Armando Ortiz

Feb 27, 2019

CS 2302 Data Structures

Lab Report 3

UTEP

Instructor: Olac Fuentes

**Introduction**

The purpose of this lab is to use binary search trees to implement several different functions: drawing a BST, iterative version of a search algorithm, building a balanced BST using a sorted list as an input, building a sorted list using a BST as an input, and printing the elements of a tree by depth. The list and BST building functions are meant to run in O(n) time.

**Proposed Solution Design and Implementation**

The function to draw the tree was the most tricky. It uses combinations of the circle and tree drawing functions from the first lab. An array was still used to draw the lines themselves, with the new left and right points being found by subtracting y from second space on the array, and subtracting x and x plus the radius of the circle on the left side, and adding x and x plus the radius on the right side. This extra addition was done to give more space between the bubbles, so they wouldn’t overlap. Two circles needed to be created, a larger one for the outline, and a slightly smaller one for the inner white circle. The lines were plotted checking to see if left or right were not none to make sure extra lines weren’t drawn. The text kept as close to center by subtracting the x and y of the text by half the radius of the inner circle. The order in which the figures were drawn had to be made so the lines wouldn’t show over the circles or text. Finally two recursive calls were made to move through the left and right branches, and divide the x by two.

The iterative search function was more complicated because it could be done a lot more simply using recursion. First, there needed to be a check to make sure that the list itself wasn’t empty. Then, a loop was created to traverse the list. The first if statement checks to see if T.item is the same as k, the item being searched for, and if it is the item is returned to the printing function to tell the user that it has been found. If not there is an if statement to check if the current node is larger than k, and if it there’s another check to make sure T.left is not null, which would cause the loop to break. Then, the current node is set to be the left node. This same process is done if T.item is smaller than the key, which would cause a move to the right. If the loop breaks by reaching the end of the tree then None is returned, and the user is informed that the item was not found.

Building a balanced BST works almost like binary search. A midpoint is found in middle of the list, and placed as the root of the tree. Then smaller lists are recursively called with numbers to the left of the midpoint, creating the left branches. Another recursive call does this with items to the right of the midpoint, using them as right branches. The tree is then returned.

Building a sorted list using a BST was similar to InOrder printing. The recursive call for the left side of the tree was called first to move all the way to the smallest item in the list. After, the Item is appended to the List, which was created as an empty python native list outside of the function. Then the right side is called using the recursive function. There are no return statements since the list is being built within the function, and the addresses are the same.

Printing the elements by depth needed two functions: a larger function to iterate through each depth of the tree and another one that goes through the list to reach each depth, and print the item. The iterative function calls the height function to set a variable used as an endpoint for the loop. It then calls the recursive function each pass to search the list recursively. The function call for the recursive call initializes a number at zero and adds one each call until it matches the other variable for the iterative number of the loop.

**Experimental Results**

**Output**

The tree was drawn correctly, but there was an issue with the text inside of the bubbles being slightly off center. The lines themselves work, this was tested by adding numbers to the end of the original list, and placing it at the smallest leaf of the right branch.

**A close up of a mans face

Description automatically generated**

The rest of the functions worked correctly, and the output matched what was required by the lab worksheet. The iterative function was tested by using all the numbers in the tree as a key, and then using a few numbers that were not inside of the tree. These all worked.

**A close up of text on a white background

Description automatically generated**

**Runtimes**

According to the complexity analysis of each of the functions, the search function runs at O(n), the balanced tree function runs at O(n), sorted list runs at O(n).

**Table(runtime found using Time.Time())**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Runs** | **DrawTree** | **FindL** | **BalancedTree** | **SortedList** | **FindDepth** |
| **1** | 0.05213 | ﻿3.50475 | ﻿2.9802 | ﻿1.09672 | ﻿0.00090 |
| **2** | 0.04788 | ﻿0.00014 | ﻿1.9789 | ﻿8.10623 | ﻿0.00052 |
| **3** | 0.05102 | ﻿2.7894 | ﻿4.7207 | ﻿ ﻿6.91413 | ﻿0.00085 |
| **4** | 0.04991 | ﻿7.70092 | ﻿8.08238 | ﻿6.19888 | ﻿0.00082 |
| **5** | 0.05294 | ﻿4.0054 | ﻿3.19481 | ﻿9.05991 | ﻿0.00090 |

**Conclusion**

This lab was very helpful in understanding how to move through a Binary search tree, and why it’s better to move through it recursively. This lab also showed a roundabout method for printing the depths of a tree. It also showed that it can be simple going from Python list to a tree, and back to a list. Finally, it provided more practice with drawing figures using Matplotlab.

**Appendix**

﻿"""

Author: Armando Ortiz

Course: CS 2302 Data Structures

Assignment: Lab 3

Instructor: Olac Fuentes

T.A.: Anindita Nath, Maliheh Zargaran

Last Modification: March 11, 2019

Purpose: The purpose of this program is to perform different operations using a Binary Search Tree

-Display a drawing of a BST

-Iteratively search for a key in a BST

-Build a balanced tree using a sorted list.

-Extracting the elements of a tree to create a sorted list

-Print the keys of the tree by depth

"""

import numpy as np

import matplotlib.pyplot as plt

import math

import time

class BST(object):

# Constructor

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

self.item = item

self.left = left

self.right = right

#Meant to insert new node to tree

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

#Finds height of tree

def height(T):

if T is None:

return 0

else:

left = height(T.left)

right = height(T.right)

if left > right:

return left + 1

else:

return right+1

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+' ')

#Recursive Functions to find key in BST

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)

#Prints whether k is found or not

def FindAndPrint(T,k):

f = Find(T,k)

if f is not None:

print(f.item,'found')

else:

print(k,'not found')

#Iterative Function to find key in BST

def FindL(T,k):

if T is None:

return None

while T.item is not None:

if T.item == k:

return T

if T.item > k:

if T.left is None:

break

T = T.left

if T.item < k:

if T.right is None:

break

T = T.right

return None

#Prints whether key is found or not

def FindLAndPrint(T,k):

f = FindL(T,k)

if f is not None:

print(f.item,'found')

else:

print(k,'not found')

#Creates a balanced tree using sorted List

def BalancedTree(L):

if len(L)==0:

return None

mid = len(L)//2

T = BST(L[mid])

T.left = BalancedTree(L[:mid])

T.right = BalancedTree(L[mid+1:])

return T

#Builds a sorted list using BST

def BuildList(T,L):

if T is not None:

BuildList(T.left,L)

L.append(T.item)

BuildList(T.right,L)

#Function to print the keys of BST by Depth

def PrintAtDepth(T,d):

i = 0

z = height(T)

while i < z :

print("Keys at Depth", i ,end=': ')

PrintDepth(T,0,i)

print("")

i +=1

#Function to move through tree to find

def PrintDepth(T,d,k):

if T is not None:

if d == k:

print(T.item,end=' ')

PrintDepth(T.left,d+1,k)

PrintDepth(T.right,d+1,k)

#Function for creating circle

def circle(cen,rad):

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

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

x = cen[0]+rad\*np.sin(t)

y = cen[1]+rad\*np.cos(t)

return x,y

#Function to draw tree

def DrawTree(ax,T,line,x,y,r):

if T is not None:

left = [line[0]-x-(x+r),line[1]-y-r]#Creates new point for left branch

right = [line[0]+x+(x+r),line[1]-y-r]#Creates new point for right branch

x1,y1 = circle(line,r)#Outline circle

r2 = r\*.9#radius of inner circle

x2,y2 = circle(line,r2)#Creates inner circle

if T.left is not None:

ax.plot([line[0],left[0]],[line[1],left[1]],color = 'k',zorder=0)#Draws left branch

if T.right is not None:

ax.plot([line[0],right[0]],[line[1],right[1]],color='k',zorder=0)#Draws right branch

ax.fill(x1, y1, 'k',x2,y2,'w',zorder=2)#Draws Circles

ax.text(line[0]-r2\*.5,line[1]-r2\*.5,T.item,fontsize=10,zorder=3)#Places text inside of circle

DrawTree(ax,T.left,left,x/2,y,r)

DrawTree(ax,T.right,right,x/2,y,r)

T = None

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

for i in A:

T = Insert(T,i)

print("T")

InOrderD(T, ' ')

start = time.time()

PrintAtDepth(T,0)#Print keys by Depth

end = time.time()

print("Print Depths runtime: ",(end-start))

print("")

#Find recursively and iteratively

start = time.time()

FindLAndPrint(T,8)

end = time.time()

print("Search runtime: ",(end-start))

start = time.time()

FindLAndPrint(T,23)

end = time.time()

print("Search runtime: ",(end-start))

#Balanced Tree from Sorted List

print("")

print("T2")

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

start = time.time()

T2 = BalancedTree(B)

end = time.time()

print("Building Tree runtime: ",(end-start))

print(" ")

InOrderD(T2, ' ')

start = time.time()

PrintAtDepth(T2,0)

end = time.time()

print("Print Depths runtime: ",(end-start))

#Build List using Tree

L = []

start = time.time()

BuildList(T,L)

end = time.time()

print("Sorted List runtime: ",(end-start))

print(L)

#Drawing the binary search tree

plt.close("all")

fig, ax = plt.subplots()

line = np.array([0,0])

start = time.time()

DrawTree(ax,T,line,100,100,40)

end = time.time()

print("Drawing Tree runtime: ",(end-start))

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('BTree.png')

A screenshot of a cell phone

Description automatically generated