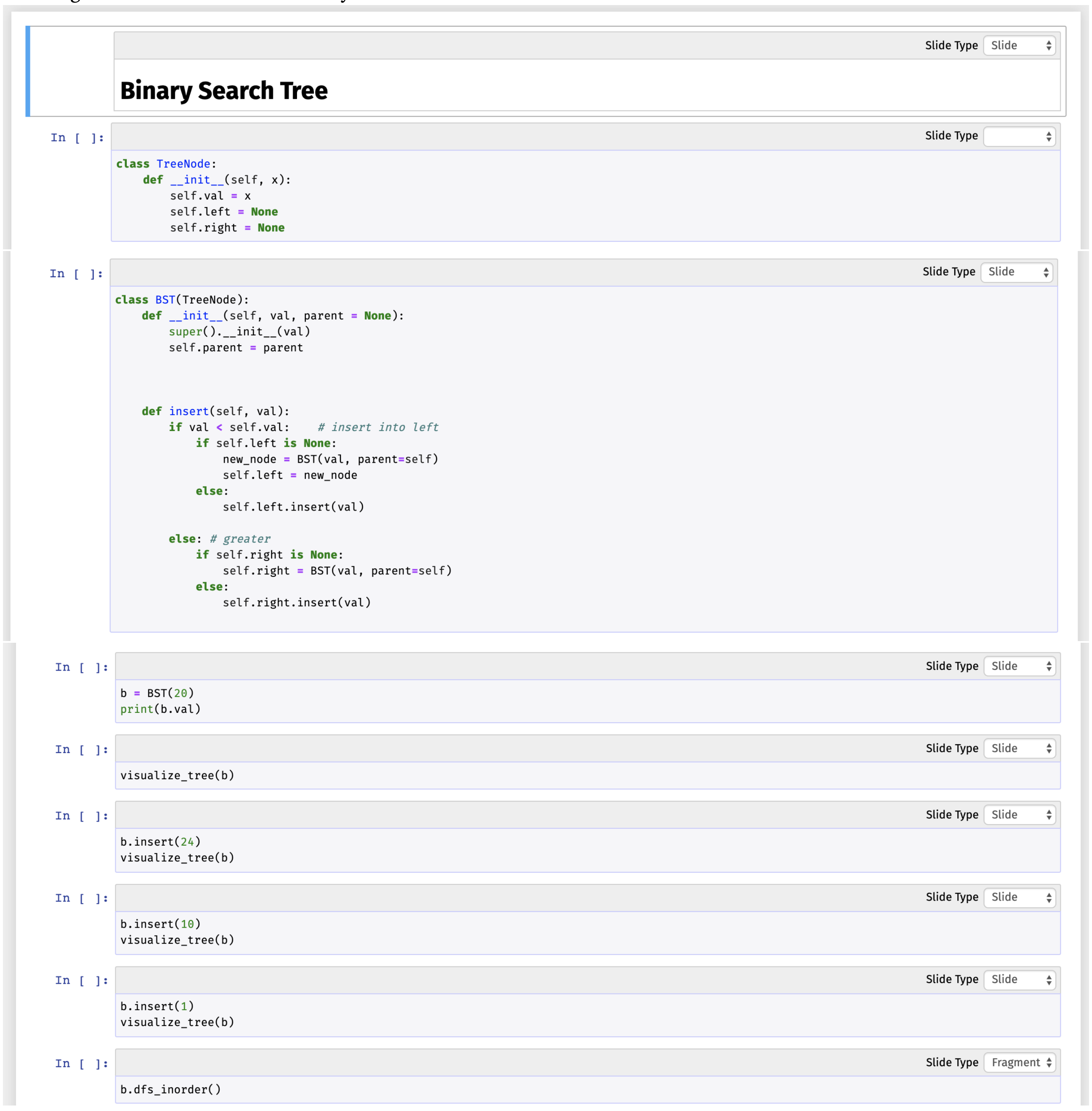
CS218 - Data Structures
FAST NUCES Peshawar Campus
Dr. Nauman (recluze.net)

October 7, 2019

1 Trees

Raster images of the notebook 13-binary-search-tree.



Slide Type Slide

Binary Search Tree

In [ ]: Slide Type

```
class TreeNode:
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None
```

In [ ]: Slide Type Slide

```
class BST(TreeNode):
    def __init__(self, val, parent = None):
        super().__init__(val)
        self.parent = parent



    def insert(self, val):
        if val < self.val:    # insert into left
            if self.left is None:
                new_node = BST(val, parent=self)
                self.left = new_node
            else:
                self.left.insert(val)

        else: # greater
            if self.right is None:
                self.right = BST(val, parent=self)
            else:
                self.right.insert(val)
```

In [ ]: Slide Type Slide

```
b = BST(20)
print(b.val)
```

In [ ]: Slide Type Slide

```
visualize_tree(b)
```

In [ ]: Slide Type Slide

```
b.insert(24)
visualize_tree(b)
```

In [ ]: Slide Type Slide

```
b.insert(10)
visualize_tree(b)
```

In [ ]: Slide Type Slide

```
b.insert(1)
visualize_tree(b)
```

In [ ]: Slide Type Fragment

```
b.dfs_inorder()
```

In [ ]: Slide Type Slide

```python
b.insert(21)
b.insert(26)
b.insert(25)
```

In [ ]: Slide Type

```python
visualize_tree(b)
```

Slide Type Slide

Deletion from the BST

Slide Type

First, we'll need some helper functions. Let's discuss those first.

In [ ]: Slide Type Slide

```python
def find_root(self):
    """Find the absolute root of the BST to which self belongs. Keep going up until you reach no-parent node."""

    temp = self
    while temp.parent is not None:
        temp = temp.parent

    return temp
    # keep going up until no parent .... return that.

BST.find_root = find_root
```

In [ ]: Slide Type Slide

```python
def find_min(self):
    """Find the minimum value starting from self.
       In BST, this is simple, keep going left until no more left is left!"""

    min_node = self

    if self.left is not None:
        min_node = find_min(self.left)

    return min_node


BST.find_min = find_min
```

In [ ]: Slide Type

```python
visualize_tree(b)
print("Min is: ", b.find_min().val)
```

In [ ]: Slide Type Slide

```python
def set_for_parent(self, new_ref):
    """Disconnect self from parent and attach new_ref to parent in self's place."""

    if self.parent is None: return

    if self.parent.right == self:
        self.parent.right = new_ref

    if self.parent.left == self:
        self.parent.left = new_ref

BST.set_for_parent = set_for_parent
```

In [ ]: Slide Type Slide

```python
def replace_with_node(self, node):
    """Replace self with node (which is a child). Make sure to fix the parent of the node and parent' pointing to node.
       Assume we have no children other than node."""

    self.set_for_parent(node)    # connect new node to parent on poper location
    node.parent = self.parent    # set node's parent correctly
    self.parent = None           # disconnect self from the parent
    return node.find_root()      # find root again

BST.replace_with_node = replace_with_node
```

In [ ]:

Slide Type Slide

```python
def delete(self, val):
    # first ... if we are alone, on the root and no children plus the value matches just return None
    if self.parent is None and self.right is None and self.left is None and self.val == val:
        return None

    # we are the node to be deleted
    if self.val == val:
        # check if we are leaf
        if self.right is None and self.left is None:
            self.set_for_parent(None)   # set in place of self a None
            return self.find_root()


        # check if we have just a left node
        if self.right is None:
            return self.replace_with_node(self.left)

        # check if we have just a right node
        if self.left is None:
            return self.replace_with_node(self.right)


        # now we have both children. Find the successor and replace "self" with it.
        # (Our succ is definitely in our right child and it can't have two children because left child will always be smaller.)
        successor = self.right.find_min()

        # copy successor's val here
        self.val = successor.val


        return self.right.delete(successor.val)
        # ^ delete the successor node, which is in our right child BST.
        # ^ It's guaranteed that it's the simpler case since successor CANNOT have a left child.


    # we were not the node to be deleted, go to children
    if val < self.val :
        if self.left is not None:
            return self.left.delete(val)
        else:
            return self.find_root() # nothing to delete
    else:
        if self.right is not None:
            return self.right.delete(val)
        else: return self.find_root()


BST.delete = delete
```

In [ ]:

Slide Type Slide

```python
b = BST(20)
b.insert(24)
b.insert(21)
b.insert(10)
b.insert(25)
b.insert(26)
visualize_tree(b)
```

In [ ]:

Slide Type Slide

```python
b = b.delete(20)
visualize_tree(b)
```

In [ ]:

Slide Type Slide

```python
b = b.delete(21)
visualize_tree(b)
```

In [ ]:

Slide Type Slide

```python
b = b.delete(25)
visualize_tree(b)
```

In [ ]:

Slide Type Slide

```python
b = b.delete(24)
visualize_tree(b)
```

In [ ]: Slide Type Slide

```
b = b.delete(25)
visualize_tree(b)
```

In [ ]: Slide Type Slide

```
b = b.delete(26)
visualize_tree(b)
```

In [ ]: Slide Type Slide

```
b = BST(5)
b.insert(1)
b.insert(20)
b.insert(10)
b.insert(50)
b.insert(30)
b.insert(40)
b.insert(60)
visualize_tree(b)
```

In [ ]: Slide Type Slide

```
b = b.delete(20)
visualize_tree(b)
```

Slide Type Slide

Getting Sorted Values Back

In [ ]: Slide Type Slide

```
l = [1, 2, 17, 9, 13, 21, 5, 71, 6, 8]
```

In [ ]: Slide Type Fragment

```
b = BST(l[0])
for i in l[1:]:
    b.insert(i)
```

In [ ]: Slide Type Slide

```
visualize_tree(b)
```

In [ ]: Slide Type Slide

```
b.dfs_inorder()
```

Issue with the BST - Balance

In [ ]:

```
l = [1, 2, 4, 9, 13, 21, 51, 71, 82]     # , 6]
```

In [ ]:

```
b = BST(l[0])
for i in l[1:]:
    b.insert(i)
```

In [ ]:

```
visualize_tree(b)
```

In [ ]:

```
b.dfs_inorder()
```

This issue is resolved using advanced BSTs such as RB trees and AVL trees. Balance is achieved using the concept of rotation based on different rules.