CS 2302 Data Structures

Spring 2019

MW 10:30-11:50 in CCSB 1.0202

LAB # 3

Claudio Angel Garcia

Instructor: Olac Fuentes

TA: Anindita Nath, Maliheh Zargaran

IA: Eduardo Lara

Peer leader: Erick Macik

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.

* Claudio Garcia

Introduction

The problem that we have in this lab is that we need to draw a binary search tree using Matplotlib library given a binary tree. Then we are asked to do a method that can search for an element in the binary tree, but this method has to be a iterative version of the search method meaning that we have to go through all the tree until we find the element the user is searching for. For the third problem, we need to build a binary tree given an already sorted list, this makes it much easier to do because we can just read the list and send half to the right and to the left. Then, for the fourth problem we need to do the opposite of the previous instruction, we are given a binary tree and we need to extract the elements in this tree and append them to an empty list.

For the fifth problem, we need to print the items in the tree and the depth in which they are.

Solution

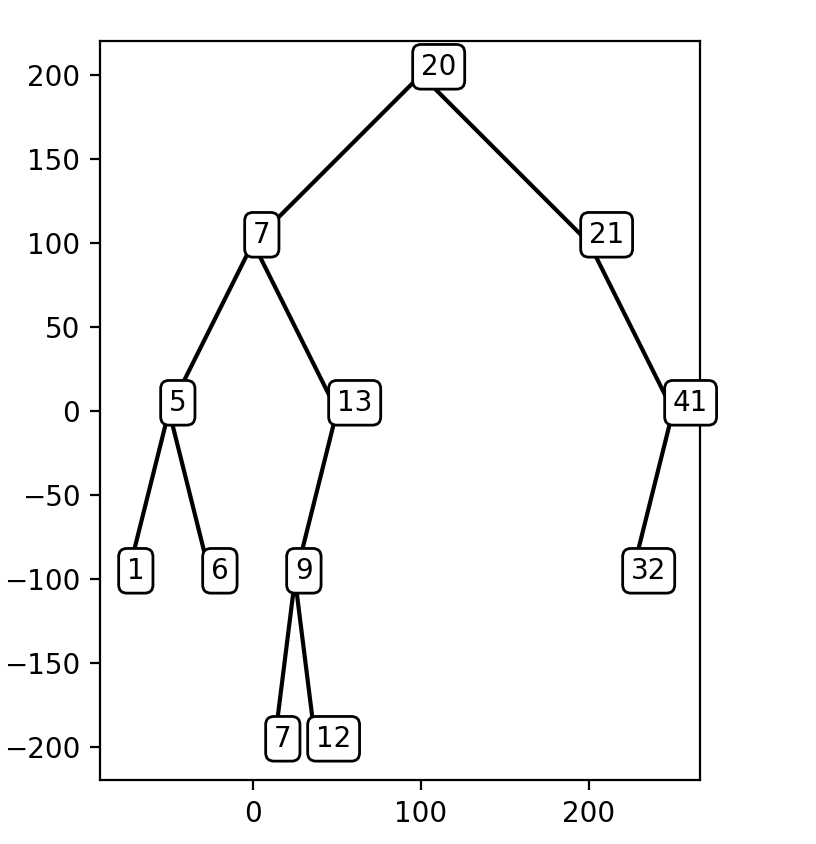
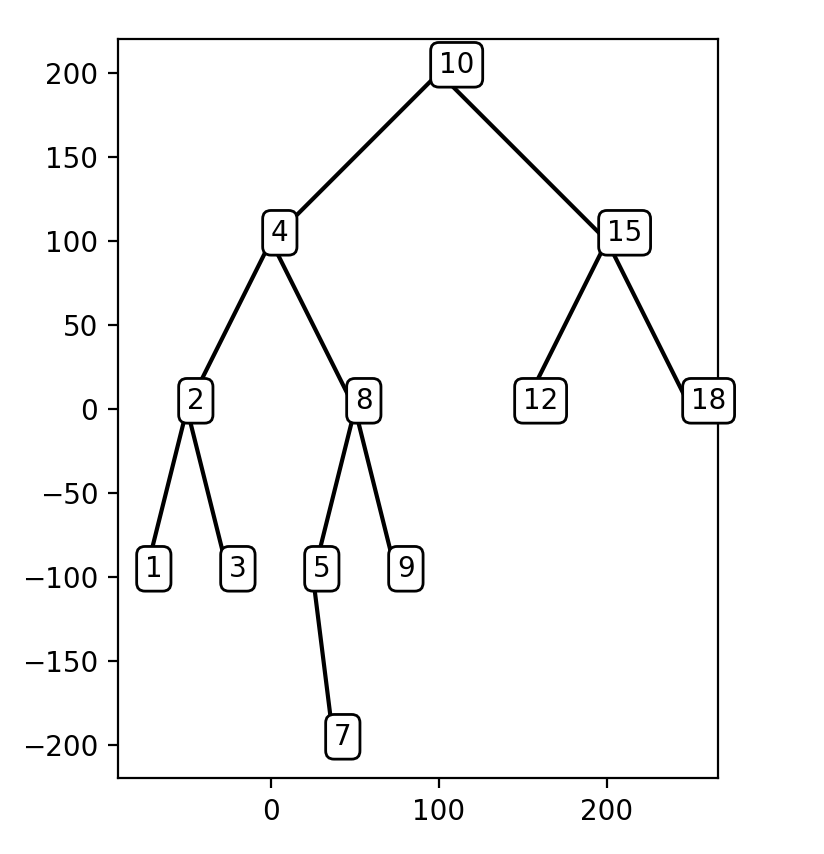
For the first problem what I did was to use a simple method to draw lines (as in the first lab) and start form there. Then I used a function in Matplotlib that lets us circle an element and give shape. After that I used the same approximation of Lab 1: draw\_tree, the difference is that I did two if statements for each part of the binary tree, the right and the left.

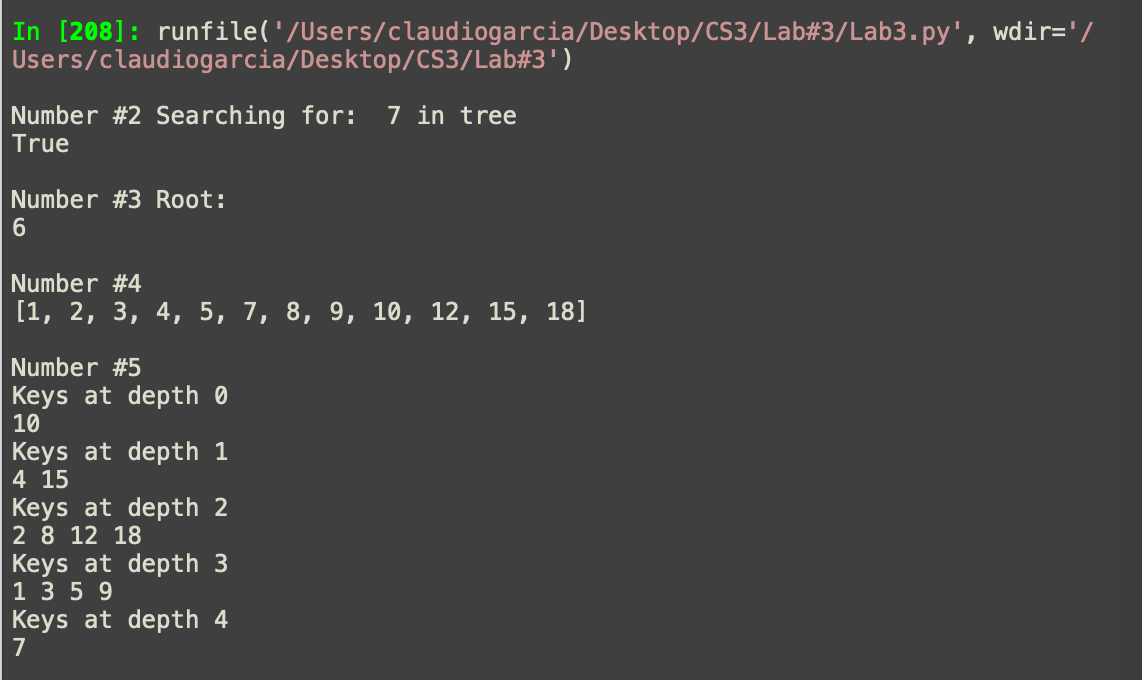
The second problem was easy because we just needed to go through all the tree searching for a given key using a while loop to cover the entire tree.

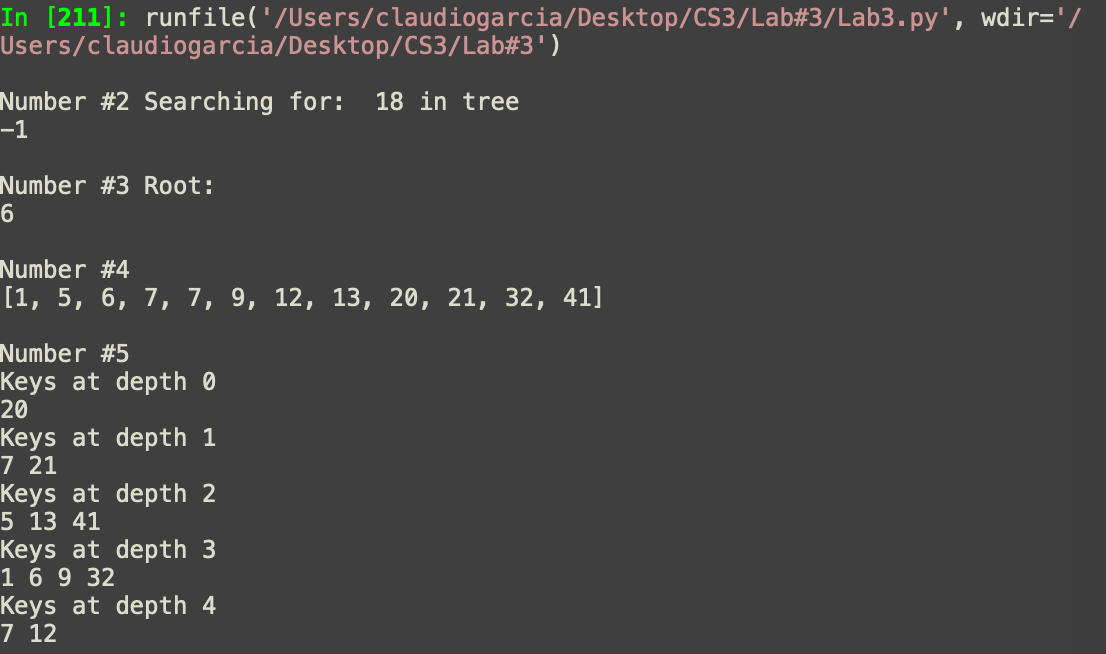
For the third problem, we just needed to insert the values of a sorted list into a binary tree, this was less complicated because of the fact we just used a sorted list an at the time of inserting the values into the tree I just created a BST variable called root that equals to the middle of the sorted list and from there use .right and .left to insert half of the list into the left and the other into the right, thus ending with a binary tree.

The fourth problem was to do the opposite of the third problem. I started with an if statement that return an empty list if there is nothing in the given tree, else, we insert .right and .left values into an empty list.

In the fifth instruction we needed three methods to be able to accomplish the task. First, the height method that returns how tall the tree is, this way we know now how large is the three and use that in a for loop. After that the ElementAtDepth method prints the values at a given height depending of the for loop in the printDepth method that basically just do a for loop calling ElementAtDepth and this way it prints the depth and the values in that depth.

Experimentation





Conclusion

With this lab I practiced more how to plot things in python and now I can draw a binary search tree with code given a tree. I also practiced and learned more about binary search trees and about the basic functions made differently.

Appendix

#!/usr/bin/env python3

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

"""

Created on Wed Feb 27 11:09:14 2019

@author: claudiogarcia

CS 2302 Data Structures Spring 2019

MW 10:30-11:50 in CCSB 1.0202

Instructor: Olac Fuentes

TAs: Anindita Nath, Maliheh Zargaran. IA: Eduardo Lara

Peer Leader: Erick Macik

Lab 3

Claudio Garcia 80628314

The purpose of this program is to recieve an binary search tree and draw it as if it was tree.

Then, to search for a given key and return true or false depending on if the key is in the tree.

Another function is to recieve a sorted list and append the values into a tree and return the root

The fourth function is the opposite, we recieve a tree and we extract the values to append them in to a list

The number five prints the values of the tree and the depth in which they are.

"""

import numpy as np

import matplotlib.pyplot as plt

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 InOrder(T):

if T is None:

return

InOrder(T.left)

print(T.item)

InOrder(T.right)

def Smallest(T):

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 SearchK(T, k):

if k > T.item:

if T.right is not None:

if T.item == k:

return T

else:

SearchK(T.right,k)

if k < T.item:

if T.left is not None:

if T.item == k:

return T

else:

SearchK(T.left,k)

def sum(T):

tempR = T.right

tempL = T.left

s = T.item

s2 = 0

while tempR is not None:

s += tempR.item

tempR = tempR.right

while tempL is not None:

s2 += tempL.item

tempL = tempL.left

return s+s2

def SumTree(T):

if T is None:

return 0

else:

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

def FindDepth(T,k):

if T is None:

return 0

else:

if k != T.item:

return -1

elif k == T.item:

return 1

FindDepth(T.right,k), FindDepth(T.left,k)

T = None

A = [20, 7, 21, 5, 1, 13, 41, 6, 9, 12, 32, 7]

for a in A:

T = Insert(T,a)

######################### LAB 3 ################################

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')

def draw\_tree(T, x, y, xChange, yChange):

if T is not None:

#Circulating T.item usign bbox function in plt

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

#Using same method of Lab 1 to draw trees

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

q=np.array([[x-xChange, y], [x, y+yChange]])

draw\_line(ax, 1, q,.9)

draw\_tree(T.left, x-xChange, y-yChange, xChange/2, yChange)

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

q = np.array([[x, y+yChange], [x+xChange, y]])

draw\_line(ax,1,q,.9)

draw\_tree(T.right, x+xChange, y-yChange, xChange/2, yChange)

def iterativeSearch(T, k):

if T is None:

return -1 #The key is not in the tree

if T.item == k:#Key at the root return true

return True

if T is not None:

while T is not None: #iterating through the binary tree with a while loop until it finds the key

if T.item < k:

T = T.right

elif T.item > k:

T = T.left

else: #else returning true when found

return True

return -1

def sortedListToBST(L):

if len(L) == 0:

return None

mid = len(L)//2

root = BST(L[mid]) #root is the mid of the sorted list

#because is sorted we just save the call in root.left and root.right

root.left = sortedListToBST(L[:mid])

root.right = sortedListToBST(L[mid+1:])

return root #return the reference

def extract(T):

if T is None:

return [] #return an empty list if T has no items

else:

return extract(T.left)+[T.item]+extract(T.right) #inserting the items in a list

def printDepth(T, h):

for i in range(h+1): #for loop to print the depth and the keys on that depth

#depending on the height of T

print('Keys at depth', i)

ElementAtDepth(T, i)

print()

def height(T):#function to get the height of a given tree

if T is None:

return -1

else:

a = 1+height(T.left)

b = 1+height(T.right)

if a>=b:

return a

if b>=a:

return b

def ElementAtDepth(T, n): #function to print T.item to be used in printDepth method

if T is None:

return

if n==0:

print(T.item, end=' ')

else:

ElementAtDepth(T.left, n-1) #going to the left of the tree and printing when n=0

ElementAtDepth(T.right, n-1)

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

key = 18

print('\nNumber #2', 'Searching for: ', key, 'in tree ')

print(iterativeSearch(T, key))

print('\nNumber #3', 'Root:')

L = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

BSTList = sortedListToBST(L)

print(BSTList.item)

print('\nNumber #4')

s = extract(T)

print(s)

print('\nNumber #5')

printDepth(T, height(T))