Lab 3

Fernando De Santiago

University of Texas at El Paso

Lab 3

The purpose of this lab was to create a display a Binary search tree as well as implementing methods that are meant to be apart of it. The first method was to implement a method to draw the tree. The second method was to create a search method that worked interactively. The third was to create a method to create a method that creates a binary tree from a sorted array. The fourth was to do the reveres and make a sorted array out of a binary search tree. The last and final was to print what numbers can be found at certain depths.

# Method 1

This was made by using parts of the first lab and then adding a string text to it to display the figure. This was probably the hardest for me since I tried to start from scratch by using my lab one as a reference but I couldn’t figure out how to fill in the circle so I decided to start from 0. When experimenting I used multiplication, division and powers in order to make my lines but found out that powers worked the best and then just drew the circles at the end of the lines except the root.

## Method 2

This method was not so hard to create since something similar was used in the code, so I went about and just deconstructed it and recreate it. I decided to go about reconstructing it by checking if the number I was looking for was less than or greater than the number I started with. Depending on which one it is then it’ll move to the left or to the right and check again until either it’s found or not.

### Method 3

For this method I had to figure out the approach of how to implement a way of creating a BST out of an Array. It started with me getting the length and using that as finding the midpoint so that it would be the root of the tree. From there I implemented the left roots by going to the left of the middle and vice versa for the right. In the end I returned the new BST.

#### Method 4

For this method I reversed engineered the third method and decided to have to start at the very left and add recursively to an empty array then getting the root then getting the numbers on the right and so on until I got the Array sorted from smallest to biggest. This was the easiest of the 5 methods to do.

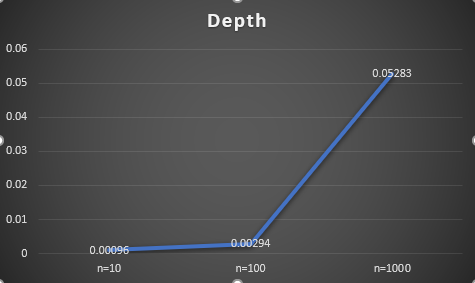
##### **Method 5**

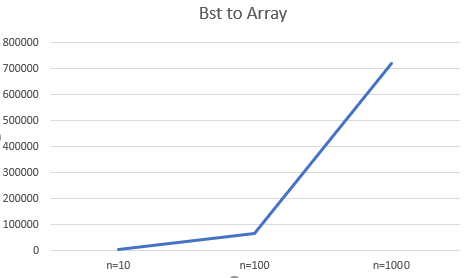
For this method I used 3 different methods to return the keys at the each depth till there is no more depth. The first method that is used as reference prints the following of key at depth is this. The second counts the depth on the left and right and then compares them and uses whichever one is bigger. The last uses recursion to print the item at the certain height and they all work together to make 1 method.

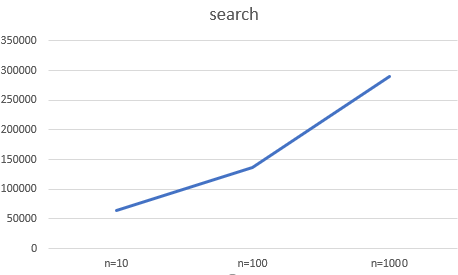
**Conclusion**

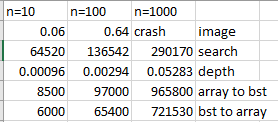
So for this lab I learned the way to traverse a Binary search tree by creating these methods as well as adding counter measures in case they ask for something that shouldn’t work. I ran into some problems when making sure my implementation of method 3 and method 4 were big 0(N) but by making several design changes then was able to meet the requirements.

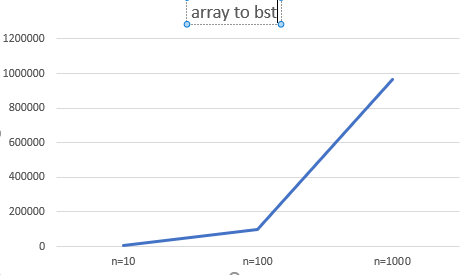
**Runtime and images**

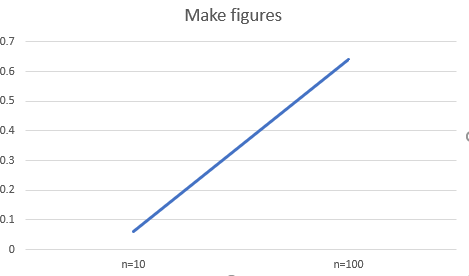


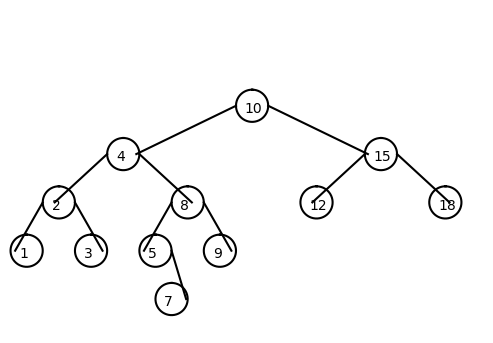


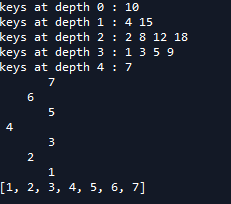












Source Code

# Code to implement a binary search tree

# Programmed by Olac Fuentes

# Last modified February 27, 2019

#CS2302

#Fernando De Santiago

#LAB3

#Olac Fuentes, Anindita Nath and Maliheh Zargaran

#last edited 3/8/19 11:44:00 PM

#Section M/W 10:30-11:50

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

"""

Created on Mon Mar 4 12:27:32 2019

@author: Fernando

"""

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 draw\_line(ax, x1,y1,x2,y2):

n = int(max( abs(x1-x2), abs(y1-y2)) )#the range of the lines

x = np.linspace(x1,x2,n)

y = np.linspace(y1,y2,n)

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

def draw\_circle(ax,center,radius):

x,y = circle(center,radius)

ax.plot((x+radius),y,color=(0,0,0))

def circle(center,rad):

n = int(4\*rad\*math.pi)#radius of each circle

t = np.linspace(0,6.3,n)#creating the circles

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

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

return x,y

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')

def Search(T,key):

if T is not None:#checking tree is not none

if key < T.item:#checking if key is less than item

Search(T.left,key)#moving left

elif key > T.item:#checking if key is greater than item

Search(T.right,key)#moving right

elif key==T.item:#key is equal to item

print(T.item, 'found')

else:

print(key,'not found')

def BSTForArrays(B):

if not B:

return None

mid=(len(B))//2#gets middle elemnt of array

T=BST(B[mid])#puts middle element in the middle of array

T.left=BSTForArrays(B[:mid])#makes T.left item a object in left of the array

T.right=BSTForArrays(B[mid+1:]) #makes T.right item a object in right of the array

return T

def BSTToArrays(T,d):

if T is not None:

BSTToArrays(T.left,d)#goes all the way to the left

d+=[T.item]#adds item to the array d

BSTToArrays(T.right,d)#goes all the way to the right

return d

def Depth(T):

counter=HeightCounter(T)#calls method to get total depth

for i in range(counter):

print('keys at depth',i,':',end=' ')#prints out message and digits at the depth

Tiers(T,i)#gets the numbers at the depth

print()

def HeightCounter(T):

if T is None:

return 0

CountL=HeightCounter(T.left)#gets item on the left

CountR=HeightCounter(T.right)#get item on the right

if CountL>CountR: #comparing which is bigger

return 1+CountL#adding to counter

return 1+CountR#adding to counter

def Tiers(T,Height):

if T is None:

return

if Height==0:#checking if i is 0

print(T.item,end=' ')#returning the item

else:

Tiers(T.left,Height-1)#gets item on left

Tiers(T.right,Height-1)#gets item on the right

def Tree(ax,x,y,width,heightcounter,T):

if T is not None:

draw\_circle(ax,[x,y],4)#draws the circle

ax.text(x+2.2,y-1.8,T.item,size=10)#writes the text

if T.left is not None:

draw\_line(ax,x,y,x-(1.9\*\*heightcounter),y-width)#draws left line

Tree(ax,x-2\*\*heightcounter,y-width,width,heightcounter-1,T.left)#recursively calls next node to the left

if T.right is not None:

draw\_line(ax,x+8,y,x+8+(1.9\*\*heightcounter),y-width)#draws right line

Tree(ax,x+(2\*\*heightcounter),y-width,width,heightcounter-1,T.right)#recursively calls next node to the right

# Code to test the functions above

T = None

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

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

for a in A:

T = Insert(T,a)

InOrder(T)

fig, ax=plt.subplots()

d=HeightCounter(T)

Tree(ax,0,0,12,d,T)

ax.set\_aspect(1.0)

plt.show()

plt.axis('off')

print()

Search(T,4)

print()

InOrderD(T,'')

print()

Depth(T)

T=BSTForArrays(B)

InOrderD(T,'')

fig, ax1=plt.subplots()

d=HeightCounter(T)

DrawTree(ax1,0,0,12,d,T)

ax1.set\_aspect(1.0)

plt.show()

plt.axis('off')

C=[]

BSTToArrays(T,C)

print(C)

#

#print(SmallestL(T).item)

#print(Smallest(T).item)

#

#FindAndPrint(T,40)

#FindAndPrint(T,110)

#Search(T,7)

#n=60

#print('Delete',n,'Case 1, deleted node is a leaf')

#T = Delete(T,n) #Case 1, deleted node is a leaf

#InOrderD(T,'')

#print('####################################')

#

#n=90

#print('Delete',n,'Case 2, deleted node has one child')

#T = Delete(T,n) #Case 2, deleted node has one child

#InOrderD(T,'')

#print('####################################')

#

#n=70

#print('Delete',n,'Case 3, deleted node has two children')

#T = Delete(T,n) #Case 3, deleted node has two children

#InOrderD(T,'')

#

#n=40

#print('Delete',n,'Case 3, deleted node has two children')

#T = Delete(T,n) #Case 3, deleted node has two children

#InOrderD(T,'')

“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.”  
 -Fernando De Santiago