Author: Carlos Fernando Castaneda

Class : CS 2302

Date Modified: March 11, 2019

Instructor: Olac Fuentes

Assignment: Lab 3 Binary Search Tree

TA: Anindita Nath & Maliheh Zaragan

**Introduction:**

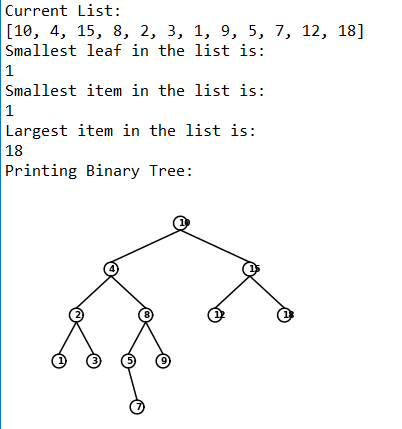
In python, you can use Binary Search Trees to organize items in a way that can be found using a structure that resembles a tree. Binary search trees can be very specific when finding items, and on this lab, I am going to play with some of these functions to solve 5 tasks. The first task is to find a way to print the binary tree exactly like it is in the lab worksheet, including circles and branches. The second task is to create an iterative version of the search algorithm. The third task is to build a balanced tree using O(n). The fourth task is to extract the elements of the binary tree to a sorted list using O(n). And finally, the last task is to print the elements of a tree ordered by its depth.

**Proposed solution design and implementation:**

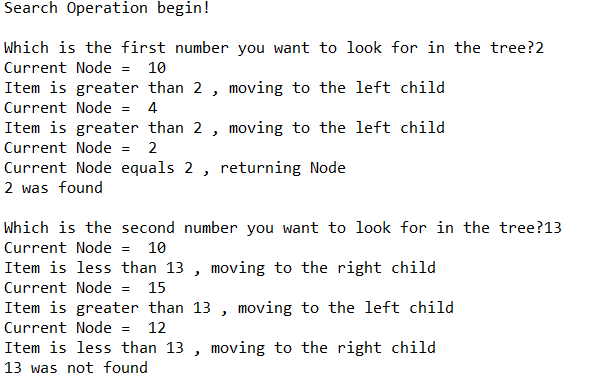
1. Printed BST: For the first task, it is obvious that I must use the mathplotlib function that I learned in Lab 1 to implement it in this lab. The tricky thing about this method is that there is a specific way to do it, what I mean is that there is a format to print this tree so that it is visible to the user in spyder. I suppose that the first thing that must be printed is the branches in the method, then a circle with a white fill, and afterwards the actual node that is placed on this tree. Afterwards, I must find a way to recursively add items to that tree so that it is correct.
2. Iterative Search: This task has no real changes to the current find method provided to us, only that every time it moves throughout the tree, it indicates whether the current node is less than or greater than the element k, which is the element that user wants to find. It also has to state whether or not it finds that item in the search tree.
3. Balanced BST: This task is tricky as it first requires altering the contents of a tree, and then prints them as done like in task 1. The sorting should be similar to Quicksort; however, I must implement it in a way that the time complexity is O(N). To complete this, I must absolutely not use any nested loops whatsoever. However, to keep track of my list, I must use some kind of index or indicator to keep track of the binary tree.
4. Sorted List: This method must use a way to return a list instead of a binary tree to check that it is correct. To make sure that I can still use the original tree for task 5, I will make a copy of the tree, and that copied tree will be altered to give me the desired result.
5. Print Depth: This method has to read the depth of each level of the binary tree from left to right. I will implement four layers for the purpose of this lab, as the binary tree I will provide has a depth of 5.

**Experimental results**:

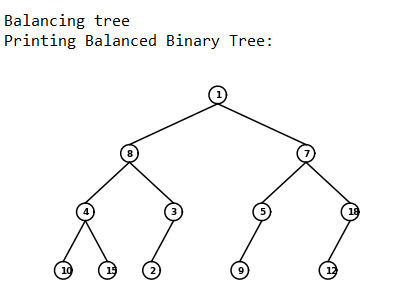
1. Printed BST: The method I used was almost identical to the one I made for Lab 1 when I made the circles and the brackets, however, this time I had to worry about building circles and placing the item in the list inside of the circle inside of that circle. What I did id call three different recursive calls throughout the program, one for drawing the circle and printing the item of the tree, one that builds the right branches, and one that draws the left branches of the tree. My result below was taken when importing the tree, comparing the smallest and largest items in the list, and the set binary tree.



1. Iterative Search: For this method, I had used the find method given by the professor with a few alterations I added. For starters, I added another case if T is none it will return none, if t equals k, it will return k. I also added print statements for every step occurring in the program, for example, if T.item was smaller than k, it will indicate so until it found or didn’t find k inside of the binary tree. Below has two user inputs that I integrated into the program to find 2 numbers, 2, which will test if it actually finds an item in the tree, and 13, an instance where the item is not in the tree.



1. Balanced BST: I used a method called ‘balancedTree’ that will help me find a median in the given copied list of the original tree, and sort the items starting from the lowest to give me my desired result. Below Is my result from using the original tree given in task 1 to reprint itself using the ‘balancedTree’ method.

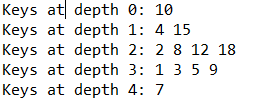


1. Sorted List: The method used for the sorted list starts the extraction of the given list from the left side of the tree, then adds all the items on the left side of the tree in the new list, the steps repeat for the right side of the the tree until all elements have been sorted in the list. My result below was from the tree given from task 1

.



1. Print Depth: On my method, I called it 5 times with integers 0 through 4 to indicate the 5 depths of the binary tree. The program will print if k is zero, else it will call itself twice based on the right and left side of the tree until it reaches the end of the list. The results below are based on the binary tree built for task 1.



**Conclusions**:

With this lab, I was able to learn to code better using the Python language, including using algorithms to find create and modify binary search trees to my advantage. I was also able to learn to solve different problems by using binary trees throughout my lab.

**Appendix :**

**binary\_tree.py**

"""

Author: Carlos Fernando Castaneda

Class : CS 2302

Date Modified: March 8, 2019

Instructor: Olac Fuentes

Assingment: Lab 3 Binary Search Tress

TA: Anindita Nath & Maliheh Zaragan

Purpose: to implement various algorithms and methods to fully understand the

process of how a Binary Search Tree works.

"""

#Imports various tools to help us plot the binary tree to be used in this lab

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

#This method creates the circle to be used when plotting the binary tree

def circle(center,radius):

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

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

y = center[1]+radius\*np.sin(t)

x = center[0]+radius\*np.cos(t)

return x,y

#This method plots the circles that will be shown when printing the binary search tree

def drawCircles(ax,center,r):

x,y = circle(center,r)

ax.plot(x,y,color='k')

#Method that inserts items into a binary tree

def Insert(T,i):

#If T is none, insert item i. This will be the root

if T == None:

T = BST(i)

#If the current item is less than item i, insert on the left branch

elif T.item > i:

T.left = Insert(T.left,i)

#If the current item is greater than item i, insert on the right branch

else:

T.right = Insert(T.right,i)

#Return Tree T

return T

# Prints items in BST in ascending order

def InOrder(T):

if T is not None:

InOrder(T.left)

print(T.item,end = ' ')

InOrder(T.right)

# Prints items and structure of BST

def InOrderD(T,space):

if T is not None:

InOrderD(T.right,space+' ')

print(space,T.item)

InOrderD(T.left,space+' ')

#Method that takes in a list Z, and returns a balanced list to be used to create a balanced tree

def balancedTree(Z):

#If the list Z is not None

if Z:

#Term median will be the node to keep track that the tree is currently being balanced

median = len(Z) // 2

T = BST(Z[median])

#Will insert the left sub-tree

T.left = balancedTree(Z[:median])

#Will insert the left sub-tree

T.right = balancedTree(Z[median+1:])

#Returns the new list T

return T

# Returns smallest item in BST. Returns None if T is None

def SmallestL(T):

if T is None:

return None

while T.left is not None:

T = T.left

return T

# Returns smallest item in BST. Error if T is None

def Smallest(T):

if T.left is None:

return T

else:

return Smallest(T.left)

# Returns largest item in BST. Error if T is None

def Largest(T):

if T.right is None:

return T

else:

return Largest(T.right)

#Method that will print the binary tree in its full shape, including its circles, branches, and numbers

def drawTree(ax,dx,dy,n,s,T):

#While T is not None

if T != None:

#Draws the circles that will be used

drawCircles(ax,[dx,dy],1.6)

#Contains the info on the current item it is being printed

ax.text(dx-.5, dy-.5, T.item, size=9, weight='bold')

#If T has a left child, then it will plot the left branch

if T.right != None:

ax.plot([dx,dx+(2\*\*n)],[dy-1.6,dy-s], color='k')

drawTree(ax,dx+(2\*\*n),dy-s-1.6,n-1,s,T.right)

#If T has a left child, then it will plot the left branch

if T.left != None:

ax.plot([dx,dx-(2\*\*n)],[dy-1.6,dy-s], color='k')

drawTree(ax,dx-(2\*\*n),dy-s-1.6,n-1,s,T.left)

# Returns the address of k in the Binary Tree, or None if k is not in the tree. This version is also iterative, meaning it will point out the current steps that it is doing

def Find(T,k):

#If T is empty, it will return none. This also checks if the item was not found

if T is None:

return None

#Prints current item

print('Current Node = ',T.item)

#If T is none, it will return the node

if T is None:

print('Current Node is None, returning Node')

return T

#If T equals k, it will return T

elif T.item == k:

print('Current Node equals',k,', returning Node')

return T

#If the item is less than k, it will return to find the item on the right branch

elif T.item<k:

print('Item is less than',k,', moving to the right child')

return Find(T.right,k)

#If the item is greater than k, it will return to find the item on the left branch

elif T.item>k:

print('Item is greater than',k,', moving to the left child')

return Find(T.left,k)

#If none of the above, it will return None

else:

return None

#Method that extracts the data of a list, and

def extractTree(T, L):

#While T is not None

if T != None:

#Checks through every item in a tree on the left side

extractTree(T.left, L)

#Adds a new item to L before returning

L += [T.item]

#Checks through every item in a tree on the right side

extractTree(T.right, L)

#Method that prints all of the items located in a particular depth of the Binary Tree. For example, k = 0 will return all the items located in depth 0

def printDepth(T, k):

#While T is not None

if T != None:

#At depth 0

if k == 0:

#Prints the item at depth 0

print(T.item, end=" ")

else:

#Passes through all of the node at the level 'k' on the left and right side

printDepth(T.left, k-1)

printDepth(T.right, k-1)

#Method that finds an item in the Binary Tree and prints the item, if it wasnt found, i will indicate so.

def FindAndPrint(T,k):

#Sends variable f to method Find, which will try to locate item k in the current Tree

f = Find(T,k)

#If find doesnt return none, the item was found

if f is not None:

print(f.item,'was found')

#If None was returned, the item is not in the tree

else:

print(k,'was not found')

# Code to test the functions above

#Sets a new Empty tree caled T

T = None

#Sets a new list called A. The numbers being used are the ones described in the Lab Assignment for Lab 3

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

#Prints the initial List that will be used in the program

print('Current List: ')

print(A)

#Inserts the content of the List into the tree T

for a in A:

T = Insert(T,a)

print('Smallest leaf in the list is: ')

print(SmallestL(T).item)

print('Smallest item in the list is: ')

print(Smallest(T).item)

print('Largest item in the list is: ')

print(Largest(T).item)

#Task 1

#Prints the binary tree using mathplotlib so it displays like that in the Lab Assignment

print('Printing Binary Tree: ')

fig, ax = plt.subplots()

drawTree(ax,0,0,4,9,T)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

#Task 2

#Starts the Iterative Search Operation with two numbers that the User can input to check the output

print('Search Operation begin!')

k1 = int(input('Which is the first number you want to look for in the tree?'))

FindAndPrint(T,k1)

k2 = int(input('Which is the second number you want to look for in the tree?'))

FindAndPrint(T,k2)

print()

#Task 3

#Makes a duplicate of list A and then proceeds to create a new balanced tree on the input of that list

#Copies List A to Z to keep the original list. List Z will be sent to method Balance

Z = A

#Sends list Z to method balancedTree so it can be balanced before it is printed

print('Balancing tree')

S = balancedTree(Z)

#Prints the balanced tree in the form of task 1. All of the preferences are still intact

print('Printing Balanced Binary Tree: ')

fig, ax = plt.subplots()

drawTree(ax,0,0,4,9,S)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

#Task 4

#A\_copy is new list that will extract the data of T and will print a sorted list to the user

A\_copy = []

print('Extracting data')

extractTree(T, A\_copy)

print('Printing extracted data, here is your new list:', A\_copy)

print()

#Task 5

#Prints the depth of a particular Binary Tree. The current tree we have righ now has a depth of 4, hence we print the four depths of the tree

#Prints at depth 0

print('Keys at depth 0:', end=' ')

printDepth(T,0)

print()

#Prints at depth 1

print('Keys at depth 1:', end=' ')

printDepth(T,1)

print()

#Prints at depth 2

print('Keys at depth 2:', end=' ')

printDepth(T,2)

print()

#Prints at depth 3

print('Keys at depth 3:', end=' ')

printDepth(T,3)

print()

#Prints at depth 4

print('Keys at depth 4:', end=' ')

printDepth(T,4)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 provide inappropriate assistance to any student in the class.

