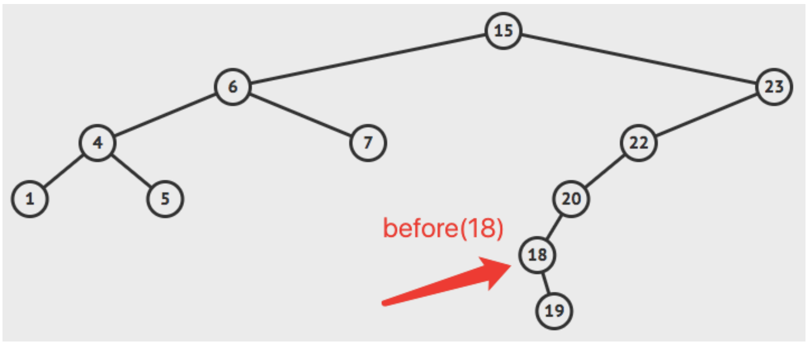
CSCI-SHU 210 Data Structures

Recitation 9 Binary Search Trees

Today's recitation, we will first practice Binary Search Tree operations by hand, then we will start coding.

1. Practice the following Binary Search Tree operations:

- **Before(node)**
  - **Case 1**



a). What is before(18) for the BST above?

____

b). There are two cases in **`before(self, node)`** function on the next page. Which case is executed for before(18)? Highlight case 1 in the code.

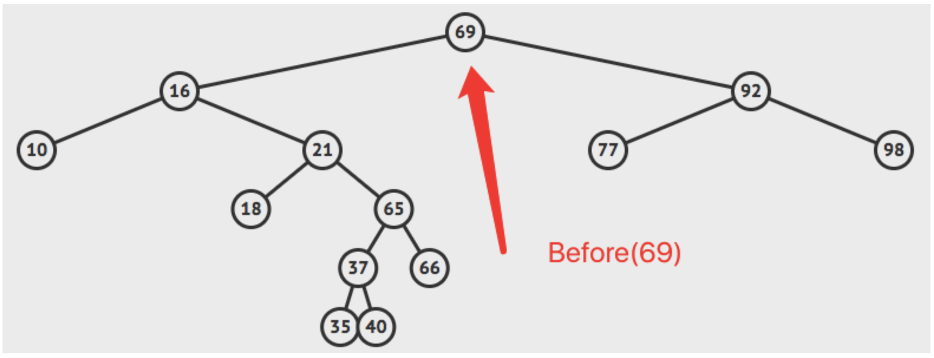
____

c). Briefly describe what to do, in order to find before(18) for the BST above.

____

____

  - **Case 2**



a). What is before(69) for the BST above?

____

b). There are two cases in **`before(self, node)`** function on the next page. Which case is executed for before(69)? Highlight case 2 in the code.

____

c). Briefly describe what to do, in order to find before(69) for the BST above.

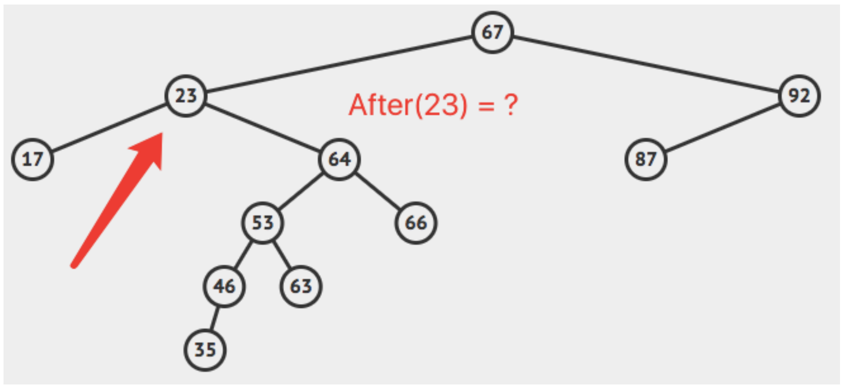
____

____

- **Code snippet for `before(self, node)`**

```python
def _subtree_last_position(self, node):
    """Return the node that contains the last item in subtree rooted at given node.
"""
    walk = node
    while walk._right is not None:                # keep walking right
        walk = walk._right
    return walk

def before(self, node):
    """Return the node that is just before the given node in the natural order.

    Return None if the given node is the first position.
    """
    if node._left is not None:
        return self._subtree_last_position(node._left)
    else:
        # walk upward
        walk = node
        above = walk._parent
        while above is not None and walk == above._left:
            walk = above
            above = walk._parent
        return above
```

- **After(node)**
    - **Case 1**



a). What is after(23) for the BST above?

____

b). There are two cases in **`after(self, node)`** function on the next page. Which case is executed for after(23)? Highlight case 1 in the code.

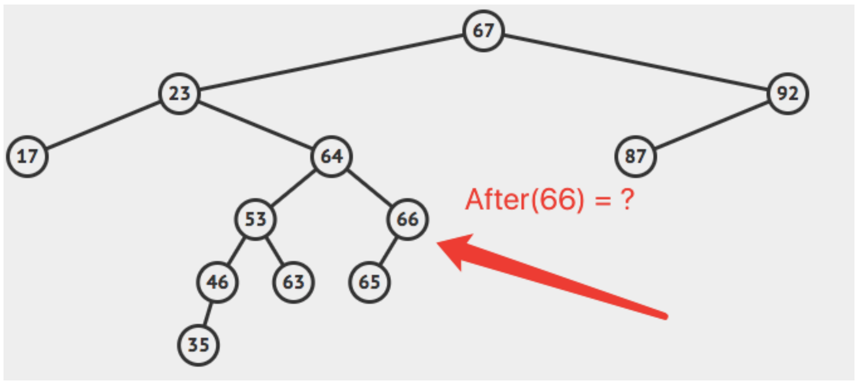
____

c). Briefly describe what to do, in order to find after(23) for the BST above.

____

____

- **Case 2**



a). What is after(66) for the BST above?

____

b). There are two cases in **`after(self, node)`** function on the next page. Which case is executed for after(66)? Highlight case 2 in the code.

____

c). Briefly describe what to do, in order to find after(66) for the BST above.

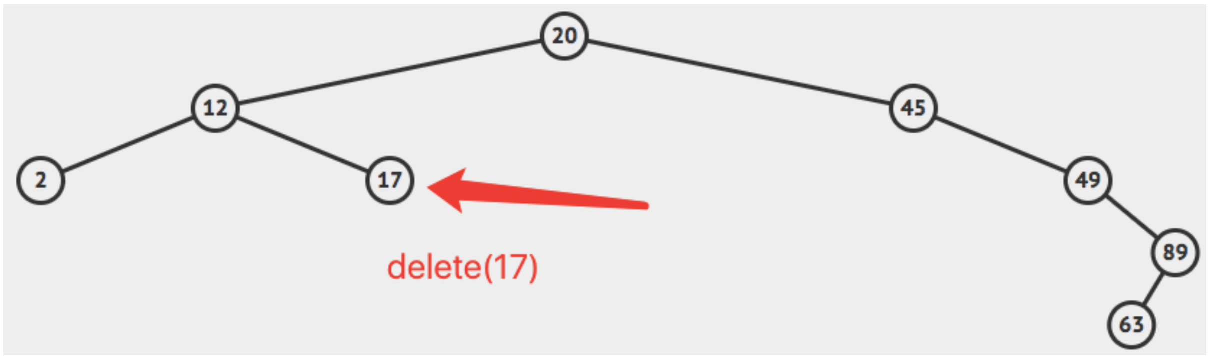
____

____

- Code snippet for `after(self, node)`

```python
def _subtree_first_position(self, node):
    """Return the node that contains the first item in subtree rooted at given node
."""
    walk = node
    while walk._left is not None:                 # keep walking left
        walk = walk._left
    return walk

def after(self, node):
    """Return the node that is just after the given node in the natural order.

    Return None if the given node is the last position.
    """
    if node._right is not None:
        return self._subtree_first_position(node._right)
    else:
        walk = node
        above = walk._parent
        while above is not None and walk == above._right:
            walk = above
            above = walk._parent
        return above
```

- **delete(node)**
    - **Case 1**



a). Suppose we have the BST above. Draw the BST after delete(17).

b). There are two cases in **`delete(self, node)`** function on the next page. Case 1 and case 2 are combined in the same function. Can you locate this function?

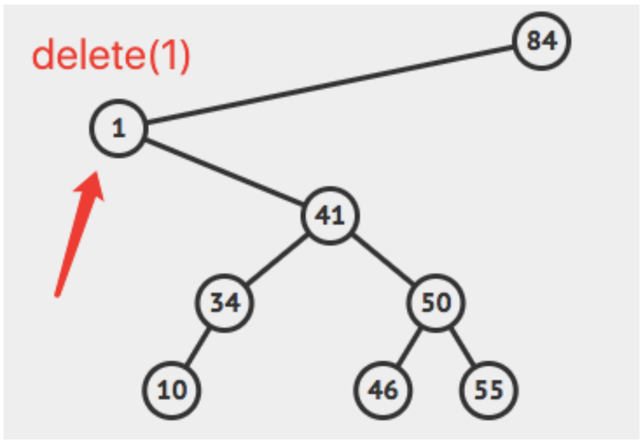
____

c). Briefly describe what to do, in order to delete(17) from the BST above.

____

____

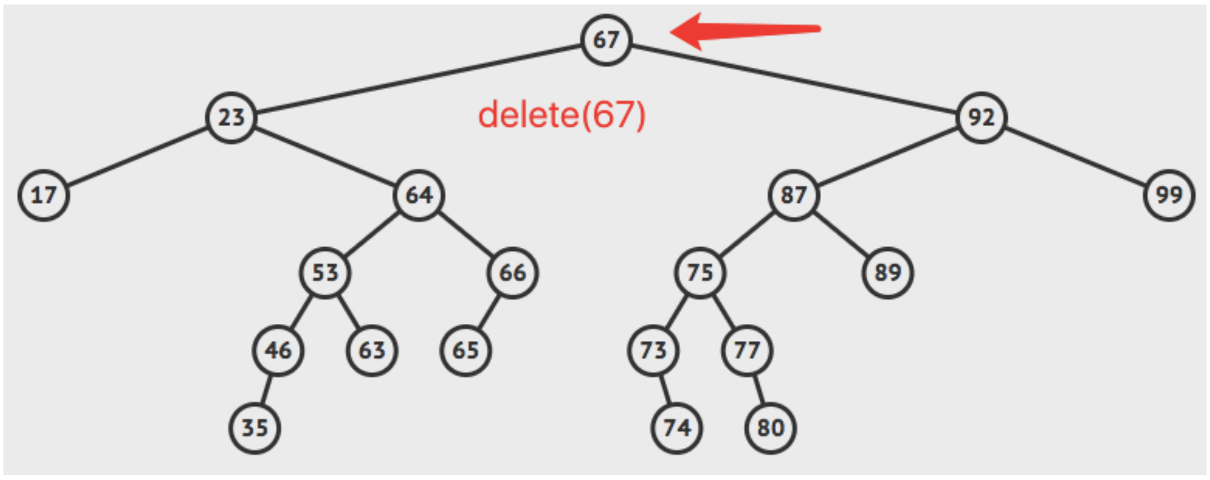
- **Case 2**



a). Suppose we have the BST above. Draw the BST after delete(1).

b). Briefly describe what to do, in order to delete(1) from the BST above.

- **Case 3**



a). Suppose we have the BST above. Draw the BST after delete(67).

b). There are two cases in **`delete(self, node)`** function on the next page. Can you locate the code for case 3?

c). Briefly describe what to do, in order to delete(67) from the BST above.

- **Code snippet for `delete(self, node)`**

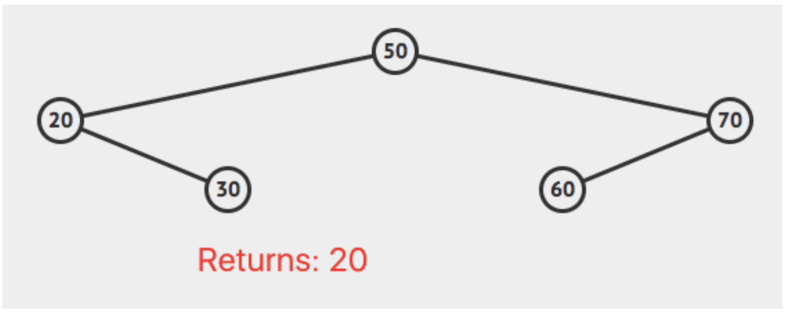
```python
def _delete(self, node):
    """Delete the given node, and replace it with its child, if any.

    Return the element that had been stored at the given node.
    Raise ValueError if node has two children.
    """
    if self.num_children(node) == 2:
        raise ValueError('Position has two children')
    child = node._left if node._left else node._right  # might be None
    if child is not None:
        child._parent = node._parent     # child's grandparent becomes parent
    if node is self._root:
        self._root = child             # child becomes root
    else:
        parent = node._parent
        if node is parent._left:
            parent._left = child
        else:
            parent._right = child
    self._size -= 1
    return node._element

def delete(self, node):
    """Remove the given node."""
    if node._left and node._right:            # node has two children
        replacement = self._subtree_last_position(node._left)
        self._replace(node, replacement._element)    # from BinaryTree(class Tree)
        node = replacement
    # now node has at most one child
    parent = node._parent
    self._delete(node)                        # inherited from BinaryTree(class Tree)
    self._rebalance_delete(parent)            # (This line only works in AVL Tree)

```

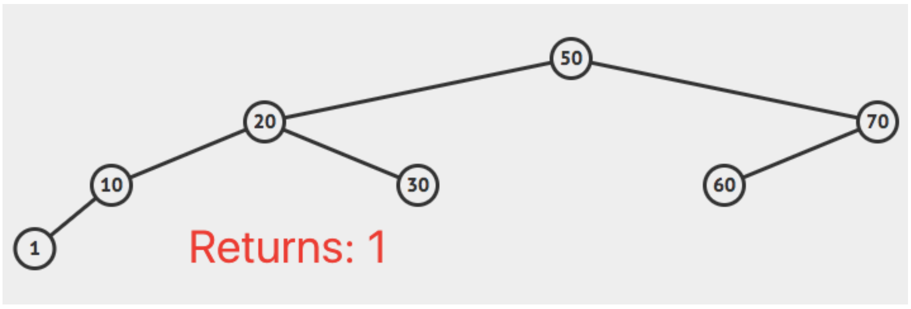
2. Coding exercises.

```
Your task 1: Try to perform some operations on our BinarySearchTree class.
1. Insert 0, 1, 2, 3, 4 into the tree.
2. Get the Node of 2 by calling get_node(self, v) function.
3. Use before(self, node) function to get node of 1.
4. Use after(self, node) function to get node of 3.
5. Delete 0, 1, 2, 3, 4 from the tree.
```

Your task 2: Implement function `minimum(self)` at line 405. When called, the minimum element within the tree is returned.

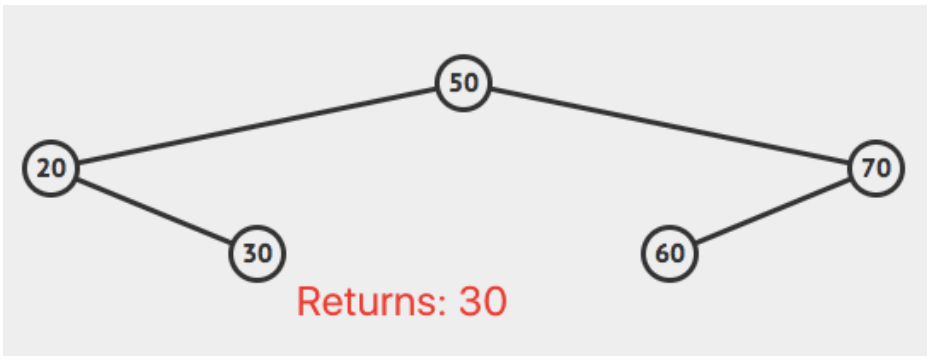
For example:

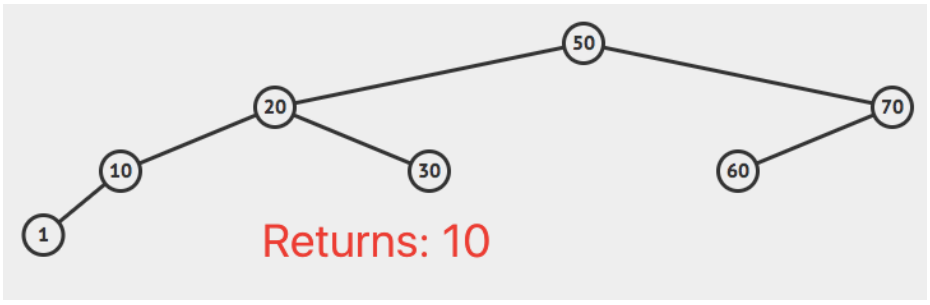




Your task 3: Implement function `second_minimum(self)` at line 412. When called, the second smallest element within the tree is returned.

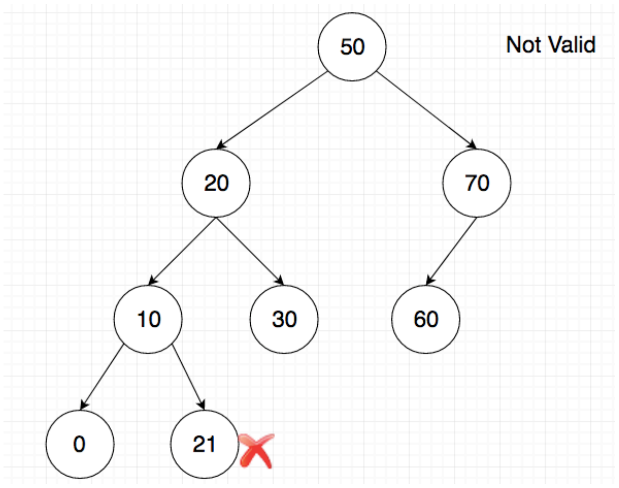
For example:





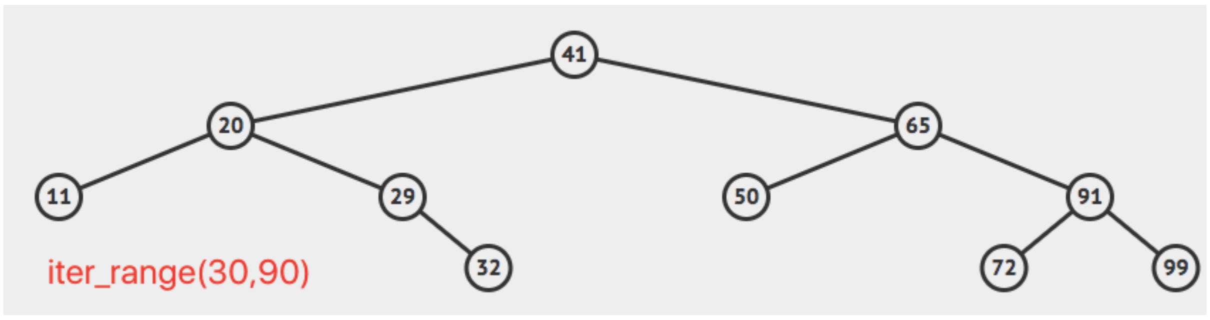
Your task 4: Implement function `is_valid(self)` at line 419. When called, returns True if the self tree is a valid binary search tree. Returns false otherwise.

For example:



Your task 5: Implement function `iter_range(self, start, stop)` at line 426. When called, Yield a generator that contains elements in order, such that start <= elements <= stop.

For example:



iter_range(30, 90) on the above BST should create an generator.

This generator object should contain:

32, 41, 50, 65, 72