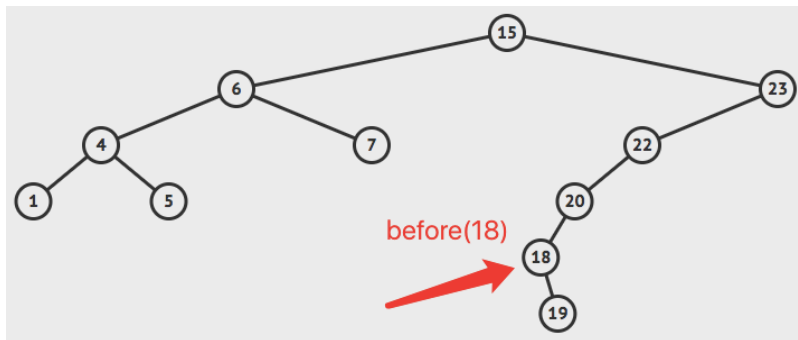# CSCI-SHU 210 Data Structures

Recitation10 Worksheet 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:
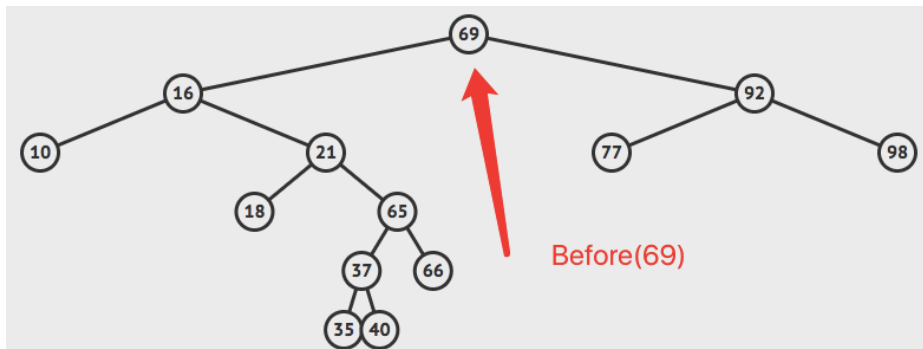
> **Before(node)**
>> o **Case 1**

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.
_____
_____

>> o **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.
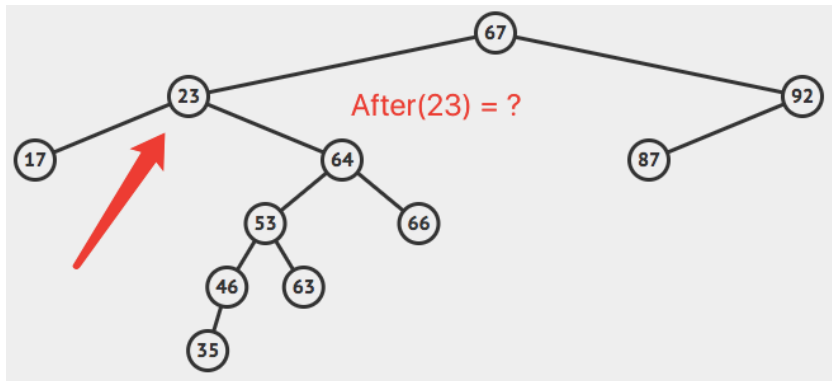_____

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

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

```python
1.  def _subtree_last_node(self, node):
2.  """Return the node that contains the last item in subtree rooted at given node."""
3.      walk = node

4.      while walk._right is not None:           # keep walking right
5.          walk = walk._right
6.      return walk
7.
8.  def before(self, node):
9.      """Return the node that is just before the given node in the natural order.
10.
11.     Return None if the given node is the first position.
12.     """
13.     if node._left is not None:
14.         return self._subtree_last_node(node._left)
15.     else:
16.     # walk upward
17.         walk = node
18.         above = walk._parent
19.         while above is not None and walk == above._left:
20.             walk = above
21.             above = walk._parent
22.         return above
```
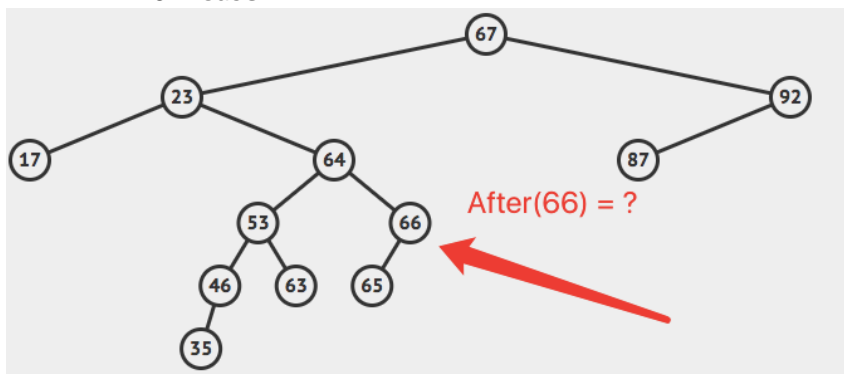
> **After(node)**
>   o **Case 1**

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.

_____

_____

  o **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
1. def _subtree_first_node(self, node):
2. """Return the node that contains the first item in subtree rooted at given node ."""
3.     walk = node

4.     while walk._left is not None:              # keep walking left
5.         walk = walk._left
6.     return walk
7.
8. def after(self, node):
9.     """Return the node that is just after the given node in the natural order.
10.
11.     Return None if the given node is the last position.
12.     """
13.     if node._right is not None:
14.         return self._subtree_first_node(node._right)
15.     else:
16.         walk = node
17.         above = walk._parent
18.         while above is not None and walk == above._right:
19.             walk = above
20.             above = walk._parent
21.         return above
```
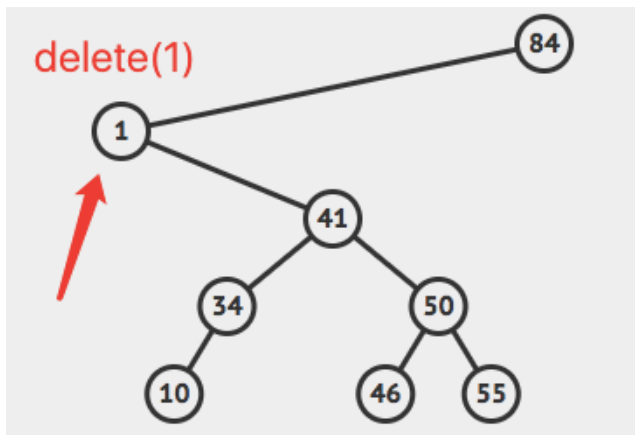
➢ **delete(node)**
  o **Case 1**
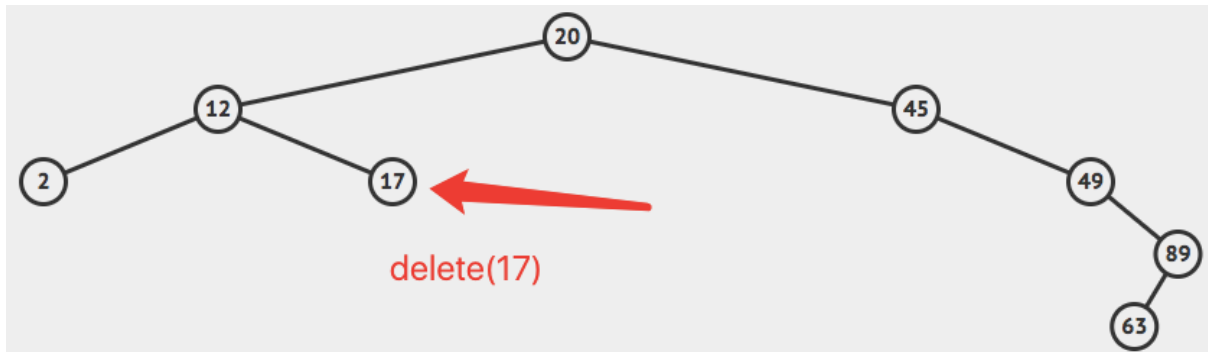
delete(17)

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.
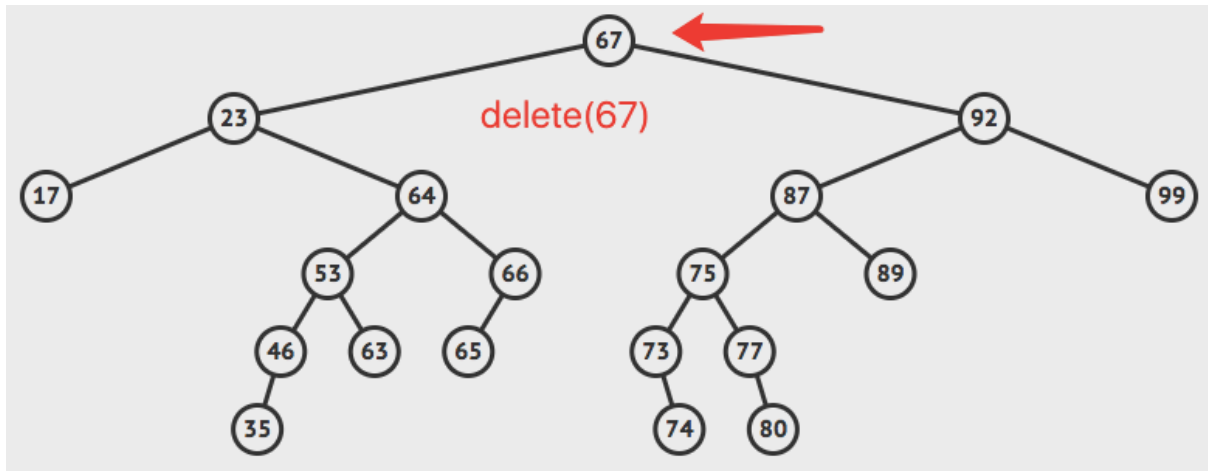
_____
_____

  o **Case 2**

delete(1)

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
1.  def _delete(self, node):
2.      """Delete the given node, and replace it with its child, if any.
3.
4.      Return the element that had been stored at the given node.
5.      Raise ValueError if node has two children.
6.      """
7.      if self.num_children(node) == 2:
8.          raise ValueError('Node has two children')
9.      child = node._left if node._left else node._right   # might be None
10.     if child is not None:
11.         child._parent = node._parent     # child's grandparent becomes parent
12.     if node is self._root:
13.         self._root = child               # child becomes root
14.     else:
15.         parent = node._parent
16.         if node is parent._left:
17.             parent._left = child
18.         else:
19.             parent._right = child
20.     self._size -= 1
21.     return node._element
22.
23. def delete(self, node):
24.     """Remove the given node."""
25.     if node._left and node._right:            # node has two children
26.         replacement = self._subtree_last_node(node._left)
27.         self._replace(node, replacement._element)    # from BinaryTree(class Tree)
28.         node = replacement
29.     # now node has at most one child
30.     parent = node._parent
31.     self._delete(node)                        # inherited from BinaryTree(class Tree)
32.     self._rebalance_delete(parent)            # (This line only works in AVL Tree)
33.
```

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.
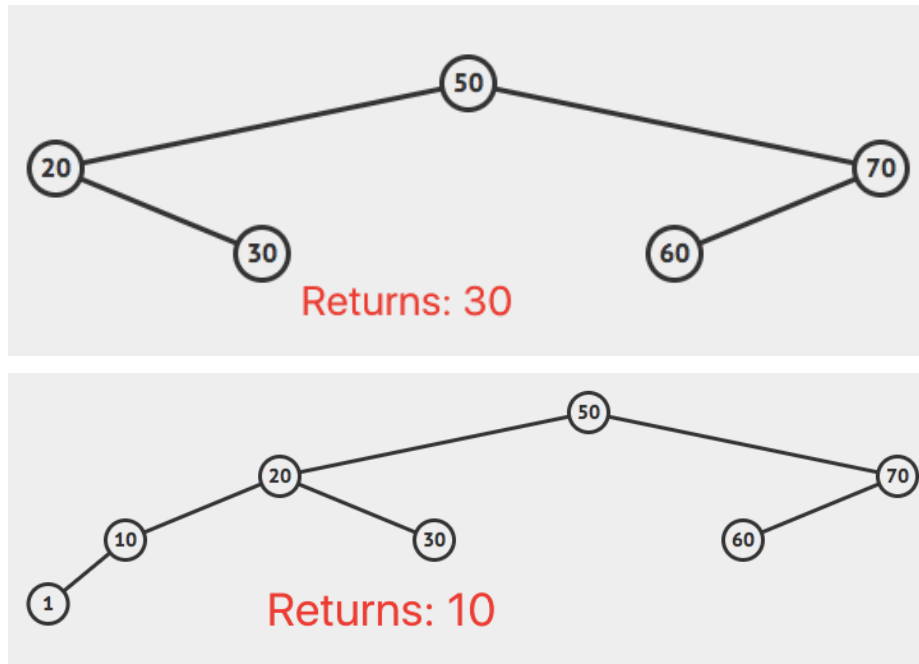
5. Delete 0, 1, 2, 3, 4 from the tree.

Your task 2: Implement function minimum(self). When called, the minimum element within the tree is returned.
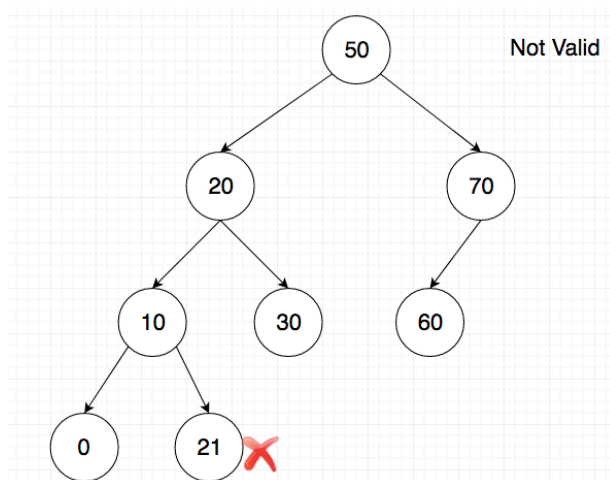
For example:

Returns: 20
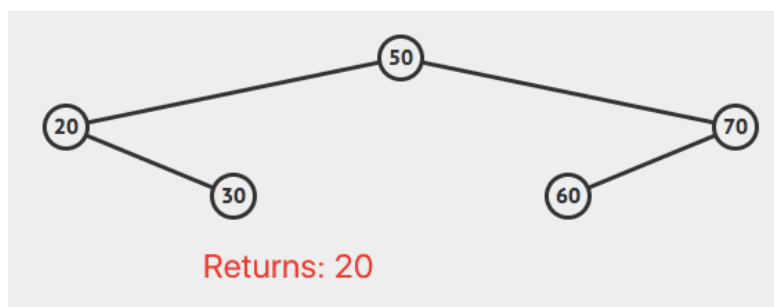
Returns: 1

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

For example:

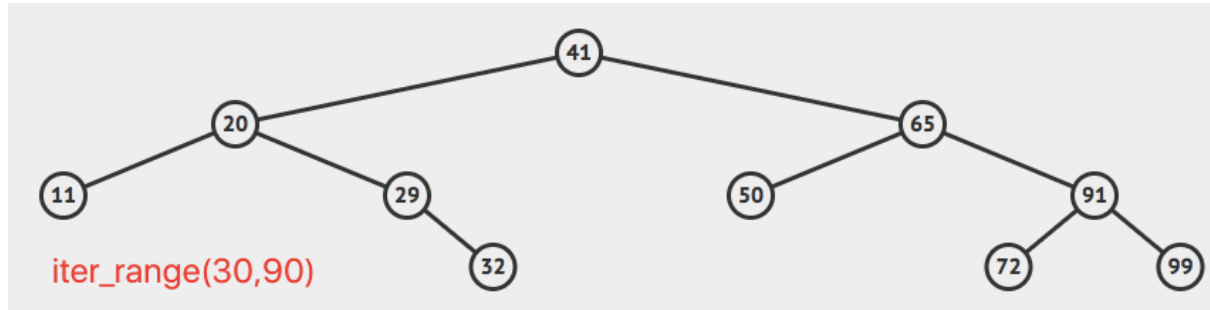

Returns: 30

Returns: 10

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

Not Valid

Your task 5: Implement function `iter_range(self, start, stop)`. 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