Introduction:

The task of this lab is to create basic implementations for a binary search tree and modify the basic implemented/provided functions.

Problem 1:

Proposed solution design and implementation:

Utilized the same methods from lab 1 for plotting the circle and creating a tree. However, the tree and branches methods were modified to the problem. The functions parameters were modified to read the BST. The draw\_tree function determines the directions in which the branches are generated based if the BST has a left or right node. And it automatically plots the circle corresponding to the current node.

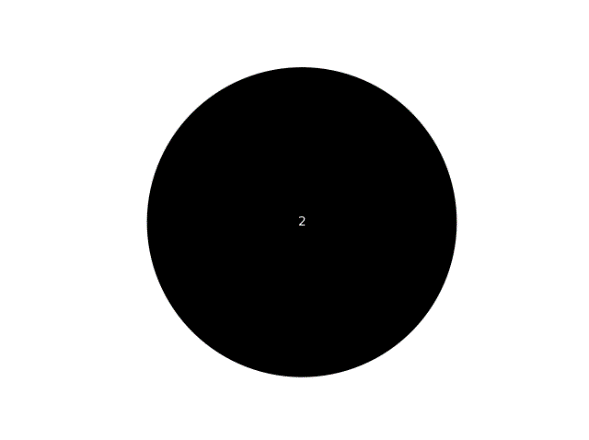
Experimental Results:

Test: A = []

Result: No tree was generated

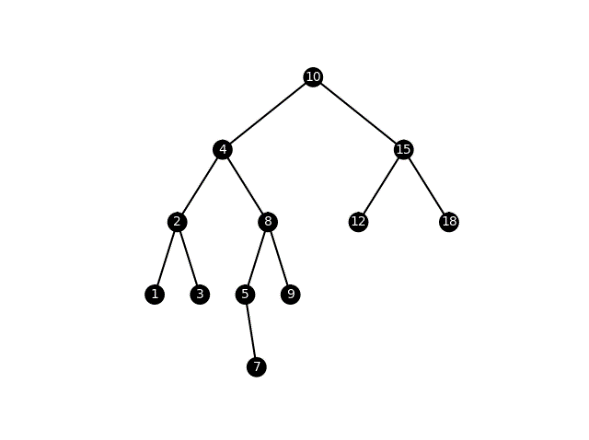
Test: A = [2]

Result: Only a node (circle) was created



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

Result: Tree was draw with draw with correct number of nodes and branches



Problem 2:

Proposed solution design and implementation:

Came up with a similar approach but with a different design to the search function previously provided by the professor. The search function checks for the current node’s value and compares it to the search number. If the current node’s value is greater than the search number, then it searches the left node. Plus, if the current node’s value is less than the search number, then it searches on the right branch of the tree. Else, it returns the current node since it’s the number is neither less than or greater than the search key provided.

Experimental Results

Test: A = [], k = 2

Result: Returns -1, since node is not found

Test: A = [2], k = 2

Result: Returns the node’s value “2”

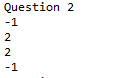
Test: A = [10,4,15,2,8,12,18,1,3,5,9,7], k = 2

Result: Returns the node’s value “2”

Test: A = [10,4,15,2,8,12,18,1,3,5,9,7], k = 100

Result: Returns -1

*Console output:*



Problem 3:

Proposed solution design and implementation:

Came up with an algorithm that always obtains the middle node from the given sorted list. Using the middle index from the list, the function recursively calls the values less than the middle and the values more than the middle respectively. This allows for the method to obtain the middle index of the list from each set of range of numbers to create the new BST.

Experimental Results:

Test: B = []

Result: Returned none

Test: B = [1]

Result: Returned 1 to the BST and then printed its content in console “1”

Test: B = [1,2,3,4,5,6,7]

Result: Returned the list to the BST and then printed its contents “1 2 3 4 5 6 7”

*Console output:*



Problem 4:

Proposed solution design and implementation:

Came up with a simple algorithm that recursively calls to the left most subtree and then shifts the right most branches. As the tree is being traversed the current nodes value is added to an array and returned when done with the tree traversal.

Experimental Results:

Test: T = []

Result: Nothing was returned

Test: T = [2]

Result: Returned the node’s value

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

Result: Returned the trees values in sorted order

*Console output*:



Problem 5:

Proposed solution design and implementation:

Came up with 2 functions, in which one traverses through the list up to the leaf nodes. While the function is traversing the list, it calls the other function to perform the printing. The printLevel function prints the nodes from the current depth level by recursively calling the left subtrees and then the right subtrees

Experimental Results:

Test: T = []

Result: Nothing was returned

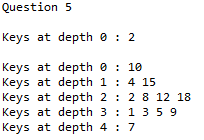
Test: T = [2]

Result: Returned the node’s value at depth 0

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

Result: Returned the BST values from depth 0 to 4

*Console output*:



Appendix:

*# Course: 2302-001*

*# Author: Esteban Retana*

*# Assignment: Create the missing methods to draw,sort and traverse through the tree*

*# Instructor: Olac Fuentes*

*# TA: Anindita Nath*

*# Date of last modification:3/10/19*

*# Purpose: Practice basic binary search tree operations*

import numpy as np

import matplotlib.pyplot as plt

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 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')

*# My own methods*

*# Creates circle figure*

def circle(x,y,rad):

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

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

*# Coordinates*

a = x+rad\*np.sin(t)

b = y+rad\*np.cos(t)

return a,b

*# Create binary tree plot*

def draw\_tree(T,ax,x\_shift,x,y):

if T != None:

*# Create circle for middle node*

a,b = circle(x,y,10)

ax.plot(a,b,color='k')

ax.fill(a,b,'k',alpha=1)

ax.annotate(T.item, xy=(x,y), fontsize=10, color='001', ha="center", va="center")

*# Create left branch*

if T.left != None:

draw\_branch(T.left,ax,x\_shift,x,y)

draw\_tree(T.left,ax,x\_shift/2,x-x\_shift,y-80)

*# Create right branch*

if T.right != None:

draw\_branch(T.right,ax,-x\_shift,x,y)

draw\_tree(T.right,ax,x\_shift/2,x+x\_shift,y-80)

*# Create branch for tree*

def draw\_branch(item,ax,x\_shift,x,y):

q = np.array([[x,y],[x-x\_shift,y-80]])

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

*# Search for Node with given integer element*

def Search(T,k):

while T != None:

*# Check left branch*

if T.item > k:

T = T.left

*# Check right branch*

elif T.item < k:

T = T.right

*# Found node*

else:

return T

return None

*# Convert Sorted Array list to BST*

def SortedToBST(B):

if not B:

return None

*# Obtain middle node*

mid = len(B) // 2

*# Create center node*

root = BST(B[mid])

*# Create subtree less than center*

root.left = SortedToBST(B[:mid])

*# Create subtree more than center*

root.right = SortedToBST(B[mid+1:])

return root

*# Have BST converted to sorted Array list*

def BSTToSorted(T,A):

if T != None:

BSTToSorted(T.left,A)

A += [T.item]

BSTToSorted(T.right,A)

return A

*# Check height of Tree*

def Height(T):

if T == None:

return 0

left = Height(T.left)

right = Height(T.right)

if left > right:

return left + 1

else:

return right + 1

*# Print all nodes to their depth level respectively*

def PrintWithDepth(T):

l = Height(T)

*# Traverse through each depth level*

for i in range(l):

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

PrintLevel(T,i)

print()

*# Print nodes at a depth*

def PrintLevel(T,i):

if T == None:

return

*# Print elements*

if i == 0:

print("%d " %(T.item), end='')

elif i > 0:

PrintLevel(T.left, i-1)

PrintLevel(T.right, i-1)

*# Code to test the functions above*

T = None

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

for a in A:

T = Insert(T,a)

*# InOrder(T)*

*# print()*

*# InOrderD(T,'')*

*# print()*

*# print(SmallestL(T).item)*

*# print(Smallest(T).item)*

*# FindAndPrint(T,40)*

*# FindAndPrint(T,110)*

*# Problem 1*

plt.close("all")

fig, ax = plt.subplots()

draw\_tree(T,ax,100,0,0)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('binarytree.png')

*# Problem 2*

print("Question 2")

print(Search(T,100).item)

*# Problem 3*

print("Question 3")

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

U = SortedToBST(B)

InOrder(U)

print()

*# Problem 4*

print("Question 4")

V = BSTToSorted(T,[])

for i in range(len(V)):

print(V[i],'', end='')

print()

*# Problem 5*

print("Question 5")

PrintWithDepth(T)

Academic Honesty:

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.