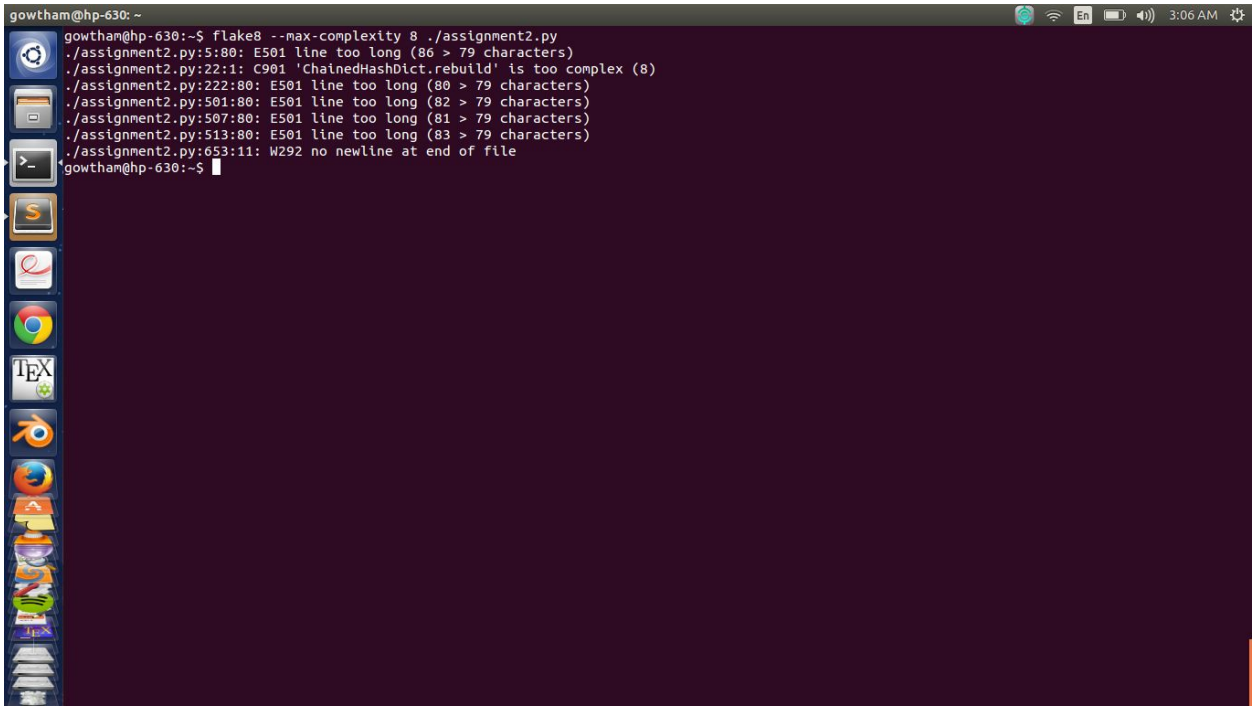
Flake 8 output:



Doc Test Output:

```
Trying:
    sampleSinglyLinkedList = SinglyLinkedList()
Expecting nothing
ok
Trying:
    sampleSinglyLinkedList.prepend(5)
Expecting:
    '5'
ok
Trying:
    sampleSinglyLinkedList.prepend(10)
Expecting:
    '10'
ok
Trying:
    sampleSinglyLinkedList.prepend(13)
Expecting:
    '13'
ok
Trying:
    sampleSinglyLinkedList.remove(5)
Expecting:
    '13'
ok
Trying:
    sampleSinglyLinkedList.remove(10)
Expecting:
    '13'
ok
Trying:
    print sampleSinglyLinkedList
Expecting:
    List:13->10->5
ok
Trying:
    sampleBinarySearchTreeDict = BinarySearchTreeDict()
Expecting nothing
ok
Trying:
```

```
    sampleBinarySearchTreeDict.__setitem__(3,3)
Expecting nothing
ok
Trying:
    sampleBinarySearchTreeDict.__setitem__(2,2)
Expecting nothing
ok
Trying:
    sampleBinarySearchTreeDict.__setitem__(1,1)
Expecting nothing
ok
Trying:
    sampleBinarySearchTreeDict.__setitem__(5,5)
Expecting nothing
ok
Trying:
    sampleBinarySearchTreeDict.__setitem__(4,4)
Expecting nothing
ok
Trying:
    sampleBinarySearchTreeDict.pre_order_keys(sampleBinarySearchTreeDict.root)
Expecting:
    3
    2
    1
    5
    4
ok
Trying:
    sampleBinarySearchTreeDict.in_order_keys(sampleBinarySearchTreeDict.root)
Expecting:
    1
    2
    3
    4
    5
ok
Trying:
    sampleBinarySearchTreeDict.post_order_keys(sampleBinarySearchTreeDict.root)
Expecting:
    1
    2
    4
```

```
    5
    3
ok
Trying:
    sampleOpenAddressing = OpenAddressing()
Expecting nothing
ok
Trying:
    sampleOpenAddressing.__setitem__(1,1)
Expecting nothing
ok
Trying:
    sampleOpenAddressing.__setitem__(2,2)
Expecting nothing
ok
Trying:
    sampleOpenAddressing.__setitem__(3,3)
Expecting nothing
ok
Trying:
    sampleOpenAddressing.__setitem__(4,4)
Expecting nothing
ok
Trying:
    sampleOpenAddressing.__setitem__(5,5)
Expecting nothing
ok
Trying:
    sampleOpenAddressing.__setitem__(6,6)
Expecting nothing
ok
Trying:
    sampleOpenAddressing.__setitem__(7,7)
Expecting nothing
ok
Trying:
    sampleOpenAddressing.__setitem__(8,8)
Expecting:
    ---------- rebuild ----------
ok
Trying:
    sampleOpenAddressing.__setitem__(9,9)
Expecting nothing
```

```
ok
Trying:
    sampleOpenAddressing.__setitem__(10,10)
Expecting nothing
ok
Trying:
    sampleOpenAddressing.__setitem__(11,11)
Expecting nothing
ok
Trying:
    sampleOpenAddressing.__setitem__(12,12)
Expecting nothing
ok
Trying:
    sampleOpenAddressing.__setitem__(13,13)
Expecting nothing
ok
Trying:
    sampleOpenAddressing.__setitem__(14,14)
Expecting nothing
ok
Trying:
    sampleOpenAddressing.__setitem__(15,15)
Expecting:
    ---------- rebuild ----------
ok
Trying:
    sampleOpenAddressing.__setitem__(16,16)
Expecting nothing
ok
Trying:
    sampleOpenAddressing.display()
Expecting:
    Null
    Null
    ('1', '1')
    ('2', '2')
    ('3', '3')
    ('4', '4')
    ('5', '5')
    ('6', '6')
    ('7', '7')
    ('8', '8')
```

```
    ('9', '9')
    ('10', '10')
    ('11', '11')
    ('12', '12')
    ('13', '13')
    ('14', '14')
    ('15', '15')
    ('16', '16')
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
ok
Trying:
    print sampleOpenAddressing.__getitem__(2)
Expecting:
    ('2', '2')
ok
Trying:
    sampleChainedHashDict = ChainedHashDict()
Expecting nothing
ok
Trying:
    sampleChainedHashDict.__setitem__(1,1)
Expecting nothing
```

```
ok
Trying:
    sampleChainedHashDict.__setitem__(2,2)
Expecting nothing
ok
Trying:
    sampleChainedHashDict.__setitem__(3,3)
Expecting nothing
ok
Trying:
    sampleChainedHashDict.__setitem__(4,4)
Expecting nothing
ok
Trying:
    sampleChainedHashDict.__setitem__(5,5)
Expecting nothing
ok
Trying:
    sampleChainedHashDict.__setitem__(6,6)
Expecting nothing
ok
Trying:
    sampleChainedHashDict.__setitem__(7,7)
Expecting nothing
ok
Trying:
    sampleChainedHashDict.__setitem__(8,8)
Expecting nothing
ok
Trying:
    sampleChainedHashDict.__setitem__(9,9)
Expecting:
    ----------calling singly linked list----------
ok
Trying:
    sampleChainedHashDict.__setitem__(10,10)
Expecting nothing
ok
Trying:
    sampleChainedHashDict.__setitem__(11,11)
Expecting nothing
ok
Trying:
```

```
   sampleChainedHashDict.__setitem__(12,12)
Expecting nothing
ok
Trying:
   sampleChainedHashDict.__setitem__(13,13)
Expecting nothing
ok
Trying:
   sampleChainedHashDict.__setitem__(14,14)
Expecting nothing
ok
Trying:
   sampleChainedHashDict.__setitem__(15,15)
Expecting:
   ----------calling singly linked list----------
ok
Trying:
   sampleChainedHashDict.__setitem__(16,16)
Expecting nothing
ok
Trying:
   sampleChainedHashDict.__delitem__(32)
Expecting nothing
ok
Trying:
   sampleChainedHashDict.__delitem__(32)
Expecting nothing
ok
Trying:
   sampleChainedHashDict.display()
Expecting:
   Null
   List:1
   List:2
   List:3
   List:4
   List:5
   List:6
   List:7
   List:8
   List:9
   List:10
   List:11
```

```
    List:12
    List:13
    List:14
    List:15
    List:16
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
ok
Trying:
    print sampleChainedHashDict.__len__()
Expecting:
    16
ok
Trying:
    print sampleChainedHashDict.__contains__(24)
Expecting:
    False
ok
55 items had no tests:
    assignment2
    assignment2.BinarySearchTreeDict
    assignment2.BinarySearchTreeDict.__contains__
```

assignment2.BinarySearchTreeDict.__getitem__
assignment2.BinarySearchTreeDict.__init__
assignment2.BinarySearchTreeDict.__len__
assignment2.BinarySearchTreeDict.__setitem__
assignment2.BinarySearchTreeDict.delitem
assignment2.BinarySearchTreeDict.display
assignment2.BinarySearchTreeDict.height
assignment2.BinarySearchTreeDict.in_order
assignment2.BinarySearchTreeDict.in_order_keys
assignment2.BinarySearchTreeDict.post_order_keys
assignment2.BinarySearchTreeDict.pre_order_keys
assignment2.BinarySearchTreeDict.transplant
assignment2.BinarySearchTreeDict.tree_minimum
assignment2.BinaryTreeNode
assignment2.BinaryTreeNode.__init__
assignment2.ChainedHashDict
assignment2.ChainedHashDict.__contains__
assignment2.ChainedHashDict.__delitem__
assignment2.ChainedHashDict.__getitem__
assignment2.ChainedHashDict.__init__
assignment2.ChainedHashDict.__len__
assignment2.ChainedHashDict.__setitem__
assignment2.ChainedHashDict.bin_count
assignment2.ChainedHashDict.display
assignment2.ChainedHashDict.load_factor
assignment2.ChainedHashDict.rebuild
assignment2.OpenAddressing
assignment2.OpenAddressing.__contains__
assignment2.OpenAddressing.__delitem__
assignment2.OpenAddressing.__getitem__
assignment2.OpenAddressing.__init__
assignment2.OpenAddressing.__len__
assignment2.OpenAddressing.__setitem__
assignment2.OpenAddressing.bin_count
assignment2.OpenAddressing.display
assignment2.OpenAddressing.load_factor
assignment2.OpenAddressing.rebuild
assignment2.SinglyLinkedList
assignment2.SinglyLinkedList.__contains__
assignment2.SinglyLinkedList.__init__
assignment2.SinglyLinkedList.__iter__
assignment2.SinglyLinkedList.__len__
assignment2.SinglyLinkedList.__repr__

assignment2.SinglyLinkedList.prepend
assignment2.SinglyLinkedList.remove
assignment2.SinglyLinkedNode
assignment2.SinglyLinkedNode.__init__
assignment2.SinglyLinkedNode.__repr__
assignment2.SinglyLinkedNode.item
assignment2.SinglyLinkedNode.next
assignment2.chained_hash
assignment2.open_hash
1 items passed all tests:
57 tests in assignment2.test
57 tests in 56 items.
57 passed and 0 failed.
Test passed.

Code:

```
#Class to implement Chanined Hashing
class ChainedHashDict(object):

    def __init__(self, bin_count=10, max_load=0.7, hash_func=hash):
        super(ChainedHashDict, self).__init__()
        self.list_llist = [None, None, None, None, None, None, None, None, None, None]
        self.bin_counter = bin_count
        self.max_load = max_load
        self.current_load = 0
        # TODO initialize
        pass

    @property
    def load_factor(self):
        return self.load_factor
        pass

    @property
    def bin_count(self):
        return self.bin_counter
        pass

    # Call this method to rebuild the hash table when the load factor exceeds
    def rebuild(self, bin_count):
        self.bin_counter = 2 * bin_count
        self.max_load = 2 * self.max_load
        print "-----------------calling singly linked list-----------------"
        new_list_llist = []
        for i in range(self.bin_counter):
            new_list_llist.append(None)

        for llists in self.list_llist:
            if llists is not None:
                for nodes in llists:
                    str_item = str(nodes.item)
                    if nodes is not None:
                        list_index = chained_hash(nodes.item, self.bin_counter)
                        if new_list_llist[list_index] is None:
                            singly_linked_list = SinglyLinkedList()
                            singly_linked_list.prepend(str_item)
                            new_list_llist[list_index] = singly_linked_list
```

```
                    self.current_load = self.current_load + 0.1
                else:
                    singly_linked_list = new_list_llist[list_index]
                    singly_linked_list.prepend(str_item)
                    self.current_load = self.current_load + 0.1
    self.list_llist = new_list_llist
    pass

# Returns a node based on the key
def __getitem__(self, key):
    list_index = chained_hash(key, self.bin_counter)
    linked_list = self.list_llist[list_index]
    str_key = str(key)
    if linked_list is not None:
        for nodes in linked_list:
            if nodes is not None:
                if nodes.item == str_key:
                    return nodes
    return None
    pass

def __setitem__(self, key, value):
    list_index = chained_hash(key, self.bin_counter)
    str_key = str(key)

    if self.current_load > self.max_load:
        self.current_load = 0
        self.rebuild(self.bin_counter)
        self.__setitem__(key, value)
    else:
        if self.list_llist[list_index] is None:
            singly_linked_list = SinglyLinkedList()
            singly_linked_list.prepend(key)
            self.list_llist[list_index] = singly_linked_list
            self.current_load = self.current_load + 0.1
        else:
            singly_linked_list = self.list_llist[list_index]
            singly_linked_list.prepend(str_key)
            self.current_load = self.current_load + 0.1
    pass

# Deletes a node in the table based on key
def __delitem__(self, key):
```

```
        list_index = chained_hash(key, self.bin_counter)
        linked_list = self.list_llist[list_index]

        if linked_list is not None:
            if linked_list.__contains__(key):
                linked_list.remove(key)
                if linked_list.__len__() == 0:
                    self.list_llist[list_index] = None
        pass

    # Checks if the table contains a given key
    def __contains__(self, key):
        list_index = chained_hash(key, self.bin_counter)
        node = self.__getitem__(list_index)
        if node is not None:
            return True
        else:
            return False
        pass

    def __len__(self):
        count = 0
        for llists in self.list_llist:
            if llists is not None:
                count = count + llists.__len__()
        return count
        pass

    def display(self):
        for llists in self.list_llist:
            if llists is not None:
                print llists
            else:
                print "Null"
        pass


# Hash function for chained hashing
def chained_hash(key, bin_size):
    return int(key) % bin_size


# Implementatin of Singly Linked List Node
```

```
class SinglyLinkedNode(object):
    def __init__(self, item=None, next_link=None):
        super(SinglyLinkedNode, self).__init__()
        self._item = item
        self._next = next_link

    @property
    def item(self):
        return self._item

    @item.setter
    def item(self, item):
        self._item = item

    @property
    def next(self):
        return self._next

    @next.setter
    def next(self, next):
        self._next = next

    def __repr__(self):
        return repr(self.item)

# Implementatin of Singly Linked List
class SinglyLinkedList(object):
    def __init__(self):
        super(SinglyLinkedList, self).__init__()
        self.head = None
        # TODO
        pass

    def __len__(self):
        count = 0
        if (self.head is not None):
            for nodes in self:
                count = count + 1
            return count
        else:
            return 0
        # TODO
        pass
```

```
# Iterator method to iterate thorough the Singly Linked List
def __iter__(self):
    current_node = self.head
    if current_node is None:
        raise StopIteration

    while current_node is not None:
        yield current_node
        current_node = current_node.next

    if current_node is None:
        raise StopIteration

def __contains__(self, item):
    str_item = str(item)
    for values in self:
        if values == str_item:
            return True
    pass

# Deletes an item in the linked list
def remove(self, item):
    str_item = str(item)
    current_node = self.head
    previous_node = None
    #if self.__contains__(str_item) is False:
     #   self.head

    if self.head is not None and self.head.item == str_item:
        self.head = self.head.next
        return self.head

    while current_node is not None:
        if current_node.item == str_item:
            previous_node.next = current_node.next
            break
        previous_node = current_node
        current_node = current_node.next
        return self.head

# Add a node at the beginning of the linked list
def prepend(self, item):
```

```
        node = SinglyLinkedNode(str(item), None)
        if self.head is None:
            self.head = node
            return self.head
        else:
            node.next = self.head
            self.head = node
            return self.head
        pass

    def __repr__(self):
        s = "List:" + "->".join([nodes.item for nodes in self])
        return s

# Implementation of Open Addressing Hashing
class OpenAddressing(object):
    def __init__(self, bin_count=10, max_load=0.7, hash_func=hash):
        super(OpenAddressing, self).__init__()
        self.list = [None, None, None, None, None, None, None, None, None, None]
        self.bin_counter = bin_count
        self.max_load = max_load
        self.current_load = 0
        # TODO initialize
        pass

    @property
    def load_factor(self):
        return self.load_factor
        pass

    @property
    def bin_count(self):
        return self.bin_counter
        pass

    # Method to call if max load exceeds the load factor
    def rebuild(self, bin_count):
        self.bin_counter = 2 * bin_count
        self.max_load = 2 * self.max_load
        self.current_load = 0
        print "---------- rebuild ----------"
        new_list = []
        for i in range(self.bin_counter):
```

```
            new_list.append(None)
        pass

        for bins in self.list:
            if bins is not None:
                i = 0
                for slots in new_list:
                    empty_bin = open_hash(bins[0], self.bin_counter, i)
                    if (new_list[empty_bin] is None):
                        new_list[empty_bin] = (bins[0], bins[1])
                        self.current_load = self.current_load + 0.1
                        break
                    i = i + 1
        self.list = new_list

    def __getitem__(self, key):
        i = 0
        for bins in self.list:
            empty_bin = open_hash(key, self.bin_counter, i)
            if (self.list[empty_bin][1] == str(key)):
                return self.list[empty_bin]
                break
            i = i + 1
        return None
        pass

    # Adds a tuple in the Hash Map
    def __setitem__(self, key, value):
        if self.current_load >= self.max_load:
            self.rebuild(self.bin_counter)
            self.__setitem__(key, value)
        else:
            i = 0
            for bins in self.list:
                empty_bin = open_hash(key, self.bin_counter, i)
                if (self.list[empty_bin] is None):
                    self.list[empty_bin] = (str(key), str(value))
                    self.current_load = self.current_load + 0.1
                    break
                i = i + 1
        pass

    # Deletes a tuple in the hash map
```

```
    def __delitem__(self, key):
        list_index = 0
        for tuples in self.list:
            if tuples is not None:
                tuple_key = tuples[0]
                if tuple_key == str(key):
                    self.list[list_index] = None
                    return
            list_index = list_index + 1
        pass

    def __contains__(self, key):
        for tuples in self.list:
            if tuples is not None:
                tuple_key = tuples[0]
                if tuple_key == str(key):
                    return True
        return False
        pass

    def __len__(self):
        count = 0
        for tuples in self.list:
            if tuples is not None:
                count = count + 1
        return count
        pass

    def display(self):
        for tuples in self.list:
            if tuples is not None:
                print tuples
            else:
                print "Null"
        pass

# Hash Funcation for Open Addressing
def open_hash(key, bin_size, i):
    return int(key)+1+i % bin_size


# Implementatin of Binary Tree Node
class BinaryTreeNode(object):
```

```
    def __init__(self, data=None, left=None, right=None, parent=None):
        super(BinaryTreeNode, self).__init__()
        self.data = data
        self.left = left
        self.right = right
        self.parent = parent

# Implementatin of Binary Search Tree
class BinarySearchTreeDict(object):
    def __init__(self):
        super(BinarySearchTreeDict, self).__init__()
        self.root = None
        self.standard_list = []
        # TODO initialize
        pass

    @property
    def height(self, node):
        if node is None:
            return -1

        left = self.height(node.left)
        right = self.height(node.right)

        if left > right:
            return left + 1
        else:
            return right + 1
        pass

    # In order traversal of the tree nodes
    def in_order_keys(self, node):
        if node is not None:
            self.in_order_keys(node.left)
            print node.data
            self.in_order_keys(node.right)
        pass

    # In post order traversal of the tree nodes
    def post_order_keys(self, node):
        if node is not None:
            self.post_order_keys(node.left)
            self.post_order_keys(node.right)
```

```
        print node.data
    pass

# In pre order traversal of the tree nodes
def pre_order_keys(self, node):
    if node is not None:
        print node.data
        self.pre_order_keys(node.left)
        self.pre_order_keys(node.right)
    pass

def in_order(self, node):
    self.standard_list = []
    if node is not None:
        self.in_order(node.left)
        self.standard_list.append(node)
        self.in_order(node.right)

# Returns a node in the tree
def __getitem__(self, node, key):
    if node is not None or node.data == key:
        return node
    elif key < node.data:
        return self.__getitem__(node.left, key)
    else:
        return self.__getitem__(node.right, key)
    pass

# Adds a node in the tree
def __setitem__(self, key, value):
    previous_node = None
    current_node = self.root
    bt_node = BinaryTreeNode()
    bt_node.data = key
    bt_node.value = value
    bt_node.parent = None

    while current_node is not None:
        previous_node = current_node
        if bt_node.data < current_node.data:
            current_node = current_node.left
        else:
            current_node = current_node.right
```

```
    bt_node.parent = previous_node
    if previous_node is None:
        self.root = bt_node
    elif bt_node.data < previous_node.data:
        previous_node.left = bt_node
    else:
        previous_node.right = bt_node
    pass

# Deletes a node in the tree
def delitem(self, key):
    node1 = self.__getitem__(self.root, key)
    if node1.left is None:
        self.transplant(node1, node1.right)
    elif node1.right is None:
        self.transplant(node1, node1.left)
    else:
        node2 = self.tree_minimum(node1.right)
        if node2.parent != node1:
            self.transplant(node2, node2.right)
            node2.right = node1.right
            node2.right.parent = node2
        self.transplant(node1, node2)
        node2.left = node1.left
        node2.left.parent = node2
    pass

def tree_minimum(self, node):
    current = node
    while current.left is not None:
        current = current.left
    return current
    pass

def transplant(self, node1, node2):
    if node1.parent is None:
        self.root = node2
    elif node1 == node1.parent.left:
        node1.parent.left = node2
    else:
        node1.parent.right = node2
    if node2 is not None:
        node2.parent = node1.parent
```

```
            pass

    def __contains__(self, node, key):
        if node is not None or node.data == key:
            return node
        elif key < node.data:
            return self.__getitem__(node.left, key)
        else:
            return self.__getitem__(node.right, key)
        pass

    def __len__(self):
        count = 0
        self.in_order(self.root)
        for i in range(len(self.standard_list)):
            count = count + 1
        return count
        pass

    def display(self):
        self.in_order(self.root, self.root)
        for i in range(len(self.standard_list)):
            node = self.standard_list[i]
            if node is not None:
                print node.data
        pass


def test():
    """
    >>> sampleSinglyLinkedList = SinglyLinkedList()
    >>> sampleSinglyLinkedList.prepend(5)
    '5'
    >>> sampleSinglyLinkedList.prepend(10)
    '10'
    >>> sampleSinglyLinkedList.prepend(13)
    '13'
    >>> sampleSinglyLinkedList.remove(5)
    '13'
    >>> sampleSinglyLinkedList.remove(10)
    '13'
    >>> print sampleSinglyLinkedList
    List:13->10->5
```

```
>>> sampleBinarySearchTreeDict = BinarySearchTreeDict()
>>> sampleBinarySearchTreeDict.__setitem__(3,3)
>>> sampleBinarySearchTreeDict.__setitem__(2,2)
>>> sampleBinarySearchTreeDict.__setitem__(1,1)
>>> sampleBinarySearchTreeDict.__setitem__(5,5)
>>> sampleBinarySearchTreeDict.__setitem__(4,4)
>>> sampleBinarySearchTreeDict.pre_order_keys(sampleBinarySearchTreeDict.root)
3
2
1
5
4
>>> sampleBinarySearchTreeDict.in_order_keys(sampleBinarySearchTreeDict.root)
1
2
3
4
5
>>> sampleBinarySearchTreeDict.post_order_keys(sampleBinarySearchTreeDict.root)
1
2
4
5
3
>>> sampleOpenAddressing = OpenAddressing()
>>> sampleOpenAddressing.__setitem__(1,1)
>>> sampleOpenAddressing.__setitem__(2,2)
>>> sampleOpenAddressing.__setitem__(3,3)
>>> sampleOpenAddressing.__setitem__(4,4)
>>> sampleOpenAddressing.__setitem__(5,5)
>>> sampleOpenAddressing.__setitem__(6,6)
>>> sampleOpenAddressing.__setitem__(7,7)
>>> sampleOpenAddressing.__setitem__(8,8)
---------- rebuild ----------
>>> sampleOpenAddressing.__setitem__(9,9)
>>> sampleOpenAddressing.__setitem__(10,10)
>>> sampleOpenAddressing.__setitem__(11,11)
>>> sampleOpenAddressing.__setitem__(12,12)
>>> sampleOpenAddressing.__setitem__(13,13)
>>> sampleOpenAddressing.__setitem__(14,14)
>>> sampleOpenAddressing.__setitem__(15,15)
---------- rebuild ----------
>>> sampleOpenAddressing.__setitem__(16,16)
```

```
>>> sampleOpenAddressing.display()
Null
Null
('1', '1')
('2', '2')
('3', '3')
('4', '4')
('5', '5')
('6', '6')
('7', '7')
('8', '8')
('9', '9')
('10', '10')
('11', '11')
('12', '12')
('13', '13')
('14', '14')
('15', '15')
('16', '16')
Null
Null
Null
Null
Null
Null
Null
Null
Null
Null
Null
Null
Null
Null
Null
Null
Null
Null
Null
Null
Null
Null
Null
Null
Null
>>> print sampleOpenAddressing.__getitem__(2)
('2', '2')
```

```
>>> sampleChainedHashDict = ChainedHashDict()
>>> sampleChainedHashDict.__setitem__(1,1)
>>> sampleChainedHashDict.__setitem__(2,2)
>>> sampleChainedHashDict.__setitem__(3,3)
>>> sampleChainedHashDict.__setitem__(4,4)
>>> sampleChainedHashDict.__setitem__(5,5)
>>> sampleChainedHashDict.__setitem__(6,6)
>>> sampleChainedHashDict.__setitem__(7,7)
>>> sampleChainedHashDict.__setitem__(8,8)
>>> sampleChainedHashDict.__setitem__(9,9)
----------calling singly linked list----------
>>> sampleChainedHashDict.__setitem__(10,10)
>>> sampleChainedHashDict.__setitem__(11,11)
>>> sampleChainedHashDict.__setitem__(12,12)
>>> sampleChainedHashDict.__setitem__(13,13)
>>> sampleChainedHashDict.__setitem__(14,14)
>>> sampleChainedHashDict.__setitem__(15,15)
----------calling singly linked list----------
>>> sampleChainedHashDict.__setitem__(16,16)
>>> sampleChainedHashDict.__delitem__(32)
>>> sampleChainedHashDict.__delitem__(32)
>>> sampleChainedHashDict.display()
Null
List:1
List:2
List:3
List:4
List:5
List:6
List:7
List:8
List:9
List:10
List:11
List:12
List:13
List:14
List:15
List:16
Null
Null
Null
Null
```

```
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    Null
    >>> print sampleChainedHashDict.__len__()
    16
    >>> print sampleChainedHashDict.__contains__(24)
    False
    """
    pass

if __name__ == '__main__':
    import doctest
    doctest.testmod()
    test()
```