**CS2302 - Data Structures**

**Spring 2019**

**Lab Report 3**

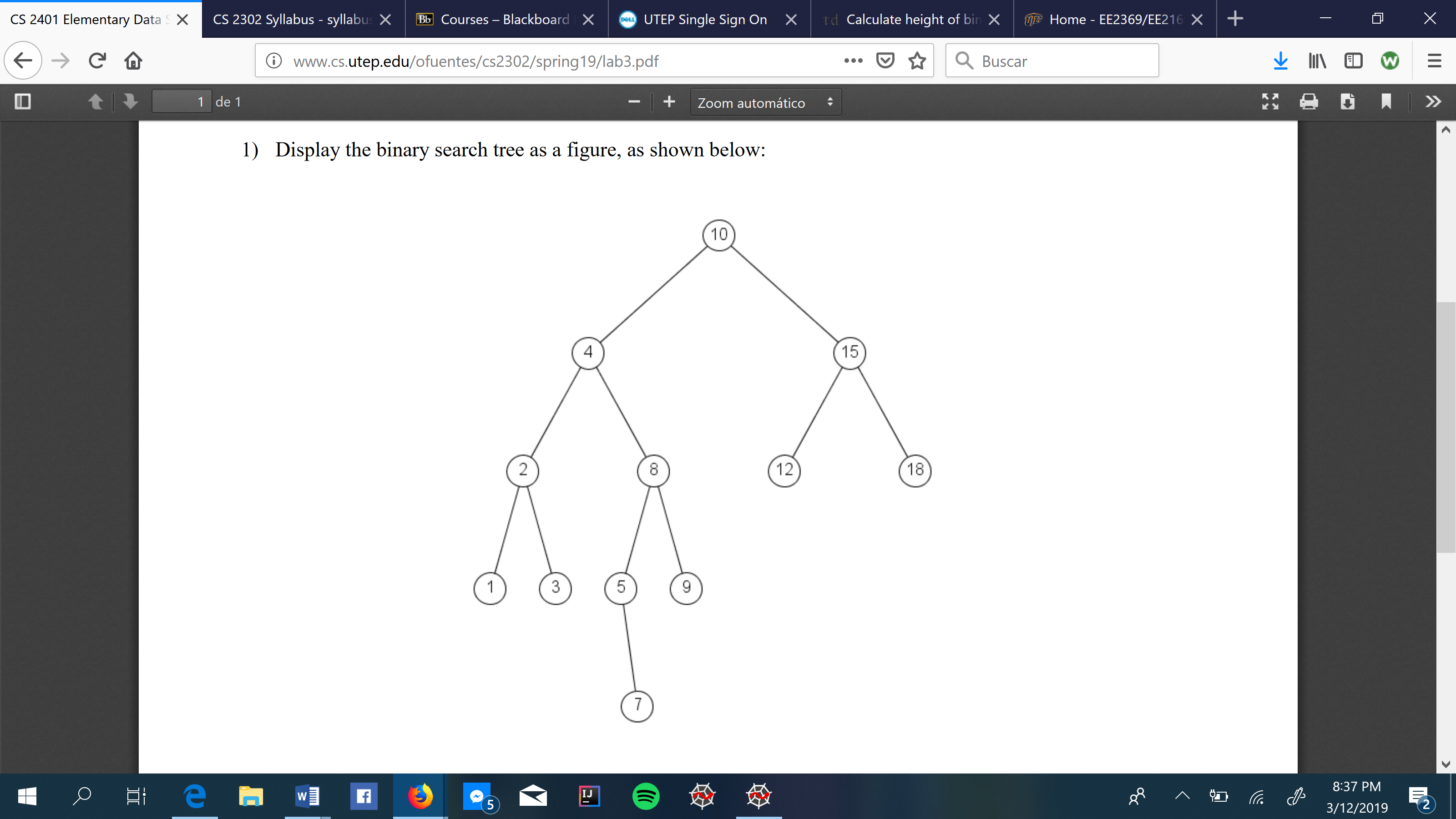
March 12, 2019

Sebastian Gomez

**Introduction:**

Given the binary search tree operations, we were asked to modify and add code to do 5 new operations performed on the binary search tree.

**Implementation:**



1. The first task was to draw the binary search tree using the code from lab 1 to draw trees. In order to do so, there is a function in matplot library, where you can draw the circle more easily, which is plt.Circle. Within that function, as parameters it expected x and y-coordinates, the size of it, the color, and to fill it, which for this case was False. The x and y-coordinates will match the ones used to draw the tree, which changed recursively, and then after drawing the tree with the circles, inside them we would put the data of each tree item to match how a binary search tree should look like. To do so you called plt.text on which you put the coordinates for the text, which would also the match the coordinates for drawing the tree, then the text displayed, which was the item of the tree, and the font size. Unfortunately, I was not able to do such operation.
2. The second operation was to code the search operation iteratively, meaning going through every item in the binary search tree. To do so, I used a while loop to go through every element checking and comparing to the wanted element(searchee), if the element wanted was smaller than the item of the tree, the cursor will now point to the next left element of the tree and then do the comparison again, if it was greater than it would go to the next right element of the tree and then do the comparison again. The while loop could find an end in two possible ways, the first one, if the wanted element was in the binary search tree, it would return true, and if it was not, it would through the whole tree, and then it would print that the element was not found and it would return false.
3. For operation number 3, we were asked to build a binary search tree from a given list, but we could not use the insertion operation. The insertion operation was useless because the given list was going to be sorted, so it would have made a really long and unusual tree going only to the right, so insertion needed to be in another way. The advantage for this type of operation is that we were using native python lists, so splitting the list was going to most viable way, so I made the center of the list the root, and then doing two recursive calls, one for the left side and one for the right side, I inserted the elements into the binary search tree. The elements passed to the left side were from the beginning of the list to the middle element and the elements that went to the right side were the elements after the middle till the end of the list. Then at last you would just return the newly created binary search tree. To prove it was correct, I called the given method InOrder(T) to check the order.
4. For operation number 4 we were asked to the opposite from number 3. We were asked to build a list from a binary search tree. At first, I thought of it to be better by using a while loop and going the smallest element and putting it as the first element and then just go to the right of the tree and keep putting each one of them into the list, but then by using recursion, I noticed that it was faster and fewer lines of code. If the tree was empty, I returned an empty list, but if not, I would pass the element of the left first obviously because they are the smaller ones, and then the right elements and to add them to the list, it first would go to the smallest element and the it would add one by one and the same with the right side. To check if the list was sorted, I printed the elements of the list using a for loop to the extent of the size of the list minus one, because indexes from a list start from zero.
5. For the last operation, which printed the elements at all depths, I did two methods. The first method was doing the method of knowing the elements at certain depth and gave as parameters the tree and the depth wanted, it was pretty similar to other codes we have done but the difference is that this one asked to print at all depths, so for that I did another method to know the height of the tree and then used a for loop going from zero to the height of the tree to give as parameter depth wanted, the i, so it would print at all levels. For the first method I just made recursion call to the left and to the right and subtracted one to the depth every time it did its recursion until it reached zero in order to print. For the height method, I just used the max function with helped to know the bigger side and used that as the height to use it on the for loop.

**Conclusions:**

I learned a lot from binary search trees with this lab by doing various methods of different purposes. Unfortunately, as it did in lab 1, the drawing operation I could not perform it and I hope I just keep getting better at coding and do better on the upcoming labs.

**I, Sebastian Gomez, certify that this project is entirely my own work, I wrote, debugged, and tested the code being presented, performed 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.**

**Appendix:**

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

"""

Created on Wed March 6 19:26:29 2019

@author: Sebastian Gomez

Course: Data Structure 2302

Assignment: Lab 3

Instructor: Olac Fuentes

T.A: Anindita Nath and Maliheh Zargaran

Purpose: Binary search tree operations

"""

import numpy as np

import matplotlib.pyplot as plt

def draw\_BStree(ax, n, deltaX, deltaY, x, y, Tr):

if n>0:

#changing the center x to right and left. y changes always downwards

left = np.array([[x, y], [x - deltaX, y - deltaY]])

right = np.array([[x, y], [x + deltaX, y - deltaY]])

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

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

np.array(plt.Circle((x,y),20,color = 'k',fill=False))

plt.text(x,y,Tr.item, fontdict = None, withdash = False, fontsize = 25)

if Tr is not None:

draw\_BStree(ax, n-1, deltaX / 2, deltaY, x - deltaX, y - deltaY,Tr.left)#Draws the left side

draw\_BStree(ax, n-1, deltaX / 2, deltaY, x + deltaX, y - deltaY,Tr.right)#Draws the right side

class BST(object):

# Constructor

def \_\_init\_\_(self, item, left=None, right=None):

self.item = item

self.left = left

self.right = right

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 PrintAtDepth(Tr,d):

if Tr is None:

return Tr

if d == 0:

print(Tr.item, end = ' ')

else:

PrintAtDepth(Tr.left,d-1)

PrintAtDepth(Tr.right,d-1)

def BSTToList(Tr):

sort = []

if Tr is None:

return sort

else:

sort = BSTToList(Tr.left)+[Tr.item] + BSTToList(Tr.right)

return sort

def Search(Tr,searchee):

if Tr is None:

return Tr

else:

while Tr is not None:

if searchee == Tr.item:

return True

elif searchee < Tr.item:

Tr = Tr.left

elif searchee > Tr.item:

Tr = Tr.right

print('Element not found')

return False

def ListToBST(B):

if len(B) == 0:

return None

median = len(B)//2

Tree = BST(B[median])

Tree.left = ListToBST(B[:median])

Tree.right = ListToBST(B[median+1:])

return Tree

def Height(Tr):

if Tr is None:

return 0

return max(Height(Tr.left), Height(Tr.right)) + 1#using max to know which side is longer

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

Tr = ListToBST(B)

plt.close("all")

fig, ax = plt.subplots()

ax.axis('on')

draw\_BStree(ax, 3, 50, 50, 0, 0,Tr)

ax.set\_aspect(1.0)

plt.show()

fig.savefig('BST.png')

wantedElement = int(input('What element would you like to search?'))

print('Element found: ',Search(Tr, wantedElement))

InOrder(Tr)

print()

sortedList = BSTToList(Tr)

h = Height(Tr)

for i in range(h):

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

PrintAtDepth(Tr,i)

print()

print()

i =0

while i <= len(sortedList) - 1:

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

i +=1

print()

print()

print()

InOrderD(Tr, ' ')