Tyler Salas

1:30-2:50 Class

Dr. Fuentes

80596221

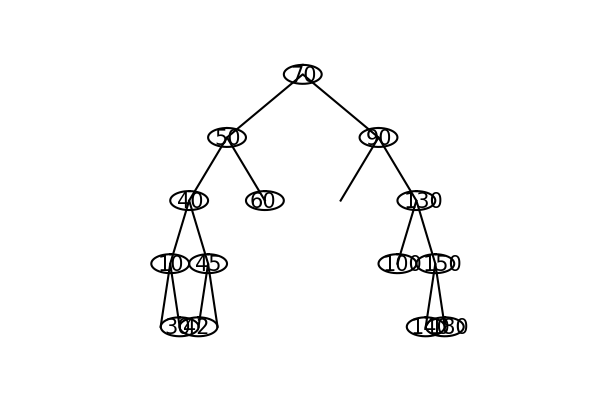
Lab 3 Report

Lab three, the lab this week, aimed to implement different uses and functions of a binary search tree, coded in Python. The methods we were asked to make in this lab started with the first being a visualization of a binary search tree, the next method being and iterative version of searching a binary search tree, the third method building a binary search tree out of a sorted list, the fourth method taking elements in a binary search tree and creating a sorted list out of the elements, and the last a method that prints all the given elements in each level of a binary search tree.

The basic idea to solve the given operations was to understand the order and structure of the binary search tree, and from that work the methods.

**Number 1**

This method was supposed to visualize a binary search tree. I could not get this method to work one hundred percent the way it was supposed to, but I had a basic idea of the visualization. The method I implement take a structure of figure used recently in lab one and creates the two branches out of out of the left and right side of a circle lying at the peak, containing the value and visualizing the node. My program checks to see if the next nodes of the current node (left and right) are null, and if so, it ceases operations, otherwise it goes down the nodes of the binary search tree printing the node and branch figures connected to their predecessors.



The issues I ran into with my tree seem to be the fact that the numbers I have are very large compared to the spacing between branches at lower levels. I experimented with different things to fix the different issues in the tree, but this was the best version I could manage to compile.

You can replicate the graphics of a different tree by inserting a different tree into the intial call on line 219.

**Number 2**

The second method we were asked to make is supposed to search for a node inside of the tree being searched. This method was like the one included in the code that Dr. Fuentes shared with us, thought it could not be recursive. This was the simplest of the methods to do, the basic idea just using a while loop that ceases if T is equal to the item being searched or T is equal to None. In each iteration of the while loop if it passes the base cases it tests if the item being searched is less or greater than the current T and goes to the right or left accordingly.

Searching for item 45 : 45

This is what the console displays after being run, you can change the number being searched by modifying the search value (s) on line 231. If the item being searched is not in the tree the program will print out None.

**Number 3**

The third method is supposed to create a balanced binary search tree out of a given sorted list. The idea I used to implement this method was using a binary search kind of idea to iterate through the array and find where subsequent nodes in the binary search tree would be placed in the list. The method requires a tree and a list, the initial call calling an empty tree. The method has a base case that ceases if the length of the list given is equal to zero. If it passes this case, it has another if statement that checks if the initial tree is equal to None. (Making it run only on the first recursive call) If it enters inside this if statement it creates a new tree using the middle element received by length of the list divided by two. It then goes to declare the left side of the newly created tree as a recursive call of the newly created tree along with the list with all the elements before the mid-point. Doing the same with the right side respectively inputting the right side of the list.

Making List [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] into a tree

10

9

8

7

6

5

4

3

2

1

This is what the console outputs for this segment of the code. If the user wanted to use a different list, they would simply change the list variable in line 244.

**Number 4**

This next method creates a sorted list out of a given binary search tree. This was the shortest method to write considering how easy the idea is. The initial call takes the tree and an empty list. From there it utilizes recursive in order traversal of the tree, each time appending the next item to the list ceasing its functions once T is equal to None.

Making Tree To List:

[10, 30, 40, 42, 45, 50, 60, 70, 90, 100, 130, 140, 150, 180]

This is what the program outputs to the console. The tree being used is the one below. If somebody wanted to use a different tree they’d simply change the tree variable in the initial call on line 254.

180

150

140

130

100

90

70

60

50

45

42

40

30

10

**Number 5**

This is the final method, we are supposed to print the level and all the elements that it holds. The initial call calls a method that gets the height of the tree and uses a for loop to call another method that prints the elements in a single level, thus covering all the levels in the tree. In the method called, it recursively runs till the n variable is equal to zero, covering all the elements in the tree and printing once all the activation records reach n = 0.

Items At Level 0:

70

Items At Level 1:

50 90

Items At Level 2:

40 60 130

Items At Level 3:

10 45 100 150

Items At Level 4:

30 42 140 180

This is what the program prints utilizing the list above. If somebody wanted to use a different list, they’d simply change the tree variable in line 263.

This lab really helped me in understanding binary search trees, their uses, function and structure. Working with ideas like in order traversal of the tree and building on ideas like binary search to create a BST were interesting in opening new ways to look out past learnt ideas.

#CS2302

#Tyler Salas

#Lab3

#Dr.Fuentes

#Anindita Nath

#Implemnt Binary Tree Functions

import time

import matplotlib.pyplot as plt

import numpy as np

import math

class BST(object):

# Constructor

def \_\_init\_\_(self, item, left=None, right=None):

self.item = item

self.left = left

self.right = right

def Insert(T,newItem):

if T == None:

T = BST(newItem)

elif T.item > newItem:

T.left = Insert(T.left,newItem)

else:

T.right = Insert(T.right,newItem)

return T

def Delete(T,del\_item):

if T is not None:

if del\_item < T.item:

T.left = Delete(T.left,del\_item)

elif del\_item > T.item:

T.right = Delete(T.right,del\_item)

else: # del\_item == T.item

if T.left is None and T.right is None: # T is a leaf, just remove it

T = None

elif T.left is None: # T has one child, replace it by existing child

T = T.right

elif T.right is None:

T = T.left

else: # T has two chldren. Replace T by its successor, delete successor

m = Smallest(T.right)

T.item = m.item

T.right = Delete(T.right,m.item)

return T

def getHeight(T):

if T is None:

return 0

else:

leftHeight = getHeight(T.left)

rightHeight = getHeight(T.right)

if leftHeight > rightHeight :

return leftHeight+1

else:

return rightHeight+1

def InOrder(T):

# Prints items in BST in ascending order

if T is not None:

InOrder(T.left)

print(T.item,end = ' ')

InOrder(T.right)

def InOrderD(T,space):

# Prints items and structure of BST

if T is not None:

InOrderD(T.right,space+' ')

print(space,T.item)

InOrderD(T.left,space+' ')

def SmallestL(T):

# Returns smallest item in BST. Returns None if T is None

if T is None:

return None

while T.left is not None:

T = T.left

return T

def Smallest(T):

# Returns smallest item in BST. Error if T is None

if T.left is None:

return T

else:

return Smallest(T.left)

def Largest(T):

if T.right is None:

return T

else:

return Largest(T.right)

def Find(T,k):

# Returns the address of k in BST, or None if k is not in the tree

if T is None or T.item == k:

return T

if T.item<k:

return Find(T.right,k)

return Find(T.left,k)

def FindAndPrint(T,k):

f = Find(T,k)

if f is not None:

print(f.item,'found')

else:

print(k,'not found')

#Searches for a node with while loop iterations (No.2)

def iterativeSearch(T,k):

if T is None:

return -1

t = T

while t is not None and t.item != k:

if k < t.item:

t = t.left

else:

t = t.right

if t is None:

return None

else:

if t is None:

return -1

return t

#Sorted List to a Tree (No.3)

def listToTree(T,l):

if len(l) == 0:

return None

mid = len(l)//2

if T is None:

head = BST(l[mid])

head.left = listToTree(T,l[:mid])

head.right = listToTree(T,l[mid+1:])

return head

#Tree To Sorted List (No.4)

def treeToList(T,l):

if T is not None:

treeToList(T.left,l)

l.append(T.item)

treeToList(T.right,l)

#(No.5)

#Iterates through all the levels to print each one using the printLevel function

def printLevels(T):

hgt = getHeight(T)

c = 0

while c < hgt:

print()

print("Items At Level ",c, end='')

print(": ")

printLevel(T, c)

c += 1

#Used by printLevels function to print a certain level given by n

def printLevel(T,n):

if T is None:

return

if n == 0:

print(T.item,end=' ')

elif n > 0 :

printLevel(T.left ,n-1)

printLevel(T.right ,n-1)

def circle(center,rad):

n = int(4\*rad\*math.pi)

t = np.linspace(0,6.3,n)

x = center[0]+rad\*np.sin(t)

y = center[1]+rad\*np.cos(t)

return x,y

#Visualizes tree

def draw\_Branch(ax,T,p,w,length,rad):

if T is not None:

if T.left is None and T.right is None:

center = [p[1,0],p[1,1]]

x,y = circle(center,rad)

ax.plot(x,y,color='k')

ax.text(center[0]-60,center[1]-60,T.item,fontsize=15)

return

center = [p[1,0],p[1,1]]

x,y = circle(center,rad)

ax.plot(x,y,color='k')

ax.text(center[0]-50,center[1]-60,T.item,fontsize=15)

length1 = p[0,0] - p[2,0]

length1 = length1 \* -1

length = (p[0,0]) - (p[2,0])

length = (length//2)

c1 = [[length//2,-500],[length,-500],[length\*1.5,-500]]

c2 = [[(length//2)+length1,-500],[length+length1,-500],[(length\*1.5)+length1,-500]]

q1 = p + c1

q2 = p + c2

ax.plot(p[:,0],p[:,1],color='k')

ax.plot(100,100)

draw\_Branch(ax,T.left,q1,w,length,rad)

draw\_Branch(ax,T.right,q2,w,length,rad)

def drawBranch1(ax,br,center,ang):

br[0,0] = center[0] -20 + np.sin(ang)

br[0,1] = center[1] -20 + np.cos(ang)

ax.plot(br[:,0],br[:,1],color='k')

return [br[0,0],br[0,1]]

#Code to visualize a tree

z = None

A = [70, 50, 90, 130, 150, 40, 10, 30, 100, 180, 45, 60, 140, 42]

for a in A:

z = Insert(z,a)

start = time.time()

plt.close("all")

fig, ax = plt.subplots()

ax.axis('off')

ax.set\_aspect(.50)

p = np.array([[-300,-500],[0,0],[300,-500]])

draw\_Branch(ax,z,p,.1,-600,75)

fig.savefig('circles1.png')

# Code to test the functions above

T = None

A = [70, 50, 90, 130, 150, 40, 10, 30, 100, 180, 45, 60, 140, 42]

for a in A:

T = Insert(T,a)

#Test case No.2

s = 90

print("Searching for item",s,": ",end='')

it = iterativeSearch(T,s)

if it is not None:

print(it.item)

else:

print(None)

print()

#Test case No.3

t = None

B = [1,2,3,4,5,6,7,8,9,10]

print("Making List ",B," into a tree")

b = listToTree(t,B)

InOrderD(b,' ')

print()

#test case No.4

E = None

A = [70, 50, 90, 130, 150, 40, 10, 30, 100, 180, 45, 60, 140, 42]

for a in A:

E = Insert(E,a)

l = []

treeToList(E,l)

print("Making Tree To List: ")

print(l)

print()

#test case No.5

print("Printing Tree Levels Of T: ")

t = None

InOrderD(T,' ')

printLevels(T)

print()

print()

print("Binary Tree Visualized")

I certify that this project is entirely my own work. I

wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also

certify that I did not share my code or report or provided inappropriate assistance to any student in the class.



-Tyler Salas