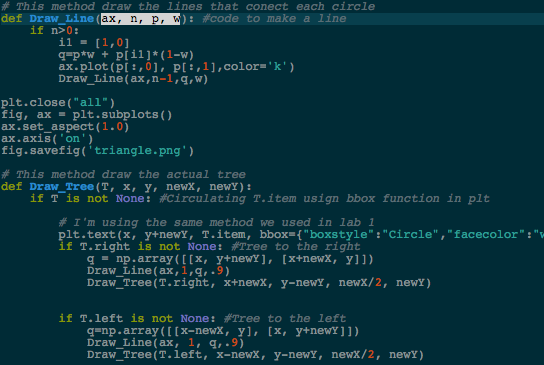
**Lab #3 Report**

**Introduction:**

The main purpose of this assignment was to learn how to use binary trees as a data structure. For this lab assignment we are making a binary search tree in which first we draw the tree, second we find an element in the tree, third we convert a list to a binary tree, fourth we convert a tree to a list, and finally we print the elements in a binary tree ordered by depth.

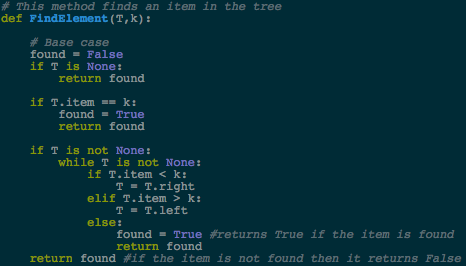
**Task 1 - Drawing the Tree:**

For this task I had to use the the matplot library to create the lines and the circles. I used the methods we used for lab #1 to draw the lines and connect them. I created to methods one name *Draw\_Lines()* is in charge of drawing the lines by connecting the vertices. And also I created a method named *Draw\_Tree()* that actually draws the tree. This last method recursively draw lines to the left and right and divide its x-axis by 2.

 O(n)

**Task 2 - Finding the element in the tree:**

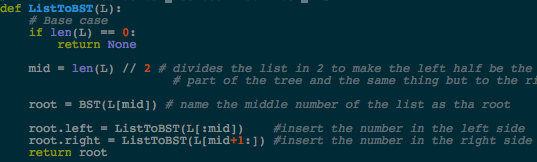
For this task I created a method that first creates a boolean variable that return True if the item is found and False if the item is not found. To transverse over the tree in O(n) time I had to do a while loop that loops until the tree is None and it looks for the number by going to each side.



O(n)

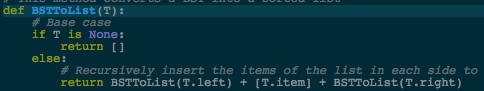
**Task 3 - Convert a sorted list into a BST:**

This method first divides the list by 2 that way the number in the middle can be the root, and then from there, recursively sends the items in the first half to the left side of the tree and the second half of the list to the right part of the tree.

 O(n^2)

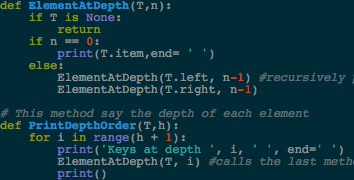
**Task 4 - Converting a BST to a list:**

I think this was the easier because if the tree is None the it will return an empty list. If not, then just recursively send the elements of the left, right, and the middle of the list, to the tree.

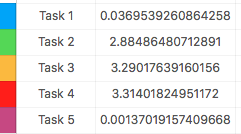
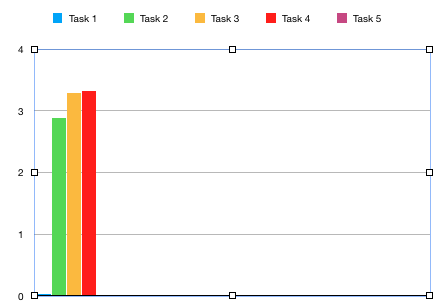
 O(n^2)

**Task 5 - Print the elements in the binary tree ordered by depth:**

This method says the depth of this element. First I created a method that prints each element of the tree and other that, in a for loop, print the element that are in the tree when the depth is 0, 1, and so on.

 O(n^2)

**Running times:**

****

**Conclusion:**

I think that this lab assignment was really helpful for us because now we know how to work with binary search trees. Also this helped me because know I feel more confident when using recursion in my codes.

﻿#Davis, David 80610756

#CS 2302. Lab 2: Binary Search Tree

#In this lab assignment we are going to work with a binary search tree in which.

#we will look for certain numbers in the tree, will convert a list to a BST and

#viceversa, will draw the tree by using matplot library, and will find the depth

#of all elements

########################### PRE-METHODS #################

import matplotlib.pyplot as plt

import numpy as np

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 FindAndPrint(T,k):

f = Find(T,k)

if f is not None:

print(f.item,'found')

else:

print(k,'not found')

############################# METHODS FOR LAB 3 ###############################

# This method draw the lines that conect each circle

def Draw\_Line(ax, n, p, w): #code to make a line

if n>0:

i1 = [1,0]

q=p\*w + p[i1]\*(1-w)

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

Draw\_Line(ax,n-1,q,w)

plt.close("all")

fig, ax = plt.subplots()

ax.set\_aspect(1.0)

ax.axis('on')

fig.savefig('triangle.png')

# This method draw the actual tree

def Draw\_Tree(T, x, y, newX, newY):

if T is not None: #Circulating T.item usign bbox function in plt

# I'm using the same method we used in lab 1

plt.text(x, y+newY, T.item, bbox={"boxstyle":"Circle","facecolor":"white"})

if T.right is not None: #Tree to the right

q = np.array([[x, y+newY], [x+newX, y]])

Draw\_Line(ax,1,q,.9)

Draw\_Tree(T.right, x+newX, y-newY, newX/2, newY)

if T.left is not None: #Tree to the left

q=np.array([[x-newX, y], [x, y+newY]])

Draw\_Line(ax, 1, q,.9)

Draw\_Tree(T.left, x-newX, y-newY, newX/2, newY)

# This method finds an item in the tree

def FindElement(T,k):

# Base case

found = False

if T is None:

return found

if T.item == k:

found = True

return found

if T is not None:

while T is not None:

if T.item < k:

T = T.right

elif T.item > k:

T = T.left

else:

found = True #returns True if the item is found

return found

return found #if the item is not found then it returns False

# This method converts a sorted list into a BST

def ListToBST(L):

# Base case

if len(L) == 0:

return None

mid = len(L) // 2 # divides the list in 2 to make the left half be the left

# part of the tree and the same thing but to the right

root = BST(L[mid]) # name the middle number of the list as tha root

root.left = ListToBST(L[:mid]) #insert the number in the left side

root.right = ListToBST(L[mid+1:]) #insert the number in the right side

return root

# This method converts a BST into a sorted list

def BSTToList(T):

# Base case

if T is None:

return []

else:

# Recursively insert the items of the list in each side to create the tree

return BSTToList(T.left) + [T.item] + BSTToList(T.right)

# This method is used to print the item at a certain depth

def ElementAtDepth(T,n):

if T is None:

return

if n == 0:

print(T.item,end= ' ')

else:

ElementAtDepth(T.left, n-1) #recursively prints the item of each side

ElementAtDepth(T.right, n-1)

# This method say the depth of each element

def PrintDepthOrder(T,h):

for i in range(h + 1):

print('Keys at depth ', i, ' ', end=' ')

ElementAtDepth(T, i) #calls the last method to print the items of each depth

print()

# Code to test the functions above

T = None

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

for a in A:

T = Insert(T,a)

#L = [1,3,5,6,7,8,9,10,15,22,44]

L = [1,3,5,6,7,8,9,10,15,22,44]

f = FindElement(T, 10)

s = ListToBST(L)

b = BSTToList(T)

print('This is the ordered tree:', InOrder(T))

time1 = time.time()

print(f)

print("--- %s seconds ---" % (time.time() - time1))

time2 = time.time()

print(s)

print("--- %s seconds ---" % (time.time() - time2))

time3 = time.time()

print(b)

print("--- %s seconds ---" % (time.time() - time3))

time4 = time.time()

PrintDepthOrder(T,4)

print("--- %s seconds ---" % (time.time() - time4))

time5 = time.time()

Draw\_Tree(T, 100, 100, 100, 100)

print("--- %s seconds ---" % (time.time() - time5))

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.

* David A. Davis