeline.Pypes.Sorting.Quicksort
- Pypeline.Pypes.Sorting.Radixsort
- Pypeline.Pypes.Sorting.Shellsort
- Pypeline.Pypes.Sorting.three_way_quickSort

Module `Pypeline.Pypes.Sorting.Helper`

Helper functions for the sorting algorithms

Functions

Function `less`

```
def less(
    a,
    b
) -> bool
```

Function `swap`

```
def swap(
    array,
    i,
    j
)
```

Module `Pypeline.Pypes.Sorting.Mergesort`

Mergesort Implementation

Functions

Function `cmp`

```
def cmp(
    L,
    R,
    i,
    j,
    order='ascend'
)
```

Function `merge`

```
def merge(
    arr,
    l,
    m,
    r,
    order
)
```

Function `mergeSort`

```
def mergeSort(
    arr,
    l,
    r,
    order='ascend'
)
```

Module `Pypeline.Pypes.Sorting.Quicksort`

Functions

Function `cmp`

```
def cmp(
    a,
    b,
    order='ascend'
)
```

Function `partition`

```
def partition(
    arr,
    low,
    high,
    order
)
```

Function `quickSort`

```
def quickSort(
    arr,
```

```
    low,
    high,
    order='ascend'
)
```

Module `Pypeline.Pypes.Sorting.Radixsort`

Radixsort Implementation

Functions

Function `countingSort`

```
def countingSort(
    arr,
    exp1
)
```

Performs counting sort.

Parameters ——= arr: the list that contains the value exp1: the exponent

Function `radixSort`

```
def radixSort(
    arr
)
```

Performs radix sort.

Parameters ——= arr: the list that contains the values to be sorted

Module `Pypeline.Pypes.Sorting.Shellsort`

Functions

Function `condition`

```
def condition(
    j,
    gap,
    arr,
    temp,
    order='ascend'
)
```

Function `shellSort`

```
def shellSort(
    arr,
    order='ascend'
)
```

Module `Pypeline.Pypes.Sorting.three_way_quickSort`

Functions

Function `cmp`

```
def cmp(
    a,
```

```
    b,
    order='ascend'
)
```

Function `exch`

```
def exch(
    arr,
    i,
    j
)
```

Function `three_way_quickSort`

```
def three_way_quickSort(
    arr,
    lo,
    hi,
    order='ascend'
)
```

Module `Pypeline.Pypes.Stack`

Implementation of the Stack Data Structure

N.B: - Array representation -> LIFO Structure

Classes

Class `Stack`

```
class Stack
```

Implementation of the Stack Data Structure (https://en.wikipedia.org/wiki/Stack_(abstract_data_type))

An abstract data type that serves as a collection of elements, with two main principal operations: push and pop from the top of the list (most recent values).

Methods

Method `isEmpty`

```
def isEmpty(
    self
) -> bool
```

Method `peek`

```
def peek(
    self
) -> Any
```

Method `pop`

```
def pop(
    self
) -> Any
```

Method `push`

```
def push(
    self,
    data: Any
) -> NoneType
```

Method `sizeStack`

```
def sizeStack(
    self
) -> int
```

Method `to_array`

```
def to_array(
    self
) -> Any
```

Method `to_list`

```
def to_list(
    self
) -> List[Any]
```

Module `Pypeline.Pypes.StringSearch`

Sub-modules

- Pypeline.Pypes.StringSearch.BoyerMoore
- Pypeline.Pypes.StringSearch.KMP

Module `Pypeline.Pypes.StringSearch.BoyerMoore`

BoyerMoore String Search Implementation

Functions

Function `badCharHeuristic`

```
def badCharHeuristic(
    string,
    size
)
```

The preprocessing function for Boyer Moore's bad character heuristic

Parameters ——=

string: the string pattern size: the length of the string

Function `search`

```
def search(
    txt,
    pat
)
```

The searching part for Boyer Moore algorithm

Parameters ——=

txt: the string text pat: the pattern we are looking for

Module `Pypeline.Pypes.StringSearch.KMP`

KMP Search Implementation

Functions

Function `KMPSearch`

```
def KMPSearch(
    pat,
    txt
)
```

A function implements KMP algorithm string search. Compute the LPS array to locate such pattern.

Parameters —=

pat: the string pattern we are looking for txt: the text we are searching on

Function `computeLPSArray`

```
def computeLPSArray(
    pat,
    M,
    lps
)
```

A method that compute the LPS array. It preprocess the pattern for the kmp search.

Parameters —=

pat: the string pattern we are looking for M: length of the pattern lps: the lps array

Module `Pypeline.Pypes.TernarySearchTree`

Implementation of the Ternay Search Tree

Classes

Class `Node`

```
class Node(
    char: str
)
```

This is a custom node class for the Doubly Linked List

Attributes —= **`char`** : The char to be stored

`leftNode` The left node child
`rightNode` The right node child
`middleNode` The middle node child
`value` The value associated with the char

Class `TST`

```
class TST
```

Implementation of the Ternay Search Tree (https://en.wikipedia.org/wiki/Ternary_search_tree)

A type of trie (sometimes called a prefix tree) where nodes are arranged in a manner similar to a binary search tree, but with up to three children rather than the binary tree's limit of two.

Methods

Method `get`

```
def get(
    self,
    key: int
) -> int
```

Method `getItem`

```
def getItem(
    self,
    node: Optional[Pypeline.Pypes.TernarySearchTree.Node],
    key: str,
    index: int
) -> Optional[Pypeline.Pypes.TernarySearchTree.Node]
```

Method `put`

```
def put(
    self,
    key: str,
    value: int
)
```

Method `putItem`

```
def putItem(
    self,
    node: Optional[Pypeline.Pypes.TernarySearchTree.Node],
    key: int,
    value: Any,
    index: int
) -> Pypeline.Pypes.TernarySearchTree.Node
```

Module `Pypeline.Pypes.Trie`

Implementation of the Trie Data Structure

Classes

Class `Node`

```
class Node(
    char
)
```

This is a custom node class for the Trie

Attributes ——= **`char`** : The char to be stored

`children` Dictionary of the nodes children
`word_finished` Inidicate whether the word is finished at this node
`counter` number of word occurences using this node

Class `Trie`

```
class Trie
```

Implementation of the Trie Data Structure (https://en.wikipedia.org/wiki/Trie)

A type of search tree, a tree data structure used for locating specific keys from within a set

Methods

Method `insert`

```
def insert(
    self,
    word: str
) -> NoneType
```

Method `search`

```
def search(
    self,
    word: str
) -> bool
```

Module `Pypeline.profile`

Classes

Class `ProfileMeta`

```
class ProfileMeta(
    *args,
    **kwargs
)
```

A base class or mixin that enables context managers to work as decorators.

Ancestors (in MRO)

- contextlib.ContextDecorator
- builtins.type

Class `WrappedProfiler`

```
class WrappedProfiler(
    *,
    html: bool = False,
    path: Union[Path, str] = 'profile.html',
    overwrite: bool = True
)
```

Methods

Method `open`

```
def open(
    self: WrappedProfiler
) -> NoneType
```

Class `profile`

```
class profile(
    func: Optional[Callable[..., T]] = None,
    *,
    html: bool = False,
    path: Union[Path, str] = 'profile.html',
    overwrite: bool = True
)
```

A base class or mixin that enables context managers to work as decorators.

Ancestors (in MRO)

- contextlib.ContextDecorator

Module `Pypeline.utils`

Generated by *pdoc* 0.9.2 (https://pdoc3.github.io).