Joaquin Hidalgo

CS 2302 MW @1:30pm

Lab 03

**Binary Search Tree (BST)**

**Introduction:** The problem at hand is to demonstrate visual representation

of the BST. As well as get familiar with basic commands such as insert, delete

and print. As well as implement functions to receive more info about the

BST such as iterative search, build balanced BST from sorted list in O(n)

time, extra BST nodes into sorted list at time O(n) and sum levels at depth

**Proposed solution design and implementation:** To show the visual BST, I would like to insert a node while drawing the actual structure of the BST. I would first start with the base case by checking if the BST is empty. When Tree is empty, insert and draw circle. If BST is not empty, either traverse left or right to appropriate position. Update new center of each object after every traverse either left or right.

The iterative search will require a for-loop or while-loop to continue until we find the item or until there are no more nodes to check. Check if Tree is empty, if so then return None. If Tree’s current item is the searchee, return searchee. Else, if Tree’s item is less than the searchee, then iteratively move to the left side of the BST. Move to the right is Tree’s item is greater than searchee.

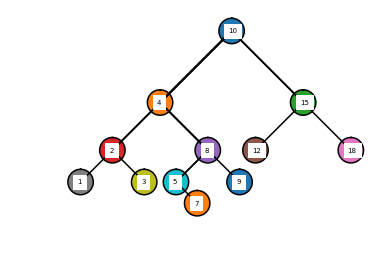
To build balanced BST given sorted list have features that we can assume. The middle node of the sorted list is the root. If we recursively pass the left half and right half of the list to only have the middle item of that new left half or right half become the child of the parent node, which in this case is the root node. This way you only visit each item once traversing through the list making it time of O(n).

To build a sorted list from a BST, we can assume that the first item of the newly created list is the furthest left node of the Tree. Traversing in, In-order (left, node, right) passes through the BST only once in ascending order. Instead of printing when getting to each node, just append to list thus making a sorted array.

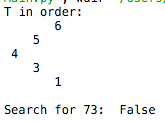
Here to print all BST by level I would create a for loop to change the level each time the for loop happens by increasing ‘I’ by 1 from 0 to the depth of the tree. I would create a function to find the certain depth of a node. Compare of the depth of that node is equal the given variable for d. If equal, then print. If not recursively call itself again with the left and right child of that node. While going down to children node, decrease the variable d by 1 in order to hopefully make the variable d eventually, zero

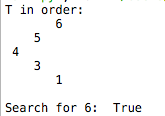
**Experimental results:**

Drawing a BST to match question #1

################# ****

Iterative search function with different searchee items for question #2.

Search is NOT FOUND🡪 

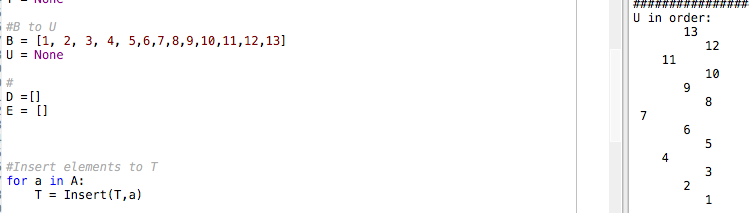
Search is FOUND🡪 

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

Question #3

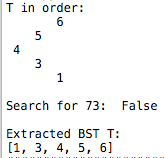
List to BST 1 🡪

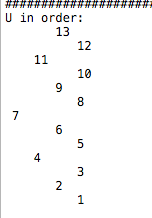
List to BST 2 🡪



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

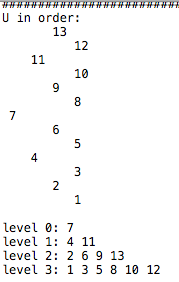
Question #4

Extracted BST to list 1 🡪 

Extracted BST to list 2🡪 

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

Question #5

Print by level 🡪 

**Conclusion:** I learned how to represent a BST with actual nodes and values. I also learned how to traverse through the BST. The BST can be traversed iteratively or recursively. I learned to make a BST that is balanced from a sorted list. The middle node of the list is the root node. Also, if I extract from the BST and depending and traverse In-Order, as I visit each node, I will append each item to the list thus, creating a sorted list. I learned how to print each level of a BST separately using a method to tell me what the height of each node is. If node height matches depth given, print it each item.

**Code:**

﻿#!/usr/bin/env python3

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

"""

Created Sun 03 Mar 19:01:07

Instructor: Olac Fuentes

TA: Dita and Mali

Last modified Sun 10 Mar 22:10:28

The purpose of this lab is to demonsotrate visual representation

of the BST. As well as get familar with basic commands such as insert, delete

and print. As well as implament functions to recieve more info about the

BST such as itterative search, build ballanced BST from sorted list in O(n)

time, extra BST nodes into sorted list at time O(n) and sum levels at depth

"""

import math

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

#Insert object into BST

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

#delete from BST

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

# 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+' ')

# 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

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)

# Returns largest item in BST

def Largest(T):

if T.right is None:

return T

else:

return Largest(T.right)

# Returns the address of k in BST, or None if k is not in the tree

def Find(T,k):

if T is None or T.item == k:

return T

if T.item<k:

return Find(T.right,k)

return Find(T.left,k)

# Search and print item

def FindAndPrint(T,k):

f = Find(T,k)

if f is not None:

print(f.item,'found')

else:

print(k,'not found')

#Get height of BST

def GetHeight(T):

if T == None:

return 0

left = GetHeight(T.left)

right = GetHeight(T.right)

#At the last node of the BST (leaf)

if T.left == None and T.right == None:

if left > right:

return GetHeight(T.left)

else:

return GetHeight(T.right)

#Children node are not empty, add 1 and continue to count edges of path

else:

if left > right:

return GetHeight(T.left) + 1

else:

return GetHeight(T.right) +1

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

# Question 2

# Iterative search in BST and return boolean

def Search(T, k):

if T == None:

return None

#iter method

while T != None:

# down, right BST.

if T.item < k:

T = T.right

# down, left BST.

elif T.item > k:

T = T.left

# found

else:

return True

if T == None:

return False

#Question 3...

#Build BST from sorted list in O(n) time

def BuildBST(B):

if len(B) == 0:

return

# mid node of list becomes parent node of next nodes

mid = len(B)//2

root = BST(B[mid])

# Pass the rest of the list into left and right sections

root.left = BuildBST(B[:mid])

root.right = BuildBST(B[mid+1:])

return root

#Extract BST to sored order list in O(n)

# question 4

def ExtractToList(T, A):

#Pass thru list InOrder and append instead of print

if T != None:

ExtractToList(T.left, A)

A.append(T.item)

ExtractToList(T.right, A)

# Question 5

# Find level at depth d

def PrintDepth(T, d):

if T == None:

return

#compare depth of node to 'd'

if FindDepthOf(T, T.item) == d:

print(T.item, end=" ")

else:

#Check rest of nodes while increasing level, thus 'd-1'

PrintDepth(T.left, d-1)

PrintDepth(T.right, d-1)

# Return depth of 'k'

def FindDepthOf(T,k):

if T == None:

return -1

if T.item == k:

return 0

if T.item < k:

#traverse down right to find k

d = FindDepthOf(T.right, k)

else:

#traverse down left to find k

d = FindDepthOf(T.left, k)

#if not found, d = -1. Else add up each edge until your back to root node

if d <0:

return -1

return d + 1

# Question 1

#Insert to BST and draw out representation

def InsertAndDraw(ax,side, c, insertee, W):

if W == None:

#insert

W = BST(insertee)

#draw circle

base\_circle(ax,1,c,800,1.0)

ax.text(c[0]-200, c[1]-100, insertee, fontsize=7, backgroundcolor = "w")

elif W.item > insertee:

#drawline

p = np.array([ [c[0],c[1]], [c[0]-side, c[1] -side] ])

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

side = side //1.5

c = p[1]

#insert

W.left = InsertAndDraw(ax, side, c, insertee, W.left)

else:

p = np.array([ [c[0],c[1]], [c[0]+side, c[1] -side] ])

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

side = side //1.5

c = p[1]

W.right = InsertAndDraw(ax, side, c, insertee, W.right)

return W

#This method returns all x,y coordinates that generates a circle with sin&cos

def get\_circle\_points(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

#Draws circle

def base\_circle(ax,n,center,radius,w):

#get x,y coordinates of circle

x,y = get\_circle\_points(center,radius)

#draw the circle

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

ax.fill(x,y)

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

#User defined presets

center = [0,0]

sideLen = 4500

#Draw BST List to match question 1

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

W = None

#A to T

A = [4, 3, 5, 1, 6 ]

T = None

#B to U

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

U = None

#

D =[]

E = []

#Insert elements to T

for a in A:

T = Insert(T,a)

#Print T in order

print("T in order: ")

InOrderD( T,'' )

print()

# iterative search

# Question 2

print("Search for 73: ", Search(T, 73) )

print()

#Extract BST to list

#Question 4

ExtractToList(T, D)

print("Extracted BST T:")

print(D)

print("#########################")

#Build Balanced BST

#Question 3

U=BuildBST(B)

#U = SortedToBST(B)

print('U in order: ')

InOrderD( U,'' )

print()

# Print Level's

# Question 5

for l in range(GetHeight(U)+1 ):

print("level", l, end = ': ')

PrintDepth(U, l)

print()

print()

#Extract BST to list

#Question 4

ExtractToList(U, E)

print("Extracted BST U: ")

print(E)

print()

print("###########################")

# DRAW BST -->

plt.close("all")

fig, ax = plt.subplots()

for b in drawingBST:

W = InsertAndDraw(ax, sideLen, center, b, W)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('bst01.png')

**4. Standards of Conduct and Academic Dishonesty**

* 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.