Lab 3 Report

Traffic Enforcer Javier Soto x1522

University of Texas at El Paso

Author Note

As required by UTEP CS2302

Abstract

The purpose of this lab is to demonstrate my knowledge on the use of binary search trees as well as to see the effect this data type has on run times at different input sizes. I must be able to create binary search trees given a list of items in liner time as well as create a list of items given a binary search tree with the same time complexity. For this lab I must also be proficient at accessing items in the search tree that are at a given depth. The following report illustrates my implementation of these functions.

Lab 3 Report

This python script I have written has been one of my more text heavy script as compared to my previous codes. Though still easy to trace, I have added more function and more print statements so that demoing my lab will be more easily readable and all methods can be tested with one execution of my script.

# Question 1

For question 1 I am asked to display a figure depicting a binary search tree. To do so I recycled my code form lab one, which I had luckily already named draw b tree, short for draw binary tree. The only modifications I needed to make was to add a circle and in the circle, I needed to add the item of the corresponding bst node. Before I get into the details of the function to perform this action, I added a function to this abstract data type called MaxHiegth that will return he height of the longest path from root to leaf. My implementation gets the max height of the left sub tree and compares it against the right sub tree then returns the size of the largest path.

This max height size is then used to know the number of recursive calls to make to draw the tree. If the max height isn’t yet exceeded, then I create a circle using a given x and y value that will be the origin of the root of the tree. The radius of all circles is set to .5. I then add the circle to the plot. Next, I add text the text is a string version of the current nodes item. Once this is drawn I recursively draw the left and right sub tree decreasing the max height by one and shifting the x and why values to their new locations. The recurrence relation for this functions evaluates at T(n) = 2T(n/2)+1

## Question 21

For question 2 I am asked to implement a search function that instead of accessing nodes by recursion, traverses the tree by iteration. To do so I create a temporary tree that is equal to the tree we are searching if the node item equals the item we are looking for then we return the item and its sub tree. If the item, we are looking for larger than the current node item then we know it will be in this current node’s right child sub tree so we increment the tree to equal the right child and we enter the loop again. Alternatively, if the searchee is smaller than the nodes item then we move to the nodes left child. If we iterate to a null node than the loop is not entered, and we return null since the item has not been found in the tree. As required by the lab the time complexity for this function is 0(n). Though it is not optimal, it is an alternative option to searching a binary search tree.

### Question 3.

As mentioned above, for this question, I need to build a balanced binary search tree given a sorted list as input. As a side note I have assumed that I would be given a sorted list as input however if I had a little more time for this lab I could have wrote a function that takes a list sorted or unsorted and creates a balanced tree. I would first check if the list is sorted. If not sorted I can take two paths either insert the random nodes and restructure should the tree be unbalanced or recycle code from lab 2 that would use a sorting algorithm, then pass it list to my already crated function. However, as we have seen at best I can sort a list in O(logn) time with a modified quick sort algorithm.

With a sorted list the algorithm is simple I take a list, I make a node where the head is equal to the middle of the sorted list. I then pass the rest of the work to my method reclusively. I set the left child equal to some sub tree that contains the sub set list of elements to the left of the middle element. The right child is set to equal some sub tree containing the elements of a sub set of elements to the right of the middle element. This works because the recursive call is made with the parameters I just mentioned, and the process is repeated, set the node item to mid list and make the two recursive calls and so on. Oddly enough, I found it was easier for me to code this function then it was for me to develop the pseudo code prior to coding. For some reason I got hung up on developing the base case. I repeatedly asked myself when we stop this process, how do we know when to end. Obviously enough when the subset is empty then we have no more elements to add. This thought was not so easy to come across at first.

One of the specifications for this function was that the bst need be balanced. That specification was easy to accomplish because every recursive call splits the list in to two sub sets that will have at most one element more than the other depending on the length of the original list being even or odd. For this reason, no matter the length of the list the tree will have at most one path of length greater than all others by one edge, which is the definition of a balanced binary tree. Because I am not using the ADT implementation of insert, my function has a time complexity of O(n)

#### Question 4.

For question for I needed to reverse the process of question two, I would need to take a binary tree as input and return a sorted list. To start I already have a function that traverses the tree in an ascending fashion. Print in order traverses the tree in the matter I need for this question furthermore the recurrence equation for in order traversal is T(n) = 2T(n/2)+C that is < 0(n), just what I need. For question for I crate a list, instead of printing the item I append the item to this list which is done in constant time.

##### Question 5.

I found question 5 to be the most challenging because my initial pseudo code, when coded, did not bare my anticipated results. Initially I wrote a code similarly to a search or print at depth function that state if depth is 0 do something with that item else re-call function with the sequential node and subtract one from depth. The results I got from this algorithm was a printing of one path then a printing of another path with the same depth printed more than once with the corresponding items. Since there was no specification to the running time requirements of this function this is what I did. I first get the max height of the tree, as described in earlier functions, I then create a list and initialize it with a length equal to the max height of the tree. For each index of the list I append string “Key at depth” and the index. What I do next is I traverse the tree recursively at each depth of the tree I append the item of this depth to the corresponding index of the created list. For example, at depth 0 I append to index one a string version of the item and so on. Once the tree is traversed and list is complete I now am able to print the list that has the elements at the correct index/depth.

In conclusion, I found this lab to be very helpful in different traversal techniques of a binary search tree. I learned how to traverse a binary tree though iterative means as opposed to reclusively in case I find myself needing to do this in future work.

1Tables

Table 1

[Table Title]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input size | 100 | 1000 | 10,000 | 100,000 |
| Question 1 | .162185 | 2.303938 |  |  |
| Question 2 | 0 | 0 | 0 | 0 |
| Question 3 | .047585 | .053866 | .049867 | .109700 |
| Question 4 | .000997 | .058842 | 01.357379 | 13.089131 |
| Question 5 | 0 | .001995 | .017981 | .301265 |
| Entire script | .239909 | 02.487214 | --- | Still running till this day |

All times are measured in seconds. I was unable to measure question two with these input sizes. For some questions, input size 10k and beyond are infeasibly large.

***Source code***

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

"""

Created on Tue Mar 5 13:00:43 2019

@author: Javier Soto

Professor Olac fuentes

TAs:

-Anindita Nath

-Maliheh Zaragaran

IA

-Eduardo Lara

Peer Leader

-Erick Macik

the puspose of this lab is to demonstrate my knowldge of implementing Binary

search trees as provided by Professor Olac Fuentes

"""

import bst

import matplotlib.pyplot as plt

from datetime import datetime as dt

import random

def SearchIte(T,k):

#will search a binary tree for item k with out using recursion

temp = T

while temp is not None:

if temp.item == k:

return temp

elif temp.item > k:

temp = temp.left

else:

temp = temp.right

return None

def draw\_btree(ax,n,x,y,delta\_x,delta\_y,T):

if T is not None:

temp = T

if n>0:

# taken from my implementation of draw tree from Lab1 CS2302

#for ease of ploting create 2 lists of corrisponding x,y values

x\_values = [x-delta\_x,x,x+delta\_x]

y\_values = [y-delta\_y,y,y-delta\_y]

circle = plt.Circle((x, y), .5)

ax.add\_artist(circle)

ax.text(x, y, str(temp.item), fontsize=10)

ax.plot(x\_values,y\_values)

#draw center then left child then right child

draw\_btree(ax,n-1,x\_values[0],y\_values[0],delta\_x\*.5,delta\_y\*.5,temp.left)

draw\_btree(ax,n-1,x\_values[2],y\_values[2],delta\_x\*.5,delta\_y\*.5,temp.right)

def ListToTree(L,T):

#will take a sorted list and create a balanced BST

if len(L) > 0:

head = int(len(L)/2)

T = bst.BST(L[head])

#T.item = L[head]

#to create right sub tree

T.right = ListToTree(L[head+1:],T.right)

#to create left sub tree

T.left = ListToTree(L[:head],T.left)

return T

def TreeToList(T,L):

#will take a BST and return a sorted list of the elements in that tree

if T is not None:

TreeToList(T.left,L)

L.append(T.item)

TreeToList(T.right,L)

def AtDepth(T,index,items):

#Will take a BST and print the items based on there location in the tree

if T is not None:

items[index] = items[index] + str(T.item) + ' '

AtDepth(T.left,index+1,items)

AtDepth(T.right,index+1,items)

def Question1(T):

print('Question 1')

print('Please see "Figure1"')

plt.close("all")

fig, ax = plt.subplots()

height = bst.MaxHeight(T)

draw\_btree(ax,height,100,100,10,10,T)

ax.axis('off')

plt.show()

print()

def Question2(T):

print('Question 2')

#searchee= int(input('Which number Would you like to search for?'))

searchee= random.randint(1,100000)

temp = SearchIte(T,searchee)

if temp is not None:

print('The subtree with item ' + str(searchee) + ' is: ',end = ' ')

bst.InOrder(temp)

else:

print('Item ' + str(searchee) + ' was not found in this tree')

print()

def Question3():

#sorted list to create a tree from

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

TempTree = None

TempTree = ListToTree(B,TempTree)

print('Question 3')

print('The new tree from sorted list is: ' , end =' ')

bst.InOrder(TempTree)

print()

Question1(TempTree)

def Question4(T):

List =[]

TreeToList(T,List)

print('Question 4')

print('The new list created from a tree is: ')

print(\*List, sep = ', ')

print()

def Question5(T):

print('Question 5')

items =[]

Height = bst.MaxHeight(T)

for i in range(Height):

items.append('Key at depth ' + str(i) + ': ')

AtDepth(T,0,items)

for i in range(len(items)):

print(items[i])

print()

start =dt.now()

T = None

A = []

for b in range(10000):

A.append(random.randint(1,10000000))

for a in A:

T = bst.Insert(T,a)

Question1(T)

Question2(T)

Question3()

Question4(T)

Question5(T)

end =dt.now()-start

print('it took: ' + str(end))

Academic dishonesty

I, Javier Soto, certify that this script and lab report are of my own unless otherwise documented above.

