Michael Gonzalez

CS 2302 Data Structures

Olac Fuentes

03/12/2019

Lab 3 – Binary Search Trees

For this lab, we have been tasked with implementing various methods for a Binary Search Tree (BST). 1. Display the binary search tree as a figure, 2. Iterative version of the search operation, 3. Building a balanced binary search tree given a sorted list as input, 4. Extracting the elements in a binary search tree into a sorted list, and 5. Printing the elements in a binary tree ordered by depth. The biggest challenge that I could see with this assignment is that I would have to figure out how to Traverse the data structure by thinking in terms of its left and right nodes in a recursive manner.

My solution for part 1 was to create a method called “DispBST” that used an “ax” a matplotlib.pyplot graph variable, an “x” and “y” variable to store the coordinates of the graph, a “p” array that holds the different coordinates to place the drawings, an “r” variable to store the radius/ length of the shapes and a “T” a BST tree. First the method would take in the x and y variables and use a circle and a drawCircle method to draw a circle at x, y with radius 1 at the top of the graph. Then it will annotate at the x, y coordinate the item stored in the current node, next it will plot lines branching to any other nodes if there are any. Update the p array with the left values and make a recursive call using the left side of the tree, then update p again with the right values and make another recursive call using the right side of the tree.

My solution for part 2 was to create a method called “IterativeSearch” that used a “T” BST tree and a number “k”. First the method goes into a while loop which has the condition “while T is not None:” if T is not None, it goes through a few if/else statements. First if the item in the node is the same as the value of k then return that node for it has been found. If else the item in the node is less than k then we set T to T.right since we know that the value k is greater than the entire left side of the current tree. Else, means that the value of k is less than the value of the node in T so we set T to T.left. If the case T is None happens then that means that the item k is not in this BST and as such, we must return None.

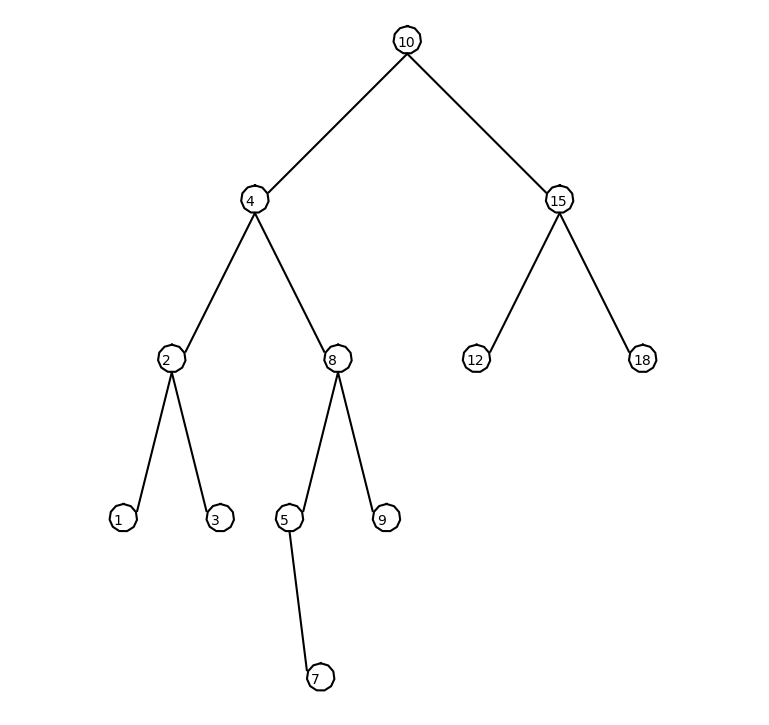
My solution for part 3 was to create a method called “BuildListSorted” that used a sorted list “L”. the method only run if the length of the list was more than 0, then it took the median of the list and made that number into a BST root node and then split the list onto two variables LList and RList which held the left and right side of the list respectively. Then the method made a recursive call to create the nodes T.left and T.right nodes each using their respective side LList and RList, and after it was all said and done it would return the node T creating a BST from a sported list. Based on what we were told in the lab assignment the method’s running time should be O(n) and by use of the master method it turns out to be true. Since the recurrence equation is T(n) = 2T(n/2) + 1 which makes a =2 b=2 and k =0, meaning that 2 > 2^0 == 2>1 so we use case 3 of the master method O(n^logba) = O(n^log22)==O(n^1) = O(n).

My solution for part 4 was a method called “MakeList” which took in a “T” BST variable. The method is simple, first make sure that T is not None, then make a recursive call to the left node, T.left, next append the item in the current node to a list outside the method and the make the recursive call again but this time for the right node ensuring that the method will traverse to the leftmost node, append those items to a list, and proceed to do the same with the right most nodes in order from least to greatest. Based on what we were told in the lab assignment the method’s running time should be O(n) and by use of the master method it turn out to be true. Since the recurrence equation is T(n) = 2T(n/2) + 1 which makes a =2 b=2 and k =0, meaning that 2 > 2^0 == 2>1 so we use case 3 of the master method O(n^logba) = O(n^log22)==O(n^1) = O(n).

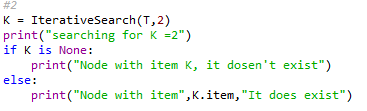
My solution for part 5 was two methods called “printhDepthOrder” and “printGivenLevel” for the first method, printhDepthOrder, it used a “T” the Root of a BST tree, it calculated the depth of this tree by a maxDepth method included in our Class BST.py file. Next it went into a for loop for which it repeated the code from zero to the max depth, i, in this loop there is an if statement if i>0 and it will print “Keys at depth”, i-1,”:” then it will call printGivenLevel by sending it the current root and I and after that it will make a new line for the next keys. The second method, printGivenLevel, takes in a “T” BST root and an “i’ indicating the level of the tree. Then it runs by a set of if statements, first, if the “T” is none then it returns since nothing else can be done, next if the root level is one i ==1 then it prints the roots item without making a new line, else, meaning i is greater than 1 then it makes two recursive calls, one using the T.left and i-1 and the second using T.right and i-1 to assure that both sides of the tree are reached.

Experiments/Results

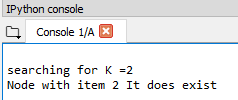
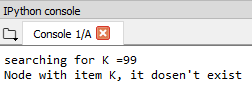
1. Display the binary search tree as a figure

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

2. Iterative version of the search operation.



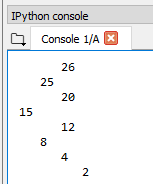
Case 1 K is in Tree Case 2 Kis not in Tree

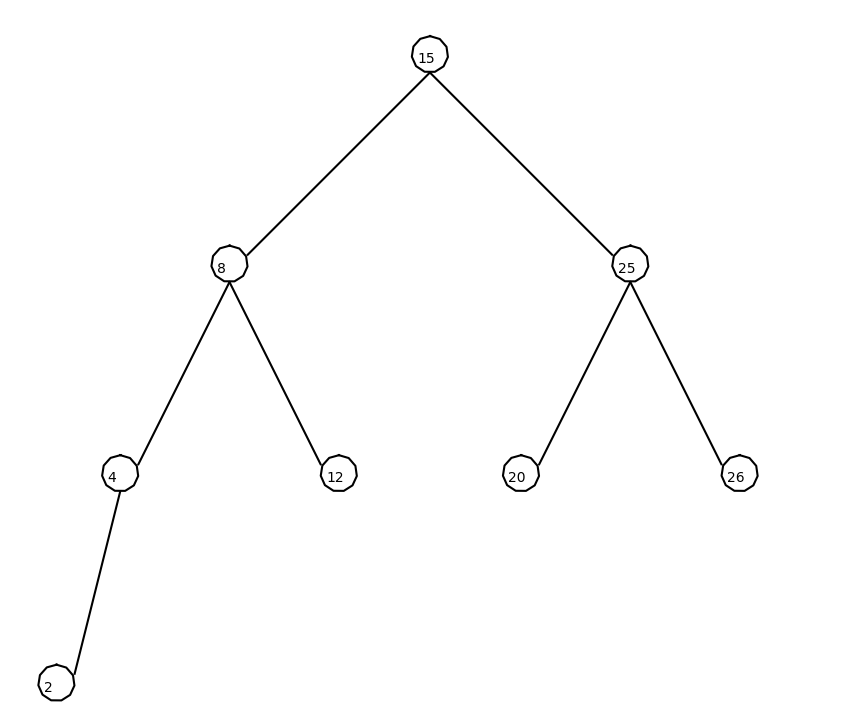
3. Building a balanced binary search tree given a sorted list as input.

BST B = [2,4,8,12,15,20,25,26]

Console

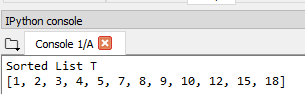


As a BST diagram



4. Extracting the elements in a binary search tree into a sorted list.

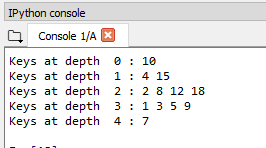
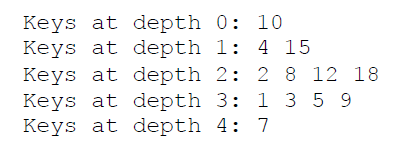
BST T = [10,4,15,2,8,12,18,1,3,5,9,7]



5. Printing the elements in a binary tree ordered by depth.

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

Console Assignment

Conclusion

In conclusion, I learned how to program various methods about BST and how to traverse and modify the items inside the tree in a recursive manner. Aside from the methods, I have now a better appreciation of BST and how efficient and inefficient they can be. I have become more comfortable with coding in python than in lab 2 and I believe that I will be able to learn more from future labs to come.

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.

– Michael Gonzalez

Appendix – code

#Author: Michael Gonzalez

#Course: CS 2302 Data Structures

#Lab 3

#TA: Anindita Nath

#Purpose:the purpose of this lab is to modify the code given in class and implement various methods.

#1. Display the binary search tree as a figure

#2. Iterative version of the search operation.

#3. Building a balanced binary search tree given a sorted list as input.

#4. Extracting the elements in a binary search tree into a sorted list.

#5. Printing the elements in a binary tree ordered by depth.

# Last modified February 27, 2019

import matplotlib.pyplot as plt

import numpy as np

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 maxDepth(T):

if T is None:

return 0 ;

else :

# Compute the depth of each subtree

lDepth = maxDepth(T.left)

rDepth = maxDepth(T.right)

# Use the larger one

if (lDepth > rDepth):

return lDepth+1

else:

return rDepth+1

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 DrawTree(T):

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 PrintRevserse(T):

if T is not None:

PrintRevserse(T.right)

print(T.item,end = ' ')

PrintRevserse(T.left)

def CountNodes(T):

Count = 0

if T is not None:

Count += 1

else:

return 0

return Count + CountNodes(T.right) + CountNodes(T.left)

def FindAndPrint(T,k):

f = Find(T,k)

if f is not None:

print(f.item,'found')

else:

print(k,'not found')

def SumTree(T):

if T is None:

return 0

return T.item + SumTree(T.left) + SumTree(T.right)

#that receives a reference to the root of a binary search

#tree T and an item k and returns the depth at which k is

# stored in the tree or -1 if k is not in the tree. For example,

#if k is stored in the root, the function should return 0, if k is stored

# in one of the root's children, it should return 1, and so on.

def FindDepth(T,k):

count = 0

while T is not None:

if T.item == k:

return count

elif T.item<k:

T=T.right

count += 1

else:

T=T.left

count += 1

return -1

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

# return 0

# if T.item<k:

# return 1 + FindDepth(T.right,k)

# if T.left.item>k:

# return 1 + FindDepth(T.left,k)

#

#That receives a reference to the root of a binary search tree T

#and an integer d and returns the sum of items in the tree that are stored in

# nodes with depth d. Hint: if the tree is empty, the sum is 0, if d is 0, the

# sum is the root's item, otherwise the sum is the sum of the items with

#depth d-1 in the left subtree plus the sum of the items with depth d-1 in the

#right subtree.

def SumAtDepth(T,d):

if T is None:

return 0

if d is 0:

return T.item

return T.item + SumAtDepth(T.left,d-1) + SumAtDepth(T.right,d-1)

#############################################################################

# LAB 3 Start

########################################################

#1. Display the binary search tree as a figure

def DispBST(ax,x,y,p,r,T):

if T is not None:

#plot circle that is stored in "p" array

drawCircle(x,y,1)

#plot the item stored at the current node

ax.annotate(T.item, (x-.7,y-.5))

y=y-1

#plot line that is stored in "p" array

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

#create variables to store new coordinate values for new branches

xleft = x-r

xrigth = x+r

ydown = y-10

#update "p" array and call method with the left tree nodes

p = np.array([[x,y],[xleft,ydown]])

DispBST(ax,xleft-1,ydown-.5,p,r\*.5,T.left)

#update "p" array and call method with the right tree nodes

p = np.array([[x,y],[xrigth,ydown]])

DispBST(ax,xrigth+1,ydown-.5,p,r\*.5,T.right)

#method to plot a circle

def drawCircle(x,y,r):

center = [x,y]

Cx,Cy = circle(center,r)

ax.plot(Cx,Cy,color='k')

ax.fill(Cx, Cy,'w')

#method to create a circle

def circle(center,rad):

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

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

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

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

return x,y

########################################################

#2. Iterative version of the search operation.

def IterativeSearch(T,k):

while T is not None:

if T.item == k:

return T

elif T.item<k:

T=T.right

else:

T=T.left

return None

########################################################

#3. Building a balanced binary search tree given a sorted list as input.

def BuildTreeSorted(L):

if len(L) >= 1:

median = L[len(L)//2]

T = BST(median)

LList = L[:((len(L)//2))]

RList = L[((len(L)//2)+1):]

T.left = BuildTreeSorted(LList)

T.right = BuildTreeSorted(RList)

return T

########################################################

#4. Extracting the elements in a binary search tree into a sorted list.

def MakeList(T):

if T is not None:

MakeList(T.left)

List.append(T.item)

MakeList(T.right)

########################################################

#5. Printing the elements in a binary tree ordered by depth.

def printDepthOrder(root):

h = maxDepth(T)

for i in range(h+1):

if i>0:

print("Keys at depth ",i-1,":",end=" ")

printGivenLevel(root, i)

print()

# Print nodes at a given level \*/

def printGivenLevel(root,level):

if root is None:

return

if level == 1:

print(root.item,end=" ")

elif (level > 1):

printGivenLevel(root.left, level-1);

printGivenLevel(root.right, level-1);

########################################################

# Code to test the functions above

T = None

List =[]

A = [12,4,20,2,8,15,25]

B = [2,4,8,12,15,20,25,26]

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

for a in C:

T = Insert(T,a)

#InOrder(T)

print()

InOrderD(T,'')

print()

#PrintRevserse(T)

print()

#1

plt.close("all")

fig, ax = plt.subplots()

x = 0

y = 0

p = np.array([[x,y],[0,0]])

#DispBST(ax,0,0,p,10,T)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('BST.png')

#2

K = IterativeSearch(T,99)

print("searching for K =99")

if K is None:

print("Node with item K, it dosen't exist")

else:

print("Node with item",K.item,"It does exist")

print()

#3

ST = BuildTreeSorted(B)

DispBST(ax,0,0,p,10,ST)

InOrderD(ST,'')

print()

#4

MakeList(T)

print("Sorted List T")

print(List)

#5

printDepthOrder(T)

#