**Python code for search trees**

class Tree:

# Empty node has self.value, self.left, self.right = None

# Leaf has self.value != None, and self.left, self.right point to empty node

# Constructor: create an empty node or a leaf node, depending on initval

def \_\_init\_\_(self,initval=None):

self.value = initval

if self.value:

self.left = Tree()

self.right = Tree()

else:

self.left = None

self.right = None

return

# Only empty node has value None

def isempty(self):

return (self.value == None)

# Leaf nodes have both children empty

def isleaf(self):

return (self.left.isempty() and self.right.isempty())

# Convert a leaf node to an empty node

def makeempty(self):

self.value = None

self.left = None

self.right = None

return

# Copy right child values to current node

def copyright(self):

self.value = self.right.value

self.left = self.right.left

self.right = self.right.right

return

# Check if value v occurs in tree

def find(self,v):

if self.isempty():

return(False)

if self.value == v:

return(True)

if v < self.value:

return(self.left.find(v))

if v > self.value:

return(self.right.find(v))

# Insert value v in tree

def insert(self,v):

if self.isempty():

self.value = v

self.left = Tree()

self.right = Tree()

if self.value == v:

return

if v < self.value:

self.left.insert(v)

return

if v > self.value:

self.right.insert(v)

return

# Find maximum value in a nonempty tree

def maxval(self):

if self.right.isempty():

return(self.value)

else:

return(self.right.maxval())

# Delete value v from tree

def delete(self,v):

if self.isempty():

return

if v < self.value:

self.left.delete(v)

return

if v > self.value:

self.right.delete(v)

return

if v == self.value:

if self.isleaf():

self.makeempty()

elif self.left.isempty():

self.copyright()

else:

self.value = self.left.maxval()

self.left.delete(self.left.maxval())

return

# Inorder traversal

def inorder(self):

if self.isempty():

return([])

else:

return(self.left.inorder()+[self.value]+self.right.inorder())

# Display Tree as a string

def \_\_str\_\_(self):

return(str(self.inorder()))