Erick Perchez

80582912

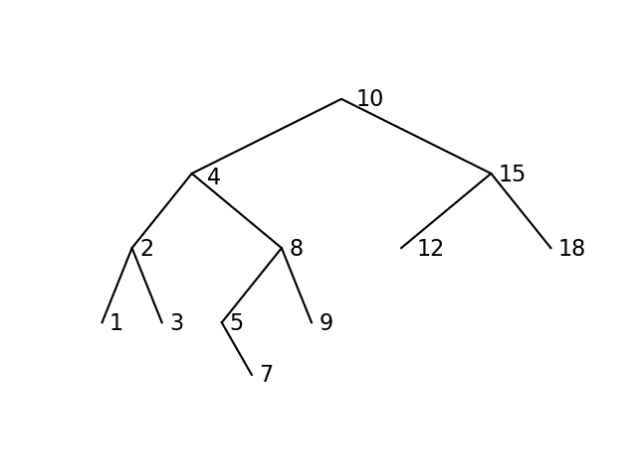
CS 2302 Data Structures

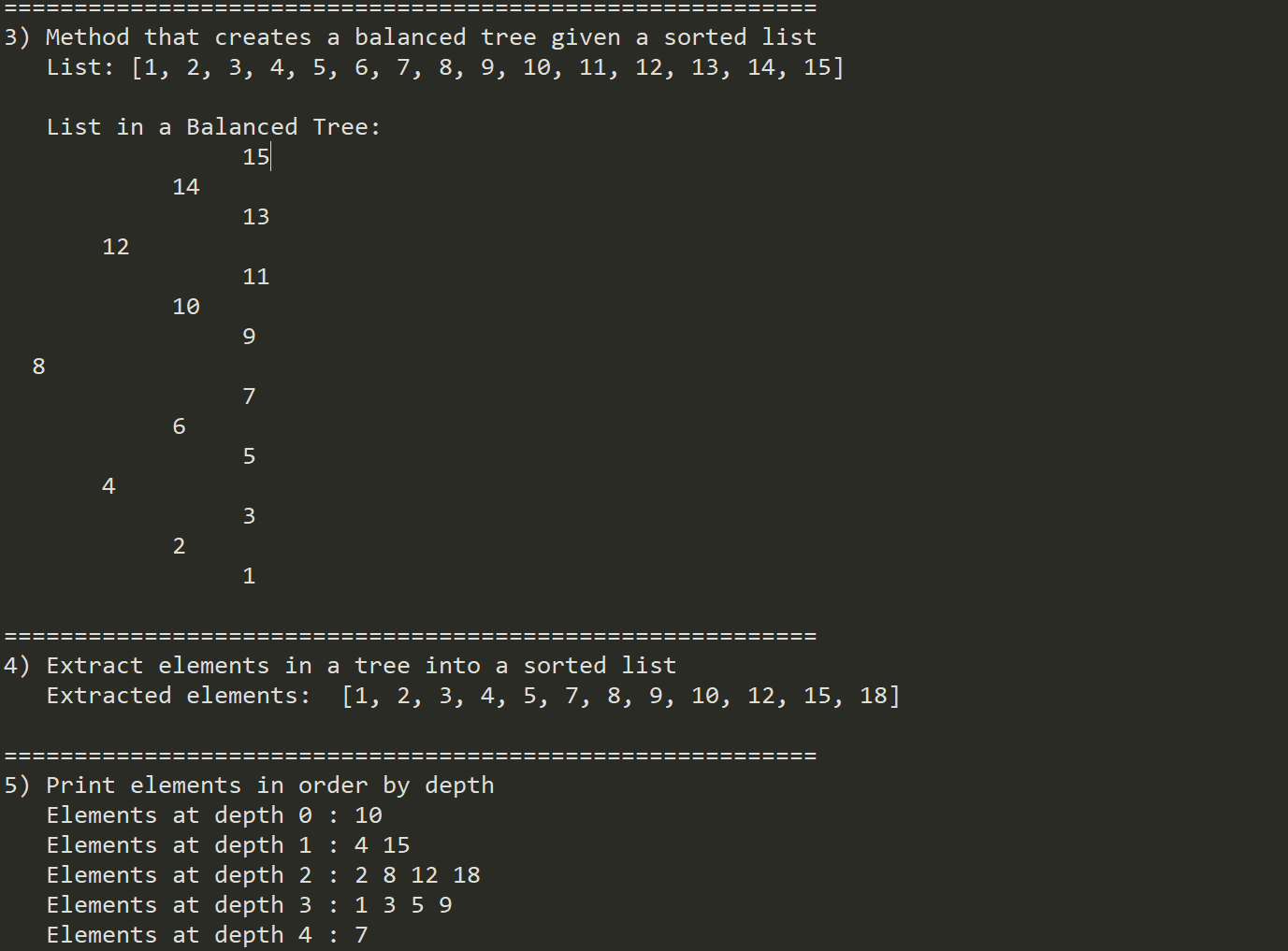
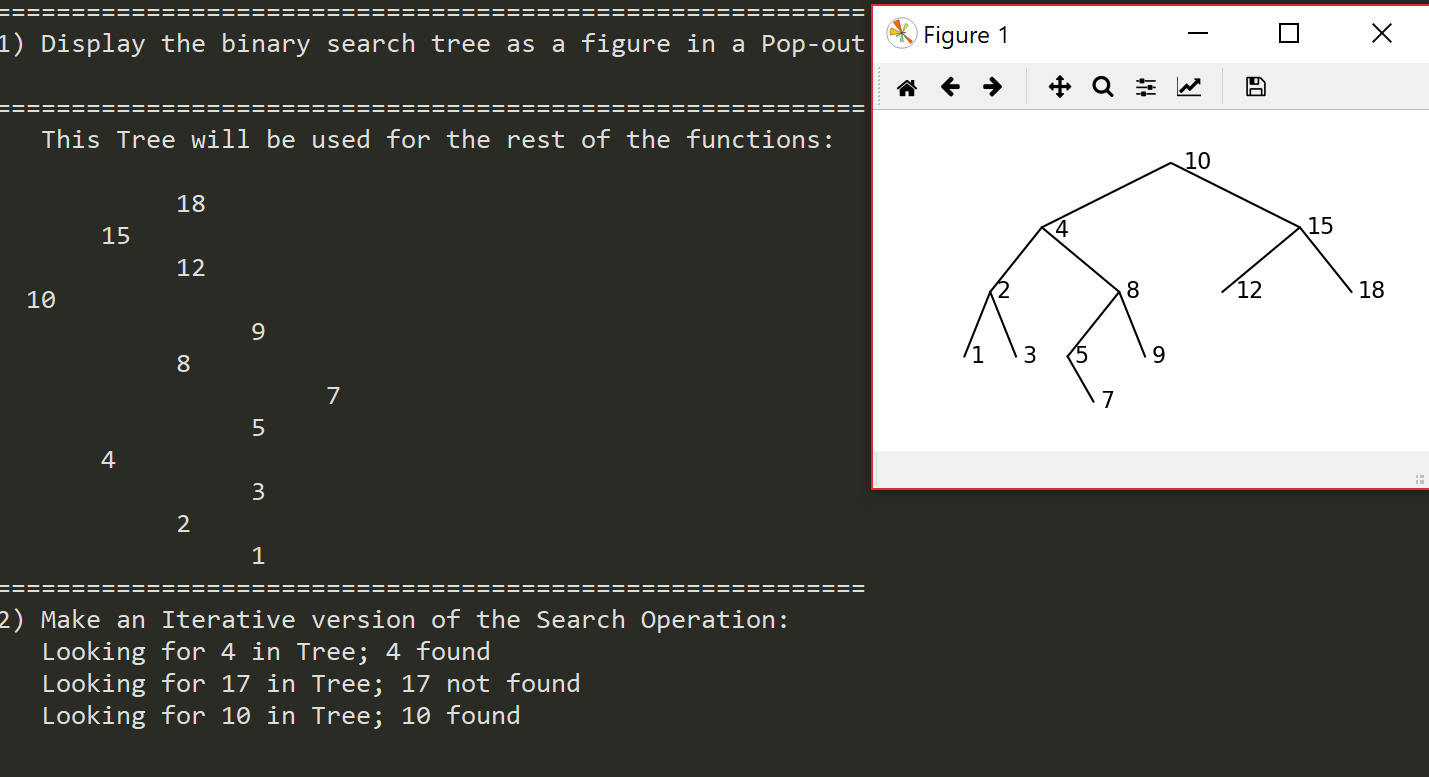
Lab 3

**Introduction:** In this lab we were given one file that defined a binary tree. We were supposed to create a tree, manipulate it, and extract information from it to display.

**Proposed solution:**

1. Displaying BST- I attempted to make the display through a recursive call, but as I was running out of time, I decided to hand plot every point using the coordinates from lab 1. The numbers were added one by one as well.
2. Iterative version of Search Operation- I followed the method Find and FindAndPrint given to us. Initially I checked to make sure T was none, and if it was, the method would return none. While T was not none, I checked if the parameter number that we were looking for was T.item, if it was then the method would return T. If k was greater than the current item, then T would equal T.right as that’s where numbers greater than the root go. Same for less than the current item, except T would equal T.left as the left contains numbers less than the root. I used FindAndPrintIter to print whether k was found or not. FindAndPrint was given to us, so I simply changed the Find method call to FindIter. If it was None, then it would print that it was not found, and if it was found, it would print the item as well as ‘found’.
3. Building a balanced BST- Initially I checked if the list was empty or if the length of the list was 0, in that case, I would return none. If the length of the list was 1, then I would create a BST and set the root as the initial number. Then I would return the element. If the list was longer than 1, I would take the length of the list and divide it by two, this would give me the middle item of the list. I used that to create a tree with the middle item as the root, then I would call the method again and set it equivalent to T.left, getting the elements left of the mid point by slicing the list. Then the right side would be called and return T. T should be balanced.
4. Extract Elements in a BST to a sorted list- To start the method, I check if T is not none, if that’s the case, then I call the method again but with T.left as a parameter this time, then I would append T.item to a list that was already defined, then I would call the method with parameters T.right. This would append all the elements in order to a list.
5. Printing Elements in a BST ordered by depth- First, my base case was that if T was none, I would return none. If k(the depth from the parameter) was 0 then I would print every element in that depth(which is the root). If k was greater than 0, I would call the method again, this time with k-1(reducing the depth) and T.left, to print the elements in order from smallest to largest. Then , I would call the method, again time with k-1(reducing the depth) and T.right, to print the elements in order from smallest to largest.

**Experimental Results:** 



**Conclusion:** I learned a lot on this lab. While learning to navigate and manipulate a tree, I managed to work on my recursive skills and improve them accordingly.

**Appendix:**

'''

Course: CS2302

Author: Erick Perchez

Assignment: Lab 2

Instructor: Dr. Fuentes

TA: Andita Nath

Date: 03/09/2019

Purpose: To make, manipulate, extract from, and understand trees and their

functions

'''

# Code to implement a binary search tree

# Programmed by Olac Fuentes

# Last modified February 27, 2019

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 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 SumTree(T):

if T is None:

return 0

if T is not None:

Sum = T.item

if T.right is not None:

Sum += SumTree(T.left)

Sum += SumTree(T.right)

return Sum

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 FindDepth(T,k):

if T is None:

return -1

if T.item == k:

return 0

while T is not None:

depth = 0

if T.item < k:

depth += 1

return depth + FindDepth(T.right,k)

if T.item > k:

depth +=1

return depth + FindDepth(T.left,k)

return depth

def SumAtDepth(T,d):

if T is None:

return 0

if d is 0:

return T.item

while T is not None:

depth = 0

Sum = 0

if depth < d:

depth += 1

return depth + FindDepth(T.right,d)

if depth > d:

depth -=1

return

return Sum

def FindAndPrint(T,k):

f = Find(T,k)

if f is not None:

print(f.item,'found')

else:

print(k,'not found')

''' #1

========================================================

'''

#using the lab1 assignment I found the coordinates and

#plotted them through a simple ax.plot.

def DisplayTree():

fig, ax = plt.subplots()

ax.plot((-5,0,5),(-10,0,-10), color='k')

ax.plot((-7,-5,-2),(-20,-10,-20), color='k')

ax.plot((2,5,7),(-20,-10,-20), color='k')

ax.plot((-8,-7,-6),(-30,-20,-30), color='k')

ax.plot((-4,-2,-1),(-30,-20,-30), color='k')

ax.plot((-4,-3),(-30,-37), color='k')

ax.set\_aspect(.25)

ax.axis('off')

#writes the number slightly off the vertices to not

#interfere with the tree display

ax.text(0.5, -1, '10', fontsize=16)

ax.text(-4.5, -11.5, '4', fontsize=16)

ax.text(5.25, -11, '15', fontsize=16)

ax.text(-6.75, -21, '2', fontsize=16)

ax.text(-1.75, -21, '8', fontsize=16)

ax.text(2.5, -21, '12', fontsize=16)

ax.text(7.25, -21, '18', fontsize=16)

ax.text(-7.75, -31, '1', fontsize=16)

ax.text(-5.75, -31, '3', fontsize=16)

ax.text(-3.75, -31, '5', fontsize=16)

ax.text(-.75, -31, '9', fontsize=16)

ax.text(-2.75, -38, '7', fontsize=16)

''' #2

========================================================

'''

def FindIter(T,k):

if T is None:

return None

while T is not None:

if k == T.item:

return T

#since K is greater than T.item, it sends it to

#the right(greater numbers than the root)

if k > T.item:

T = T.right

if k < T.item:

T = T.left

return None

#uses the FindAndPrint method given to us to print whether

#or not the element is found

def FindAndPrintIter(T,k):

f = FindIter(T,k)

if f is not None:

print(f.item,'found')

else:

print(k,'not found')

''' #3

========================================================

'''

def BalanceTree(List):

if List is None or len(List) == 0:

return None

#if the list only has one element, it simply gets

#added onto the Tree

if len(List) == 1:

T = BST(List[0])

return T

else:

mid = len(List)//2

T = BST(List[mid])

T.left = BalanceTree(List[0:mid])

T.right = BalanceTree(List[mid+1:])

return T

''' #4

========================================================

'''

def ExtractToList(T):

if T is not None:

#looks at the leftmost element and appends it to

#the list 'List' defined in the main then looks

#towards the right of the tree

ExtractToList(T.left)

List.append(T.item)

ExtractToList(T.right)

''' #5

========================================================

'''

def ElementsAtDepth(T,k):

if T is None:

return None

#base case for the recursion, it prints the elements

#at the desired depth

if k == 0:

print(T.item, end = ' ')

else:

#Since the desired depth is not reached yet

#it lowers the 'k' for the next recursion

ElementsAtDepth(T.left, k-1)

ElementsAtDepth(T.right, k-1)

'''

#Code to test the functions above

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)

S = None

InOrderD(T,'')

Find(T, 10)

print()

print(SmallestL(T).item)

print(Smallest(T).item)

FindAndPrint(T,10)

FindAndPrint(T,110)

n=1

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=5

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=15

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

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

InOrderD(T,'')

n=2

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

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

InOrderD(T,'')

DisplayTree()

'''

print('==========================================================')

print('1) Display the binary search tree as a figure in a Pop-out.')

DisplayTree()

print()

print('==========================================================')

print(' This Tree will be used for the rest of the functions:')

print()

T = None

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

for a in A:

T = Insert(T,a)

InOrderD(T, ' ')

print('==========================================================')

print('2) Make an Iterative version of the Search Operation: ')

k = 4

print(' Looking for', k, 'in Tree;', end= ' ')

FindAndPrintIter(T,k)

k = 17

print(' Looking for', k, 'in Tree;', end= ' ')

FindAndPrintIter(T,k)

k = 10

print(' Looking for', k, 'in Tree;', end= ' ')

FindAndPrintIter(T,k)

print()

print('==========================================================')

print('3) Method that creates a balanced tree given a sorted list')

List = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]

print(' List:', List)

print()

print(' List in a Balanced Tree:')

SA = BalanceTree(List)

InOrderD(SA, ' ')

print()

print('==========================================================')

print('4) Extract elements in a tree into a sorted list')

List = []

ExtractToList(T)

print(' Extracted elements: ', List)

print()

print('==========================================================')

print('5) Print elements in order by depth')

k=0

print(' Elements at depth',k,':',end= ' ')

ElementsAtDepth(T,k)

print()

k=1

print(' Elements at depth',k,':',end= ' ')

ElementsAtDepth(T,k)

print()

k=2

print(' Elements at depth',k,':',end= ' ')

ElementsAtDepth(T,k)

print()

k=3

print(' Elements at depth',k,':',end= ' ')

ElementsAtDepth(T,k)

print()

k=4

print(' Elements at depth',k,':',end= ' ')

ElementsAtDepth(T,k)

“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”

