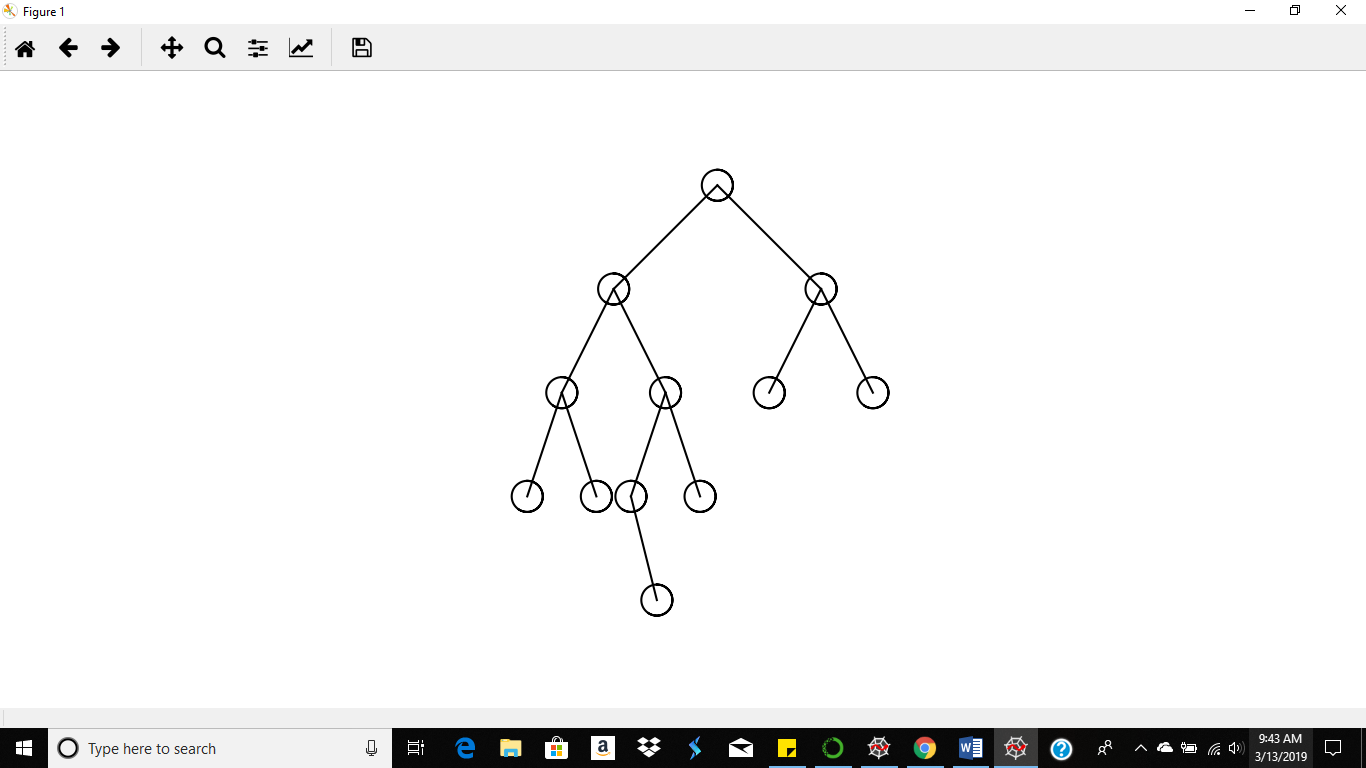
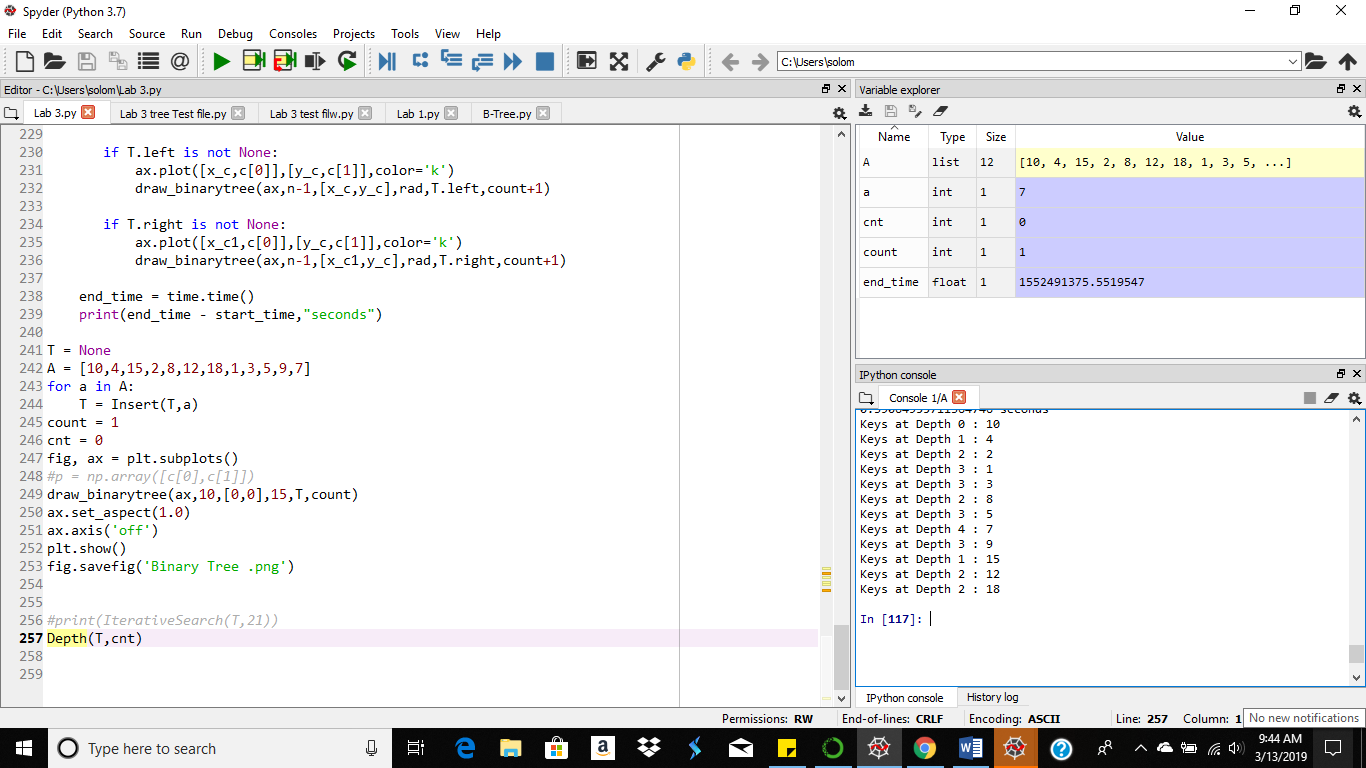
Solomon Davis Lab 3 Report

CS 2302 - MW 1:30 Spring 2019

In Lab 3 the code is supposed to put the items in the binary search tree, use a search method without recursion, sort the items in a binary tree, list the keys at a specific depth, and creating a balanced tree. When drawing the binary search tree, I was unable to create blank circles so I can input the data in the tree.

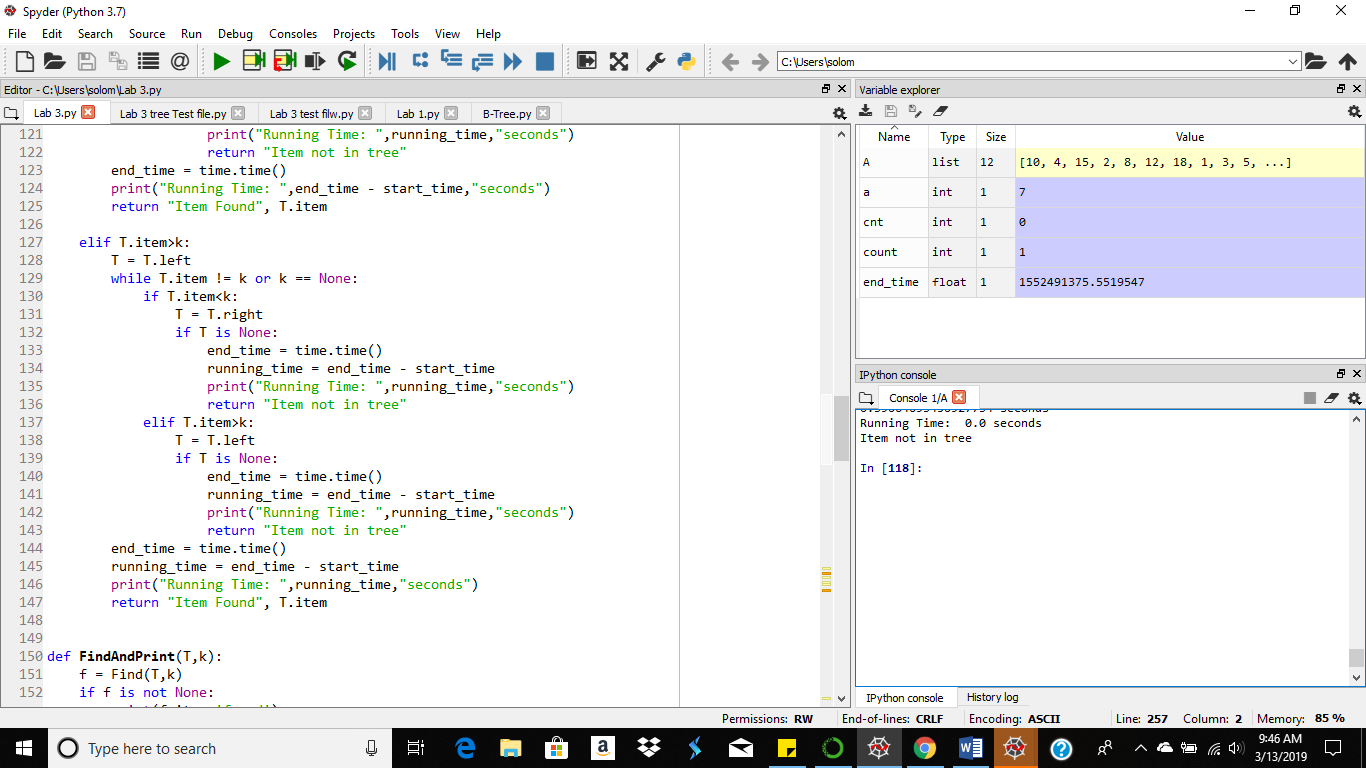


To find the keys at the specific depths recursively pulled the items from the tree and kept and count for the depth.

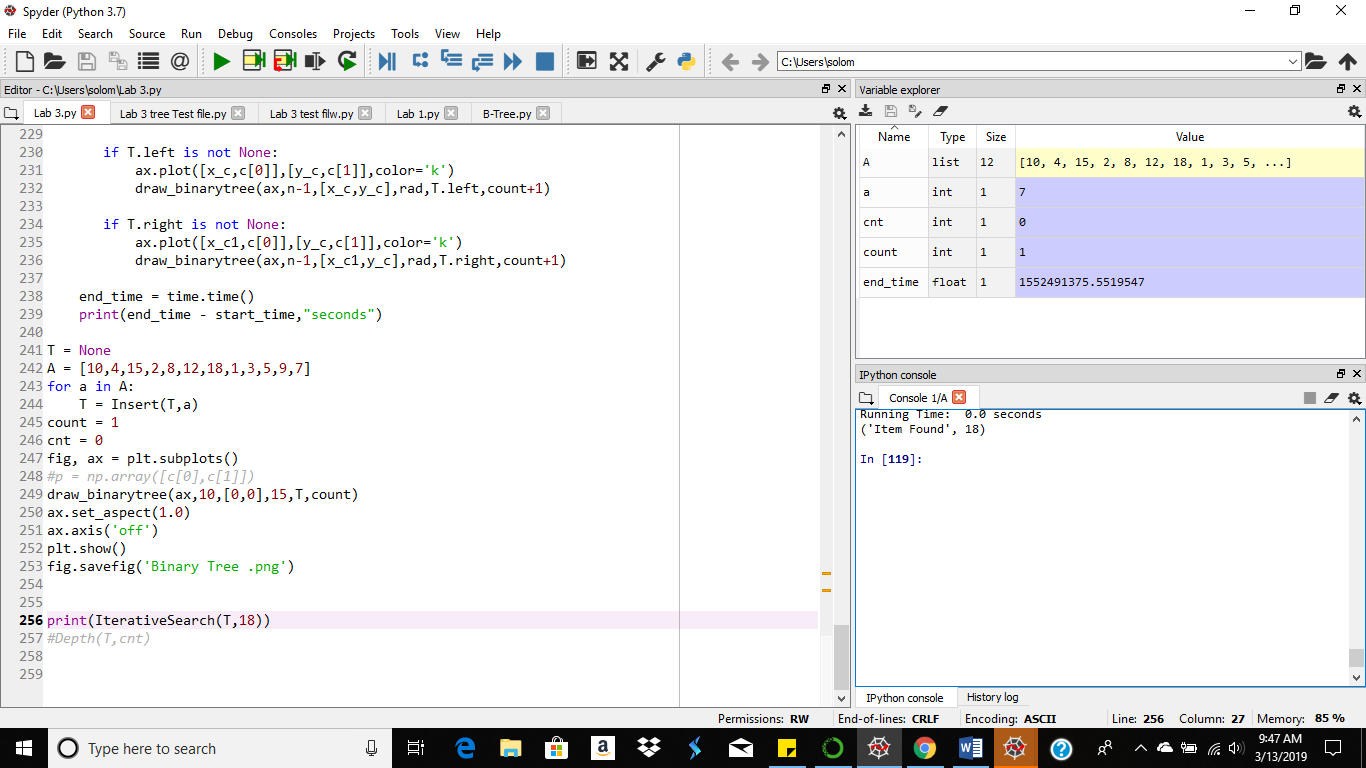


For the search operation I compare the search value to the first item in the tree. If its equal I return output saying item found and the matching value. If the search key is less than the first item in the tree I move to the right side of the tree and begin to compare all the values on that side of the tree. If the search value is greater than the initial item, I traverse the left side of the tree and begin tom compare all the values on that side of the tree. If the item is not found in the binary search tree, then the output will show the “item wasn’t found on the screen”.

When Item is not found

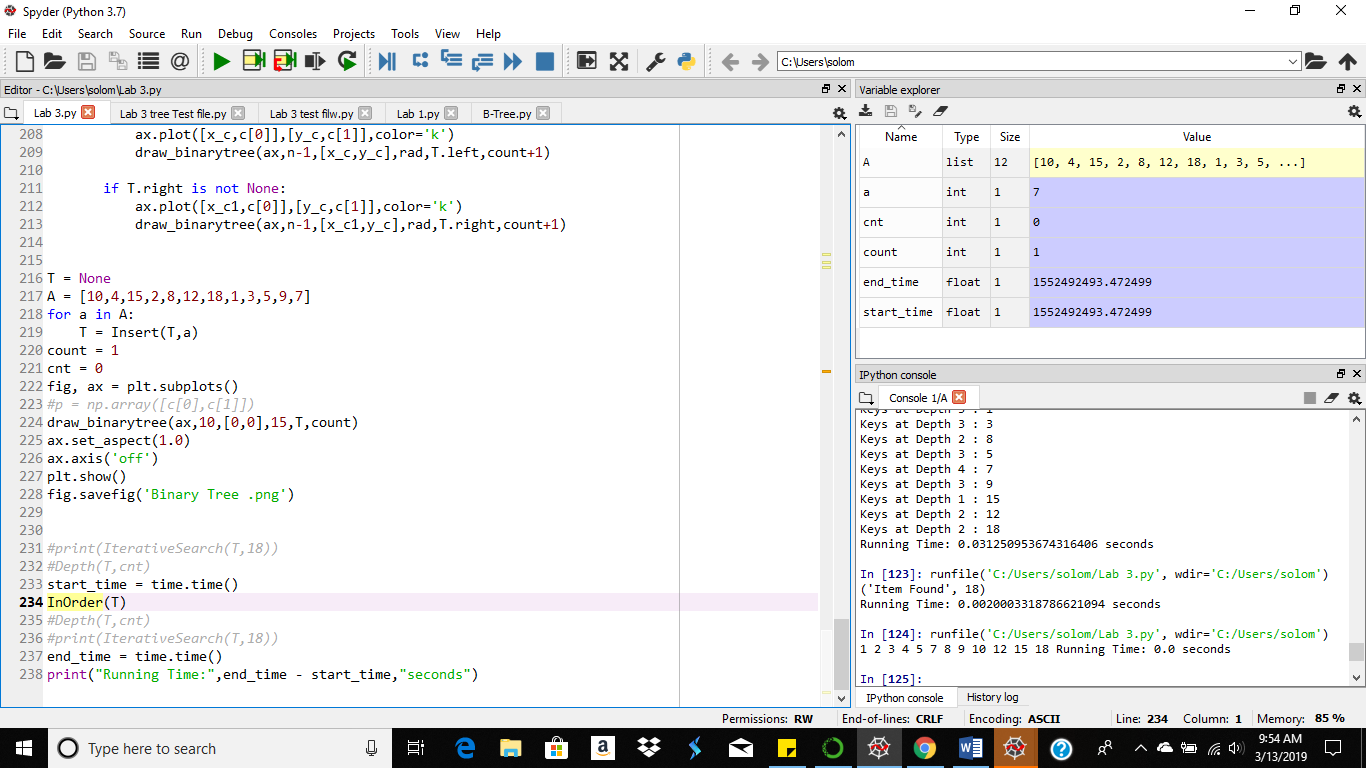


When Item is found



For putting the binary search tree items in order, the function is simply displaying the output of the left side of the tree then the items in the right side of the tree. To find the keys at the specific depths recursively pulled the items from the tree and kept and count for the depth.

Items in Order



#Course: CS2302 - Spring 2019

#Author: Solomon Davis

#Lab Number: 3

#Instructor: Olac Fuentes

#Last Modified: March 13, 2019

#Due Date: March 13, 2019

#Description: This code will display keys at a certain depth, and search for

# keys without recursion. I was unable to complete the binary

# search tree as I was unable to fill the circles correctly with

# the data.

import numpy as np

import matplotlib.pyplot as plt

import math

import time

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)

def IterativeSearch(T,k):

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

return "Item Found", T.item

elif T.item<k:

T = T.right

while T.item != k or k == None:

if T.item<k:

T = T.right

if T is None:

return "Item not in tree"

elif T.item>k:

T = T.left

if T is None:

return "Item not in tree"

return "Item Found", T.item

elif T.item>k:

T = T.left

while T.item != k or k == None:

if T.item<k:

T = T.right

if T is None:

return "Item not in tree"

elif T.item>k:

T = T.left

if T is None:

return "Item not in tree"

return "Item Found", T.item

def FindAndPrint(T,k):

f = Find(T,k)

if f is not None:

print(f.item,'found')

else:

print(k,'not found')

def ReverseOrder(T):

# Prints items in BST in descending order

if T is not None:

ReverseOrder(T.right)

print(T.item, end =' ')

ReverseOrder(T.left)

def NumberOfNodes(T):

if T is None:

return 0

return 1 + NumberOfNodes(T.right) + NumberOfNodes(T.left)

def Range(T):

return Largest(T) - Smallest(T)

def Valid(T):

if T is None:

return True

else:

if T.left is not None and T.left.item > T.item:

if T.right is not None and T.right.item < T.item:

return False

return Valid(T.left) and Valid(T.right)

def ElementsInASortedListed(T,T2):

if T is not None:

InOrderD(T.left,' ')

print(T.item,end = ' ')

T2 = Insert(T2,T.item)

InOrderD(T.right,' ')

def KeysAtDepth(T,d):

if T is None:

return ''

if d == 0:

return T.item

else:

return KeysAtDepth(T.left,d-1) , KeysAtDepth(T.right,d-1)

def Depth(T,cnt):

if T is not None:

print("Keys at Depth",cnt,':',end=' ')

print(T.item)

Depth(T.left,cnt+1)

Depth(T.right,cnt+1)

def circles(c,rad):

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

t = np.linspace(0,10,n)

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

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

return x,y

def draw\_binarytree(ax,n,c,rad,T,count):

x\_c = c[0]-(100/count)

y\_c = c[1]-100

x\_c1 = c[0]+(100/count)

if n>0:

x,y = circles(c,rad)

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

ax.fill(x,y,'white')

if T.left is not None:

ax.plot([x\_c,c[0]],[y\_c,c[1]],color='k')

draw\_binarytree(ax,n-1,[x\_c,y\_c],rad,T.left,count+1)

if T.right is not None:

ax.plot([x\_c1,c[0]],[y\_c,c[1]],color='k')

draw\_binarytree(ax,n-1,[x\_c1,y\_c],rad,T.right,count+1)

T = None

A = [10,4,15,2,8,12,18,1,3,5,9,7]

for a in A:

T = Insert(T,a)

count = 1

cnt = 0

fig, ax = plt.subplots()

#p = np.array([c[0],c[1]])

draw\_binarytree(ax,10,[0,0],15,T,count)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('Binary Tree .png')

#print(IterativeSearch(T,18))

#Depth(T,cnt)

start\_time = time.time()

InOrder(T)

Depth(T,cnt)

print(IterativeSearch(T,18))

end\_time = time.time()

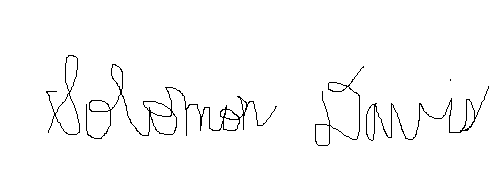
print("Running Time:",end\_time - start\_time,"seconds")

RUNNING TIMES(seconds) – O(n)

|  |  |
| --- | --- |
| Binary Search Tree | 0.5937869548797607 seconds |
| Search | 0.0 seconds |
| InOrder | 0.023006677627563477 seconds |
| Keys at Depth | 0.01562047004699707 seconds |

Academic Service Certificate:

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.



Solomon Davis