CS3 Report Lab Three

Daniela Flores

Intro:

Lab three consisted of adding various functions to add to the BST program already provided to us by Dr.Fuentes. We had to plot a BST figure with the help of the program we did for lab1. We also had to write a method to iteratively search for a given value in the BST, a method to get the elements from a BST to an array, build a balanced binary tree with a given sorted list, and to write a method that'll print the elements of the arrays by their depths.

Proposed Solution Design and Implementation:

I first decided to start write the method that'll order the elements of the tree by their depth. In this method I passed the BST as the parameter. I then initialized a counter so that I'll be able to keep track of the depth. A while loop was used that'll iterate though the tree till the it reaches the leafs. inside the while loop a list is created which will be used to store the elements of the current depth. A for loop is then used to iterate through the elements of the list to check if those elements have children, and if they do, they'll be stored in the list which was created inside the while loop. The counter is then incremented by one and the current list is then replaced with the contents in the list that was being populated with the children of the elements being printed. The while loop will keep iterating till the list has no elements stored in it.

Th next method I wrote was used to iteratively search for a value within the tree. The parameters for this method was the BST and the value being searched for. An if statement is used to check whether if the BST is empty, and if it is to return false. A while loop is used to iterate through the tree. One if statement within the loop checks if the item or num in the node being passed is the same of the value being searched, and if they're equal, to return true. After that if statement an elif statements checks whether that value is less than the value of 'T,item' and if it is, the variable T will be updated to the left subtree. Another elif statement checks if the value is greater than the value of T,item. If the value is bigger, the right subtree will be traversed by updating the variable T to T.right. Outside the while loop a return False is implemented in case the whole tree is traversed and the value was not found, that'll mean that the value is indeed not in the BST.

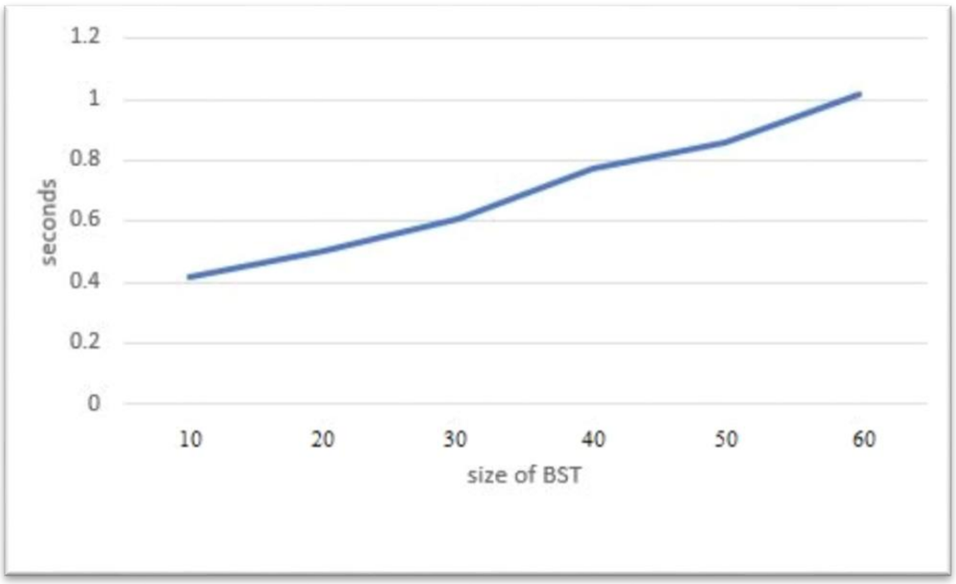
I implemented the method to get the elements within the BST into a sorted list by first setting the parameters of this method the BST and an empty list. An if statement or base case is then implemented to check if the tree is equal to None and if it is to return 0. A recursive call is made with the left subtree, after it the value of the current node is appended to the empty list, afterwards another recursive call is made with the right subtree. Lastly the populated list is returned.

To implement the method to create a balanced tree when given a sorted list I first set the parameters of this method to be a sorted list L. An if statement then checks if the length of the list is not zero, and if it's not, inside the if statement a variable gets the middle element of the list by dividing the length of the list by 2. Afterwards a variable that'll be the balanced tree, is initialized, and the root is set to be the middle element of the list previously found. The left subtree is set by recursively calling the method with the first half of the list and the right subtree by recursively calling the method with the second half of the sorted list. After the two recursive calls, the balanced tree is returned.

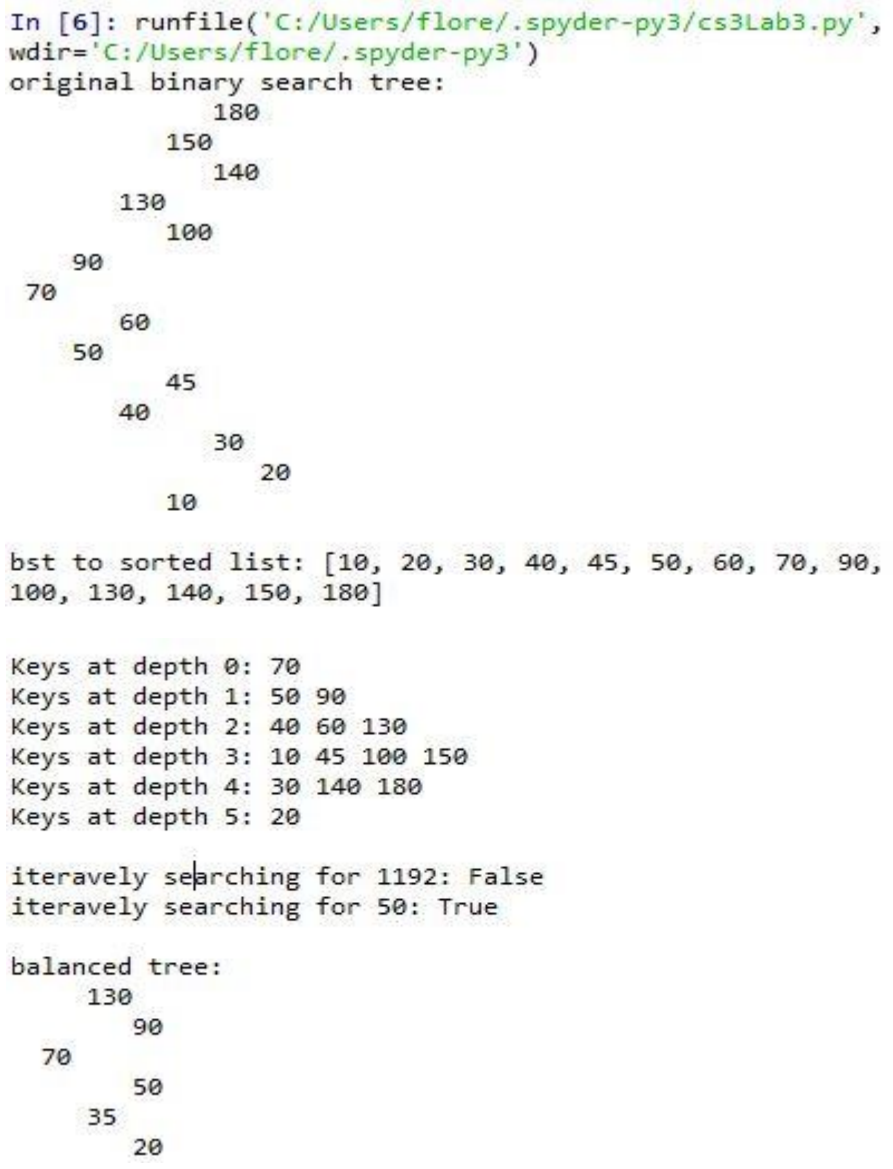
The method to draw a figure of BST parameter's was a set of coordinates, the BST and the amount of units one wanted to change coordinates x and y. inside of the method an if statement is used to check if the tree has a left child, and if it does a line is drawn for the left child, after that a recursive call is made with the left subtree and with different coordinates and different units of change for x and y. Another if statement checks if the tree has a right child, and if it does, a line drawn for it. A recursive call is made with the parameters being the right subtree and different units of change for x and y. outside the if statement a circle is plotted with the size being 23 white('wo') filling, and black edge color on the assigned coordinates. After the circles are drawn, a bolded text of size 8 is added inside the circles with the values in the BST.

Experimental Results

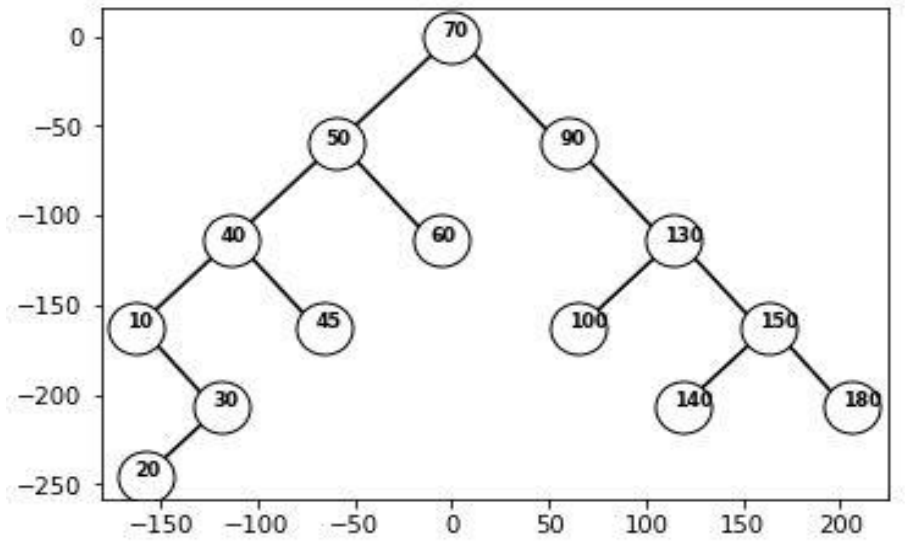
I experimented with my methods by changing the values of the BST that was being passed as parameter. Changing the values of the BST helped me discover some bugs that were in my code. Below is a graph of the different runtimes when the size of the BST increased:



Below there's screenshots of the output of my program:



```
In [6]: runfile('C:/Users/flore/.spyder-py3/cs3Lab3.py',
wdir='C:/Users/flore/.spyder-py3')
original binary search tree:
             180
          150
             140
       130
          100
    90
 70
       60
    50
          45
       40
             30
                20
          10

bst to sorted list: [10, 20, 30, 40, 45, 50, 60, 70, 90,
100, 130, 140, 150, 180]


Keys at depth 0: 70
Keys at depth 1: 50 90
Keys at depth 2: 40 60 130
Keys at depth 3: 10 45 100 150
Keys at depth 4: 30 140 180
Keys at depth 5: 20

iteravely searching for 1192: False
iteravely searching for 50: True

balanced tree:
     130
        90
  70
        50
     35
        20
```



Conclusions:

What I learned from this lab was how to traverse though a tree and how this data structure can be useful. I also learned the difference between a BST and a balanced tree. I also now have a better understanding on what goes on behind the curtains when one uses recursion to traverse the right and left subtrees of a BST.

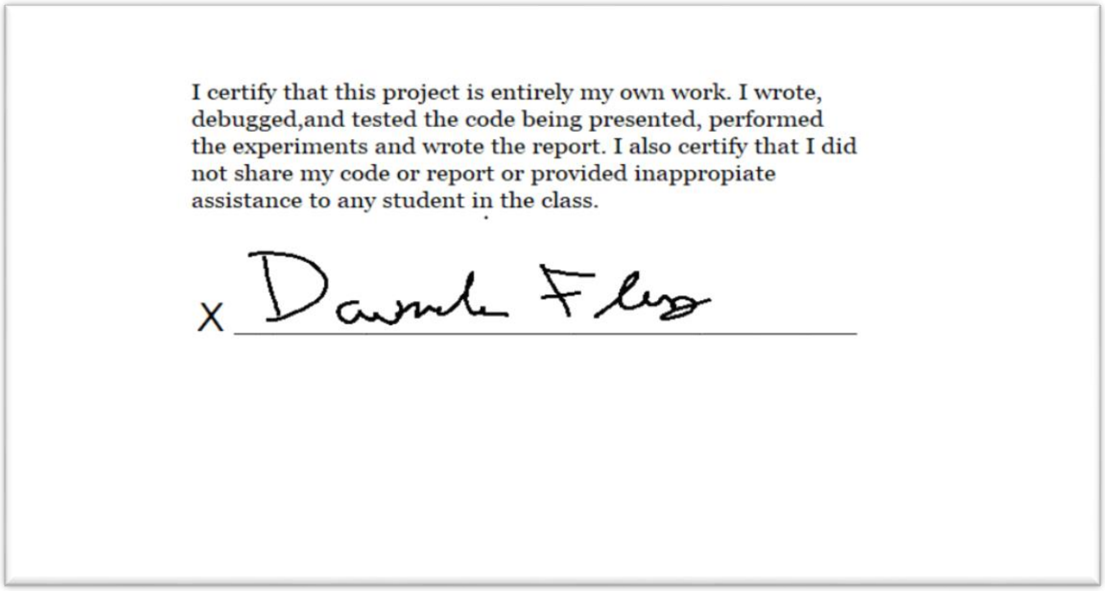
Appendix:

```
#Course:CS2302

#aAuthor:Daniela Flores

#Lab3

#Instructor:Olac Fuentes

#T.A: Dita Nath

#date of last Mod: 3/8/19

#purpose of program: in this program I had to write various functions to better understand binary seach trees.

import matplotlib.pyplot as plt


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 Smallest(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 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')

def height(bst):
    if bst is None:
        return 0
    else:
        return 1 + max(height(bst.left), height(bst.right))
######################################################

# traverses thtough a bst iteravely to search for a given value
def iterativeSearch(T, x):
    # Base Case
    if (T == None):
        return False
    while T != None:
        if x == T.item:
            return True
        #if root is bigger than element given, itll keep iterating left subtree
        elif x < T.item:
            T = T.left
         #if root is smaller than element given, itll keep iterating right subtree
        elif x > T.item:
            T = T.right
    #if done iterating though tree and element not found, return false
    return False

##############################################################
#prints all elements throught the different depths of bst
def treeDepth(T):
  #makes a copy of T
  currentL = [T]
  #counter
  i = 0
  #iterates till no more items are added to list
  while len(currentL) != 0:
    print('Keys at depth %d:' % i, end=' ' )
    #create empty list
    nextL = list()
    #iterates through elements of currentL
    for n in currentL:
      print (n.item, end = ' ')
      if n.left is not None:
         nextL.append(n.left)
      if n.right is not None:
         nextL.append(n.right)
    print('')
    i = i+1
    currentL = nextL

#########################################################
#The recursive step makes two recursive calls to find the sum of the sub trees.
# When the recursive calls complete, we add the parent and return the total
def total (T):
    if T is None:
        return 0
    return total(T.left)+ total(T.right) +T.item
####################################################
#gets tree and returns sorted list
def treeToSortedL(tree,A):
    if tree == None:
        return 0
    treeToSortedL(tree.left,A)
    A.append(tree.item)
    treeToSortedL(tree.right,A)
    return A

######################################################
#draws a tree figure
def treeFig(ax, c, T, dx, dy):
    #checks if theres a left child
    if T.left is not None:
        #line is drawn for left child
        ax.plot([c[0], c[0]-dx], [c[1], c[1]-dy], color='k')
        #recursion call with left child as root
        treeFig(ax, [c[0]-dx, c[1]-dy], T.left, dx*0.9, dy*0.9)
    #checks if theres a right child
    if T.right is not None:
        #line is drawn for right child
        ax.plot([c[0], c[0]+dx], [c[1], c[1]-dy], color='k')
        #recursive call with right child in parameter
        treeFig(ax, [c[0]+dx+3, c[1]-dy+3], T.right, dx*0.9, dy*0.9)
    #plots a circle at given coordinates
    plt.plot(c[0], c[1],'wo', markersize=23, markeredgecolor='black')
    #nodes value is inserted inside drawn circle
    ax.text(c[0]-5, c[1], T.item, size=8, weight='bold')
#########################################################
# when given a list it creates a balanced tree
def balancedTree(L):
    if L:
        # gets the middle element of the array
        midEl = len(L)//2
        # sets the root of the BTree to middle element
        T = BST(L[midEl])
        T.left = balancedTree(L[:midEl])
        T.right = balancedTree(L[midEl+1:])
        return T
#######################################################

# Code to test the functions above
T = None
A = [70, 50, 90, 130, 150, 40, 10, 30, 100, 180, 45, 60, 140]
```

```
for a in A:
    T = Insert(T,a)



B = [70, 50, 90, 130,35,20]
B.sort()



print('original binary search tree:')
InOrderD(T,'')
print()

List=[]
print('bst to sorted list:',treeToSortedL(T,List))
print('')
treeDepth(T)
print('')
print('iteravely searching for 1192:',iterativeSearch(T,1192))
print('iteravely searching for %d:'% A[1],iterativeSearch(T,A[1]))
print('')
#creates and prints out balanced tree
print('balanced tree:')
BT = balancedTree(B)
InOrderD(BT,' ')
#draws Tree figure
plt.close("all")
fig, ax = plt.subplots()
treeFig(ax, [0,0], T, 60, 60)
ax.set_aspect(1.0)
ax.axis('on')

plt.show()

    fig.savefig('trees.png')
```



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 inappropiate assistance to any student in the class.

X [illegible]