David Amparan

CS Lab 3

Professor: Fuentes, Olac

March 11, 2019

**Introduction**

For our third lab assignment, we implemented binary trees and their various useful implementations for storing and finding data. We were assigned to create total methods, draw a binary tree (like in lab 1), iterative version of the binary search method, building a binary out of a sorted list, extracting the elements into an ordered list, and printing all items at a certain depth of the binary tree.

**Solution**

1. Drawing the binary tree
   1. Drawing a binary tree, I used the code that was used in lab 1, except this time at each point we draw a circle and insert a number within that circle, check for the children and draw the connections needed
2. Iterative Search
   1. For the search method we can traverse this with only a while loop by moving according to its value (comparing item). This is possible since we only need one item and don’t need to traverse the entire tree
3. Tree out of a sorted list
   1. To create this tree, we need to slice the list into halves, insert the item at the middle, then recursively append the rest of the tree
4. Extracting elements into ordered List
   1. To extract the elements all we need is to do is traverse the tree much like with the ordered print method
5. Printing at a Depth
   1. We pass the tree as well as a key for the depth and recursively traverse up to that depth (key is zero) and print the contents at that depth

**Experimental Results**

The only experiment results were attained from the drawn tree method which successfully took a stamp of the time. All others recorded 0 consistently. But as we can see the drawn tree method grows rapidly once the input seize begin to grow and has a rapid growth rate.

**Conclusion**

With the completion of this lab we can see the functionality and ease of manipulating data with a binary tree. Depending on the task at hand, you can simply iterate through the tree or go through the entire tree. Also, ordering information within a tree is very simple and comes with many advantages. In all, binary trees are a useful ADT for information sorting and allocation.

**Appendix**

I, David Amparan certify that this code was written by myself with no external help from classmates or any internet sources. If anything is found on this code I will take full responsibility of the course of actions as described in the syllabus.

* David Amparan

"""

Author: David Amparan

Instructor: Dr. Fuentes, Olac

TA: Anindita Nath, Maliheh Zargaran

Last Modified: 3/10/2019

Purpose: Lab 3 involves discovering the funtionality and ease of

binary trees. Various funtions are providede as well as designed by myself

"""

import numpy as np

import math as math

import matplotlib.pyplot as graph

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')

"""

Method Name: iterFind | Parameters: Tree, key

Functionality: The find(t,k) finds the key within the tree and

returns that tree node

"""

def iterFind(T, key):

temp = T

while temp is not None:

if temp.item == key:

return temp

#if the key is greater or less

if temp.item > key:

temp = temp.lef

else:

temp = temp.right

return -1

"""

Method Name: getLength Parameters: Tree

Functionality: Will attain the total size, number of nodes and return this

"""

def getLength(T):

if T is None:

return 0

#we have an item so we add on to it

if T is not None:

return 1 + getLength(T.right) + getLength(T.left)

"""

Method Name: getChildren Parameters: width radius, lessThan

Functionality: Will return point based the size of the width and

if the point for that item is bigger or smaller than its root

"""

def getChildren(lessThan, origin):

#if the point we want is less that the original root

child = [0,0]

#if the item coordinate is less than the origin

if lessThan is True:

child[0] = origin[0] - 20

#the item must be bigger than the original root and belongs to the right

if lessThan is False:

child[0] = origin[0] + 20

child[1] = origin[1] - 20

return chil

"""

Method Name: drawTree Parameter: Tree, axis, width, radius

Functionality: This method draws the binary tree like how we drew the tree

in Lab 1

"""

def drawTree(Tree, axis, origin, cirRad):

if Tree is None:

return

#if there is more than 0 nodes

totalP = np.zeros((2,2))

right = getChildren(False, origin)

left = getChildren(True, origin)

if Tree.right is not None:

totalP[0] = origin

totalP[1] = right

#totalP[1] = totalP[1]\*.71

#totalP[0] = totalP[0] + (totalP[1]\*.40)

axis.plot(totalP[:,0], totalP[:,1], color='k')

axis.plot()

if Tree.left is not None:

totalP[0] = left

totalP[1] = origin

#totalP[0] = totalP[0]\*.71

#totalP[1] = totalP[1] + (totalP[0]\*.40)

axis.plot(totalP[:,0], totalP[:,1], color='k')

axis.plot()

#we must attain the coordinates in which we will plot

x,y = circle (origin, cirRad)

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

#here we add text

#the text positioning is off this corrects it

text = [0,0]

text [0] = origin[0]-2

text [1] = origin[1]-1

axis.annotate(Tree.item, origin, xytext=text)

#check for right and left to conenct the line

drawTree(Tree.right, axis, right, cirRad)

drawTree(Tree.left, axis, left, cirRad)

"""

Method Name: circle | Parameters: center, radius| Returns an xy array

Functionality: circle will calculate the circles circumference

and create a set of points for that circumeference

"""

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

'''

Method Name: orderedTreeList Parameters: L

Functionality: Will take in an ordered list and return a tree built from

that ordered list, will not use the insert function

'''

def orderedTreeList(L):

if len(L) == 0:

return None

if len(L) == 1:

return BST(L[0])

#here we find our middle

t = None

middle = len(L)//2

#we append our middle here

t = BST(L[middle])

#recursive call

t.left = orderedTreeList((L[:middle]))

t.right = orderedTreeList(L[middle+1:])

return t

'''

Method Name: orderedExtract | Parameters: Tree

Functionality: The method will take in a binary tree and extract the elements into a sorted list

and finally it will return that list as well

'''

def orderedExtract(T):

if T is None:

return []

L = []

L = orderedExtract(T.left)

L = L +[T.item]

L = L + orderedExtract(T.right)

return L

'''

Method Name: printAtDepth(T, D) | Parameters: t

Functionality: Will print all the items at that depth

'''

def printAtDepth(T,D):

if T is None:

return

if D == 0:

print(T.item, end=' '

printAtDepth(T.left, D-1)

printAtDepth(T.right, D-1)

t = None

A = [6,4,8,20,10,15,2]

for a in A:

t = Insert(t, a)

fig, axis = graph.subplots()

drawTree(t, axis, [0,0], 8)

axis.set\_aspect(1)

graph.show()

print()

k = 10

print("Iterative Search of ", k, ":" , end=' ')

if iterFind(t,k) == -1:

print(-1)

else:

print(iterFind(t,k).item)

print()

print("----------------------------------------------------------")

B = [0,2,4,6,10,25,31

print("Balanced Tree")

print()

InOrderD(orderedTreeList(B), ' ')

print()

print('------------------------------------------------------------')

print("Extracting an Ordered List from Tree")

print("Ordered List: ", orderedExtract(t))

print()

print('---------------------------------------------')

d = 2

print("Printing elements at depth ", d)

print("Depth",d, ":", end=' ')

printAtDepth(t,d)