Ericka Najera

CS2302

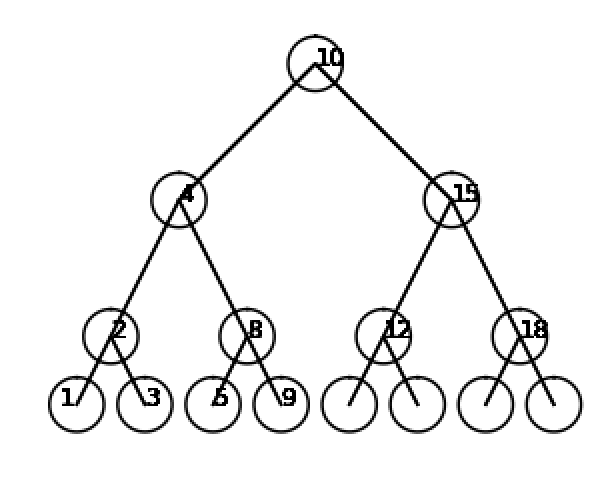
Dr. Fuentes

11 March 2019

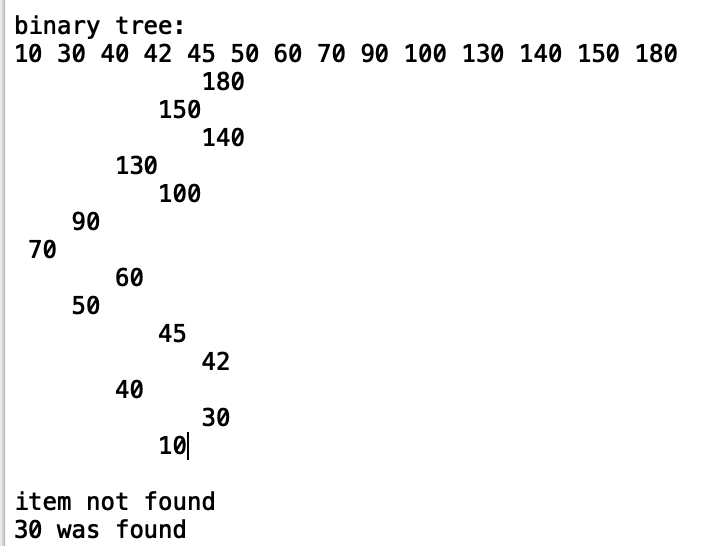
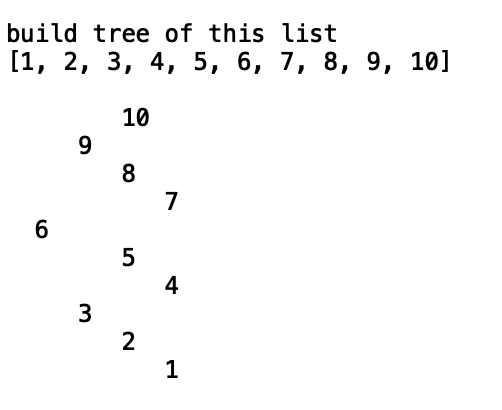
Lab 3

Lab 3 was solely about practicing and understanding a binary tree. Where we had to draw a binary tree with the number inside the circles of each node. Then we had to search a tree and return if the item was found or not. Then we had to build a balanced tree of a sorted list. Extract the binary tree items and make a sorted list. The last module is to print the depths of a binary tree along with which depth it is at.

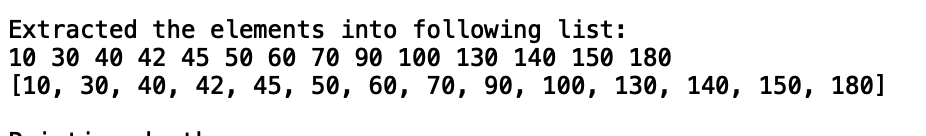
As for the module of drawing a tree, since the lab1 where we had to draw a tree I couldn’t do it and in this lab i finally built. I thought about sending a list with the initial points which were the root which were at (0,0), ( half of the size it will be) and (size,0). From there I made the module call three different x values and two y values since y only changed once and not three like x. The left point x value would be the recent point minus the change in x, then that middle point would be the either the left or right point depending on which call it is at. The right point x value will increase by change in x. The new y will decrease by change of y. After the recursive calls I draw a circle by sending that middle point. Inside those circles I tried to print the numbers but I realized you can’t just print them, you have to plot the text. I used three lists one for the x coordinates, one for the y coordinates and one for the values that need to be printed.

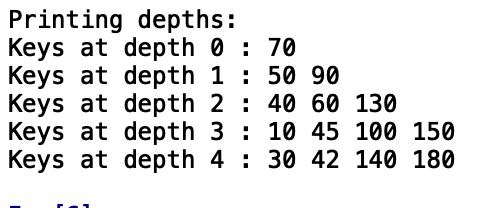


The iterative search module I used a temporary variable to save the root. Since it is iterative, I thought about just making a while loop and checking if whether to go left or right. So, I did a while loop to run while the root is not empty. Inside the while loop I did three if statements to check whether the key is less or more than that current key. In the first case if that root is the number then it returns the item. If it is less than the key, then it goes to the left subtree or if it the right subtree then it goes to the right subtree. After the while loop concludes and the item is not returned then it returns -1. At first, I printed the message and did not return anything but I decided to change it to return the value or -1 like a search method should do.

In order to build a balanced tree of a sorted list, I thought about how a balanced tree should look like. Since it is sorted if you start plugging in values from the begging of the list it will look like just a straight line instead of having equals right and left subtrees. I realized that you had to keep plugging in at the middle and keep choosing the middle after recursion. So, I started by creating my base case and checking if the length of the list was not zero. If it is there, we return an none I thought about formatting it the way the printing inorder module works. From there I save the left side of the recursive call calling the 0 to the middle of the list. After that since the middle will be the root, I create a new node with that value of the list. From there I connect the left of the recursive calls to the left subtree of the root. Then do the same procedure but for the right side. After the tree is recursively formed, I will return the root of the tree.

The module to extract the values of a tree into a sorted list. The printing inorder is the way to get the tree values in order, so I just had to append those values to a list. At the beginning I wanted to create a new list at the beginning but since I made it a recursive call the list will be empty every time. To avoid any problem I made a new list part of the parameters and just sent that new empty list when I called the module. For my base case I checked if the tree itself was empty, if in that case it was then it would return none. As for the rest I checked if that root had a left side. If it had then I would call the method again and call the left side. In the middle of my left and right I would append the root item to the list; this makes it append in order. As for the right side I also check if there is one side, if so I send the right side to the recursive call. At the very end I will return the empty list which will be filled with the tree in order.



In order to print the depth in order I thought about putting the values in a queue and pop them out once that level was completed and print them. I couldn’t get to figure out how to start a queue in python so I tried to do it in a list. I researched to see if you could use a list to save the tree like a temporary variable and I found out that you can since it stores the address not the values. So I new I had to create two lists in the a way like a queue where I would leave the items in the level in that list and append the remaining levels the other list and recursively it would be like just sending that cut of list. I did I a while loop to run while the first list was not empty or known as the tree. Then I did a for loop as well to traverse the tree. Once I hit the root, I would print it, and check if there was a right or left side. If there was, I would append that remaining subtree to the second list. At the end of the for loop I would make the first list equal to the second which would eventually the length of the first will become shorter and shorter causing the while loop to terminate.

Running times:

Iterative Search: O(log n), since the while loop goes either to the left or right cutting n in half each time.

Build a list: T(n) = 2T(n/2)+1 == O(n)

Extract elements: T(n) = 2T(n/2)+1 = =O(n)

Print depths: O( n^2 ) because of the while loop and for loop.

﻿# Ericka Najera Lab 3 MW 10:30-11:50

#Professor Fuentes CS2302

#Lab 3 based on the implementation of binary trees

#class provided by professor

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')

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

#search module to search an item return the item or -1 if not found

def search(T,k):

#save the tree into a temperary node

temp = T

#if the tree is empty return it is empty

if T is None:

return None

#while loop to traverse tree

while temp is not None:

#if the item is found return the item

if(temp.item ==k):

return temp.item

#if not then check that current item to see whether to go right or left

if k< temp.item:

temp = temp.left

else:

temp = temp.right

return -1

#builds a tree of a sorted list and returns list

def buildtree(L):

#base case:if the tree is empty the return a none

if len(L) ==0:

return None

#else build the left side of the tree of the first middle of the array

left = buildtree(L[:len(L)//2])

#the middle becomes the middle item

T = BST(L[len(L)//2])

#the left side connects to the root

T.left = left

#repeasts for the right

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

T.right = right

return T

#extract the items of a tree into a list

def extract(T, newlist):

if T == None:

return

#if there is a left item recursive call the left

if T.left is not None:

extract(T.left,newlist);

#append the root to the list in the middle to make it in order

newlist.append(T.item)

#call the right side

if T.right is not None:

extract(T.right,newlist);

return newlist;

#prints depths of the tree together

def printdepths(T):

#save the root into a list

#counter to save the depth it is at

l = [T]

counter =0

#traverse the list while it is not empty

while l:

#create another list

remaining = list()

print('Keys at depth',counter,':',end = ' ')

#travese the items of the list

for n in l:

#print that item of the list

print(n.item,end = ' ')

#add the left and right to the remaining list

if n.left:

remaining.append(n.left)

if n.right:

remaining.append(n.right)

counter+=1

print()

l = remaining

#draws a binary tree

def tree(ax,n,w,p,size,height,x1,x2,x3,oldy,newY):

if n>0:

#decreases the length of x and y axis

x = size\*.5

y =(2\*size)\*(1/height)

#plots the lines

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

#creates a new arr to send the other values

p = np.array([[x1,newY],[x2,oldy],[x3,newY]])

#the values of each point decrease and increasing depending on y and x and left and right

tree(ax,n-1,w,p,x,height,x1-x,x1,x1+x,newY,newY-y)

tree(ax,n-1,w,p,x,height,x3-x,x3,x3+x,newY,newY-y)

#draw a circle in the middle point

draw\_circles(ax,1,[x2,oldy],(80),w)

#method to get the center and radius of the circles

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

def draw\_circles(ax,n,center,radius,w):

if n>0:

#in each each circle the numbers of the tree will be plotted

#xs is the x coordinates

xs=[400,0,800,-200,200,600,1000,-350,-100,100,300]

#ys is the y coordinates

ys=[400,0,0,-400,-400,-400,-400,-600,-600,-600,-600]

#t is the values to be printed

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

#keeps track of which value in t to be printed

counter = 0

#for loop to run in buth xs and ys

for x, y, in zip(xs,ys):

#plot the text at the x and y coordinate and value of counter in t

plt.text(x,y,str(t[counter]),color='k', fontsize=12)

counter+=1

a,b = circle(center,radius)

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

draw\_circles(ax,n-1,center,radius\*w,w)

T = None

A = [70, 50, 90, 130, 150, 40, 10, 30, 100, 180, 45, 60, 140, 42]

for a in A:

T = Insert(T,a)

size = 800

height =2

xs=[10]

ys=[2]

p = np.array([[0,0],[size/2,size/2],[size,0]])

plt.close("all")

fig, ax = plt.subplots()

tree(ax,4,.2,p,size/2,height,0,400,800,400,0)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

print('binary tree:')

InOrder(T)

print()

InOrderD(T,'')

print()

if(search(T,200)==-1):

print('item not found')

else:

print(search(T,200),'was found')

if(search(T,30)==-1):

print('item not found')

else:

print(search(T,30),'was found')

print()

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

print('build tree of this list')

print(L)

print()

N= None

N =buildtree(L)

InOrderD(N,' ')

print()

newlist=[]

print('Extracted the elements into following list:')

InOrder(T)

print()

m = extract(T,newlist)

print(m)

print()

print('Printing depths:')

printdepths(T)

I Ericka Najera certify that this project is entirely my own work. I wrote, debugged, and teste 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.

