Module `main`

Classes

`class TestTreeSet (methodName='runTest')`

A class whose instances are single test cases.

By default, the test code itself should be placed in a method named 'runTest'.

If the fixture may be used for many test cases, create as many test methods as are needed. When instantiating such a TestCase subclass, specify in the constructor arguments the name of the test method that the instance is to execute.

Test authors should subclass TestCase for their own tests. Construction and deconstruction of the test's environment ('fixture') can be implemented by overriding the 'setUp' and 'tearDown' methods respectively.

If it is necessary to override the **init** method, the base class **init** method must always be called. It is important that subclasses should not change the signature of their **init** method, since instances of the classes are instantiated automatically by parts of the framework in order to be run.

When subclassing TestCase, you can set these attributes: * failureException: determines which exception will be raised when the instance's assertion methods fail; test methods raising this exception will be deemed to have 'failed' rather than 'errored'. * longMessage: determines whether long messages (including repr of objects used in assert methods) will be printed on failure in *addition* to any explicit message passed. * maxDiff: sets the maximum length of a diff in failure messages by assert methods using difflib. It is looked up as an instance attribute so can be configured by individual tests if required.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

Ancestors

unittest.case.TestCase

Methods

`def test_add_different_types(self)`

Test to verify that different types cannot be mixed in the set.

`def test_add_elements(self)`

Test to verify that elements can be added to the set correctly and duplicate verification.

`def test_boundaries_of_ceiling_floor(self)`

Test to verify ceiling and floor at boundaries.

`def test_boundaries_of_higher_lower(self)`

Test to verify higher and lower at boundaries.

`def test_ceiling_floor(self)`

Test to verify the ceiling and floor operations in the set.

`def test_clear_and_empty(self)`

Test to verify that the set can be cleared and checked if it is empty.

`def test_contains(self)`

Test to verify if the set contains certain elements.

`def test_duplicates_handling(self)`

Test to verify that duplicates are not added to the set.

`def test_empty_set_operations(self)`

Test to verify operations on an empty set.

`def test_first_last_elements(self)`

Test to verify that the first and last elements of the set can be obtained.

`def test_higher_lower(self)`

Test to verify the higher and lower operations in the set.

`def test_iterators(self)`

Test to verify that iterators iterate correctly in normal and descending order.

`def test_large_number_of_elements(self)`

Test the performance and correctness with a large number of elements.

`def test_mixed_type_handling(self)`

Test to verify that set does not accept mixed types after initial type is set.

`def test_poll_methods(self)`

Test to verify the functionality of pollFirst and pollLast.

`def test_remove(self)`

Test to verify element removal and that an absent element cannot be removed.

`def test_type_consistency_after_clear(self)`

Test to verify that data type consistency is maintained even after clearing the set.

```
def test_type_error_on_empty_remove(self)
```

Test to verify that an attempt to remove from an empty set is handled correctly.

Index

Classes

Generated by *pdoc* 0.10.0.

Module **`RedBlack`**

Classes

`class Node (value, color='RED')`

Initializes a new node with a specific value and color, defaulting to RED. Also initializes the left, right, and parent node links as None.

`class RedBlackTree`

Initializes a new red-black tree by setting the root to None and the size to 0.

Methods

`def add(self, value)`

Adds a value to the red-black tree. If the value already exists, the function exits without making changes. If the tree is empty, inserts a new black node as the root. Otherwise, inserts a red node and then adjusts the tree to correct red-black properties violations.

`def atIndex(self, index)`

Returns the value of the node at the given index, using an in-order traversal of the tree. This method considers the size of the left subtree to determine the relative position of the index.

`def ceiling(self, value)`

Finds the smallest value in the tree that is greater than or equal to the given value.

`def clear(self)`

Clears the tree by removing all references to the nodes, setting the root to None, and the size to 0.

`def contains(self, value)`

Checks if a specific value exists in the red-black tree. Utilizes a recursive helper method to search through the sub-trees.

`def first(self)`

Returns the value of the first node in the tree, which is the smallest.

`def fix_double_black(self, node)`

Fixes double black violations that may occur after the removal of a black node. This method uses rotations and recolors nodes to restore red-black properties.

`def fix_red_red_violation(self, node)`

Fixes violations of the red-black properties caused after insertion. Colors are adjusted and necessary rotations are performed to maintain the tree's balance.

`def higher(self, e)`

Finds and returns the lowest value in the tree that is greater than the given value. If there is no such value, returns None.

`def left_rotate(self, node)`

Performs a left rotation on a given node, reassigning the node's links, its right child, and the parent to maintain the order of the binary search tree.

`def length(self)`

Returns the size of the tree, i.e., the total number of nodes.

`def pollFirst(self)`

Removes and returns the value of the node with the minimum value in the tree. Uses _min_value_node to find this node and then removes it.

`def pollLast(self)`

Removes and returns the value of the node with the maximum value in the tree. Uses the last method to find this node and then removes it.

`def remove(self, value)`

Removes a node with a specific value from the tree. If the node has two children, it finds the successor to replace it and then removes the successor node.

`def right_rotate(self, node)`

Performs a right rotation on a given node, reassigning the node's links, its left child, and the parent to maintain the order of the binary search tree.

Index

Classes

Module **`TreeSet`**

Classes

`class TreeSet`

Constructor of the TreeSet class.

Initializes a new TreeSet with an empty Red-Black Tree and undefined data type.

Methods

`def add(self, obj)`

Adds an element to the set.

If the data type of the set is not defined, it defines it with the type of the first added element.

Args

`obj`
The object to add to the set.

Returns

True if the object was added successfully, False if the data type does not match the set's type.

`def addAll(self, objList)`

Adds a list of elements to the set.

Args

`objList`
A list of objects to add to the set.

Returns

True after adding all elements.

`def ceiling(self, e)`

Finds the smallest value in the set that is greater than or equal to the given element.

Args

`e`
The element for which the ceiling is sought.

Returns

The smallest value in the set that is greater than or equal to the given element, or None if none.

`def clear(self)`

Removes all elements from the set.

`def clone(self)`

Creates and returns a shallow copy of the set.

Returns

A shallow copy of the set.

`def contains(self, obj)`

Checks if the set contains a given object.

Args

`obj`
The object to check for its presence in the set.

Returns

True if the object is present in the set, False otherwise.

`def descendingIterator(self)`

Returns an iterator to traverse the set in descending order.

Returns

An iterator to traverse the set in descending order.

`def first(self)`

Returns the first value in the set.

Returns

The first value in the set, or None if the set is empty.

`def floor(self, value)`

Finds the largest element in the set that is less than or equal to the given value.

Args

`value`
The value for which the floor is sought.

Returns

The largest element in the set that is less than or equal to the given value, or None if none.

`def higher(self, value)`

Finds the smallest element in the set that is greater than the given value.

Args

`value`

The value for which a greater value is sought.

Returns

The smallest element in the set that is greater than the given value, or None if none.

`def isEmpty(self)`

Checks if the set is empty.

Returns

True if the set is empty, False otherwise.

`def iterator(self)`

Returns an iterator to traverse the set.

Returns

An iterator to traverse the set.

`def last(self)`

Returns the last element of the set.

Returns

The last element of the set, or None if the set is empty.

`def lower(self, e)`

Finds the largest element in the set that is less than the given element.

Args

`e`
The element for which a smaller value is sought.

Returns

The largest element in the set that is less than the given element, or None if none.

`def pollFirst(self)`

Removes and returns the first element of the set.

Returns

The first element of the set, or None if the set is empty.

`def pollLast(self)`

Removes and returns the value of the node with the maximum value in the tree. Uses the last method to find this node and then removes it.

`def raise_type_error(self, obj, supported_datatype)`

Raises a TypeError exception indicating that the datatype is not supported.

Args

`obj`
The object with the unsupported datatype.

`supported_datatype`

The datatype supported by the set.

`def remove(self, obj)`

Removes an element from the set if it is present.

Args

`obj`

The object to remove from the set.

Returns

True if the object was removed successfully, False if the object is not present.

`def size(self)`

Returns the number of elements in the set.

Returns

The number of elements in the set.

Index

Classes

`size`