Marco Valerio

Lab 3 Report

Olac Fuentes

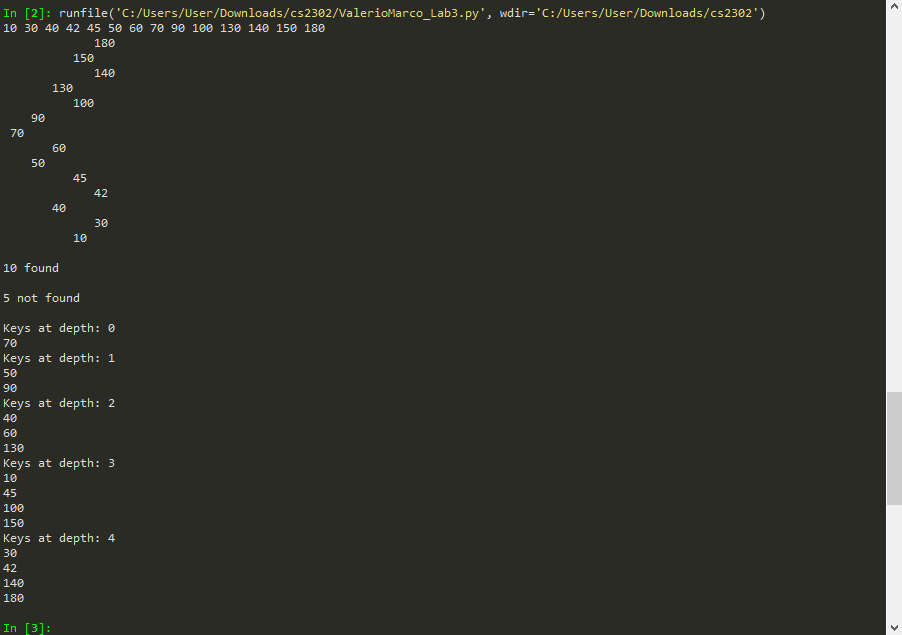
CS 2302

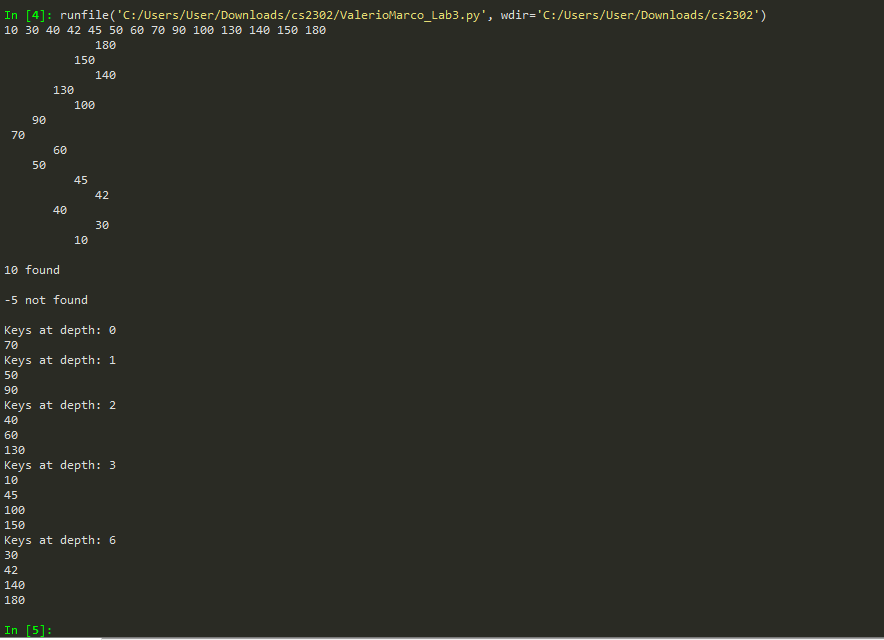
03/12/2019

***Introduction:*** Lab 3, focused on the student’s knowledge of the implementation and design of binary search trees. Our tasks were to create a visual representation of a binary search tree, a iterative version of the search method, build a balanced BST without the insert method, extract the elements of a BST into a list, and finally print the elements of the tree at each depth.

***Proposed Solution:*** When it came the design of the BST I was unable to get the design, I attempted to use “matplotlib.pyplot”, but I still the knowledge of how to properly create and implement the designs recursively. My intentions were to plot a circle and at the center place the number and use the binary tree method from the one of the previous labs. For the iterative search, I assumed it just meant to search using while loops, so I used a while to compare ‘k’ to the item of the current ‘T’ and choose routes based on the outcome until the tree is None or it finds the ‘k’. The search function is hardcoded, I had made to take user input, but the instructions did not mention to do this. I had no attempts for the list creator and extractor, I was not able to think a method that could create a list without the insert, I believe it is necessary. Lastly, for the keys at all depths I created a method that would traverse the BST using recursive calls dependent on ‘k’ and would subtract with each call and would print once ‘k’ was 0; there were two different calls for the right and left leaf.

***Experimental Results:*** My experiments for the iterative search function were searching for a number that was in the tree, one that was not, and a negative number by calling the method ‘searchIt(T, k)’; ‘k’ being one of the test numbers. At first the method would crash if it tried to search for a number that was not in the list, but I later realized this was because the clause to exit the while loop just needed to be an ‘else’ instead of another ‘if k = T.item’. For the depth, I created different calls for each depth and one for a depth that didn’t exist. I had also made this method to take user input to check whichever depth the user wanted, but the lab requested it check and display all depths.





***Conclusion:*** In conclusion, I can say that this lab has improved my understanding of BST in the means of traversing both recursively and iteratively. I still need to look into ‘matplotlib.pyplot’ because I believe this is going to be a recurring theme even though this is a data structures class, and not designs.

***Source Code:***

# -\*- coding: utf-8 -\*-

"""

Lab 3

Professer: Olac Fuentes

MW 10:30-11:50

@author: Marco Valerio

"""

# Code to implement a binary search tree

# Programmed by Olac Fuentes

# Last modified February 27, 2019

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 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)

#The iterative version of find, the while serves the function of the recursive call

def searchIt(T,k):

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

return T

while T is not None:

if k > T.item:

T = T.right

if k < T.item:

T = T.left

else:

return T

#This method traverse the list until K is zero and prints all on that level

def printD(T,k):

if T is None:

return None

if k == 0:

print(T.item, ' ')

else:

printD(T.left, k-1)

printD(T.right, k-1)

def FindAndPrint(T,k):

t = searchIt(T,k)

if t is not None:

print(t.item,'found')

else:

print(k,'not found')

# 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)

InOrder(T)

print()

InOrderD(T,'')

print()

FindAndPrint(T,10)

print()

FindAndPrint(T,5)

print()

print('Keys at depth:', 0)

printD(T,0)

print('Keys at depth:', 1)

printD(T,1)

print('Keys at depth:', 2)

printD(T,2)

print('Keys at depth:', 3)

printD(T,3)

print('Keys at depth:', 4)

printD(T,4)

***Academic Statement:*** “I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, preformed 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.”