Binary Search Tree Exercises

Download Starter Code <../dsa-bsts.zip>

We've supplied you with a ***Node*** class and a constructor for the ***BinarySearchTree*** class. Here are descriptions of the methods you should write for instances of ***BinarySearchTree***:

insert

This function should insert a node in a binary tree. It should return the BinarySearchTree and should be solved using iteration.

```
var binarySearchTree = new BinarySearchTree();
binarySearchTree.insert(15);
binarySearchTree.insert(20);
binarySearchTree.insert(10);
binarySearchTree.insert(12);
binarySearchTree.root.value // 15
binarySearchTree.root.right.value // 20
binarySearchTree.root.left.right.value // 12

var binarySearchTree = new BinarySearchTree();
binarySearchTree.insert(15)
binarySearchTree.insert(20)
binarySearchTree.insert(10)
binarySearchTree.insert(12);
binarySearchTree.root.value // 15
binarySearchTree.root.right.value // 20
binarySearchTree.root.left.right.value // 12
```

insertRecursively

This function should insert a node in a binary tree. It should return the BinarySearchTree and should be solved using recursion.

```
var binarySearchTree = new BinarySearchTree();
binarySearchTree.insertRecursively(15);
binarySearchTree.root.value // 15
binarySearchTree.root.left // null
binarySearchTree.root.right // null

var binarySearchTree = new BinarySearchTree();

binarySearchTree.insertRecursively(15);
binarySearchTree.insertRecursively(20);
binarySearchTree.insertRecursively(10);
binarySearchTree.insertRecursively(12);
binarySearchTree.root.value // 15
binarySearchTree.root.right.value // 20
binarySearchTree.root.left.right.value // 12

var binarySearchTree = new BinarySearchTree();
```

```
binarySearchTree.insertRecursively(15)
binarySearchTree.insertRecursively(20)
binarySearchTree.insertRecursively(10)
binarySearchTree.insertRecursively(12);
binarySearchTree.root.value // 15
binarySearchTree.root.right.value // 20
binarySearchTree.root.left.right.value // 12
```

find

This function should find a node in a binary tree. It should return the node if found, otherwise return ***undefined***. This should be solved using iteration.

```
var binarySearchTree = new BinarySearchTree();
binarySearchTree.insert(15)
binarySearchTree.insert(20)
binarySearchTree.insert(10)
binarySearchTree.insert(12);
var foundNode = binarySearchTree.findIteratively(20);
foundNode.value // 20
foundNode.left // null
foundNode.right // null

var binarySearchTree = new BinarySearchTree();

binarySearchTree.insert(15)
binarySearchTree.insert(20)
binarySearchTree.insert(10)
binarySearchTree.insert(12);
var foundNode = binarySearchTree.findIteratively(120);
foundNode // undefined
```

findRecursively

This function should find a node in a binary tree. It should return the node if found, otherwise return ***undefined***. This should be solved using recursion.

```
var binarySearchTree = new BinarySearchTree();
binarySearchTree.insert(15)
binarySearchTree.insert(20)
binarySearchTree.insert(10)
binarySearchTree.insert(12);
var foundNode = binarySearchTree.findRecursively(20);
foundNode.value // 20
foundNode.left // null
foundNode.right // null

var binarySearchTree = new BinarySearchTree();
binarySearchTree.insert(15)
binarySearchTree.insert(20)
binarySearchTree.insert(10)
binarySearchTree.insert(12);
var foundNode = binarySearchTree.findRecursively(120);
foundNode // undefined
```

dfsPreOrder

This function should search through each node in the binary search tree using pre-order depth first search and return an array containing each node’s value.

```
var binarySearchTree = new BinarySearchTree();
binarySearchTree.insert(15)
binarySearchTree.insert(20)
binarySearchTree.insert(10)
binarySearchTree.insert(12)
binarySearchTree.insert(1)
binarySearchTree.insert(5)
binarySearchTree.insert(50);
binarySearchTree.dfsPreOrder() // [15, 10, 1, 5, 12, 20, 50]
```

dfsInOrder

This function should search through each node in the binary search tree using in-order depth first search and return an array containing each node’s value.

```
var binarySearchTree = new BinarySearchTree();
binarySearchTree.insert(15)
binarySearchTree.insert(20)
binarySearchTree.insert(10)
binarySearchTree.insert(12)
binarySearchTree.insert(1)
binarySearchTree.insert(5)
binarySearchTree.insert(50);
binarySearchTree.dfsInOrder() // [1, 5, 10, 12, 15, 20, 50]
```

dfsPostOrder

This function should search through each node in the binary search tree using post-order depth first search and return an array containing each node’s value.

```
var binarySearchTree = new BinarySearchTree();
binarySearchTree.insert(15)
binarySearchTree.insert(20)
binarySearchTree.insert(10)
binarySearchTree.insert(12)
binarySearchTree.insert(1)
binarySearchTree.insert(5)
binarySearchTree.insert(50);
binarySearchTree.dfsPostOrder() // [5, 1, 12, 10, 50, 20, 15]
```

bfs

This function should search through each node in the binary search tree using breadth first search and return an array containing each node’s value.

```
var binarySearchTree = new BinarySearchTree();
binarySearchTree.insert(15)
binarySearchTree.insert(20)
binarySearchTree.insert(10)
binarySearchTree.insert(12)
binarySearchTree.insert(1)
binarySearchTree.insert(5)
binarySearchTree.insert(50);
binarySearchTree.bfs() // [15, 10, 20, 1, 12, 50, 5]
```

Further Study

remove

This function should remove a node from a binary search tree. Your remove function should be able to handle removal of the root node, removal of a node with one child and removal of a node with two children. The function should return the node removed.

```
var binarySearchTree = new BinarySearchTree();
binarySearchTree.insert(15)
binarySearchTree.insert(20)
binarySearchTree.insert(10)
binarySearchTree.insert(12)
binarySearchTree.insert(1)
binarySearchTree.insert(5)
binarySearchTree.insert(50);
binarySearchTree.remove(50);
binarySearchTree.root.right.value // 20
binarySearchTree.root.right.right // null

binarySearchTree.remove(5);
binarySearchTree.root.left.left.value // 1
binarySearchTree.root.left.left.right // null

var binarySearchTree = new BinarySearchTree();
binarySearchTree.insert(15)
binarySearchTree.insert(20)
binarySearchTree.insert(10)
binarySearchTree.insert(12)
binarySearchTree.insert(1)
binarySearchTree.insert(5)
binarySearchTree.insert(50);
binarySearchTree.remove(1);
binarySearchTree.root.left.left.value // 5
binarySearchTree.root.left.left.left // null
binarySearchTree.root.left.left.right // null

binarySearchTree.remove(20);
binarySearchTree.root.right.value // 50
binarySearchTree.root.right.right // null
binarySearchTree.root.right.left // null

var binarySearchTree = new BinarySearchTree();
binarySearchTree.insert(15)
binarySearchTree.insert(20)
binarySearchTree.insert(10)
binarySearchTree.insert(12)
binarySearchTree.insert(1)
binarySearchTree.insert(5)
binarySearchTree.insert(50)
binarySearchTree.insert(60)
binarySearchTree.insert(30)
binarySearchTree.insert(25)
binarySearchTree.insert(23)
binarySearchTree.insert(24)
binarySearchTree.insert(70);

binarySearchTree.remove(10);
binarySearchTree.root.left.value // 12
binarySearchTree.root.left.left.value // 1
binarySearchTree.root.left.left.right.value // 5

binarySearchTree.remove(50);
binarySearchTree.root.right.value // 20
binarySearchTree.root.right.right.value // 60
binarySearchTree.root.right.right.left.value // 30

var binarySearchTree = new BinarySearchTree();
binarySearchTree.insert(22)
binarySearchTree.insert(49)
binarySearchTree.insert(85)
binarySearchTree.insert(66)
binarySearchTree.insert(95)
binarySearchTree.insert(90)
binarySearchTree.insert(100)
binarySearchTree.insert(88)
binarySearchTree.insert(93)
binarySearchTree.insert(89)

binarySearchTree.remove(85);
binarySearchTree.root.right.right.value // 88
binarySearchTree.root.right.right.right.left.left.value // 89
```

isBalanced

Write a function which accepts a binary search tree and returns true if the tree is balanced, otherwise returns false.

findSecondHighest

Write a function which accepts a BST and returns the second highest value.

dfsInOrder Iteratively

Write another version of the dfsInOrder function but do not use recursion. This can be challenging. Think about what the computer is doing for you when you make a recursive call.

Solution

View our solutions <solution/index.html>