Lab 3 Report

Problem:

In this lab we were asked to draw a binary search tree, then we need to apply the search algorithm iteratative version then a method that gets a binary search tree and converts it into an ordered array. Then we get an array and convert it into a binary search tree. Lastly print each item by depth.

Proposed solution:

I thought about using the first lab for part 1 about drawing the tree. For part 2 which was using iteration to Search I used a temporary variable to not lose the root then I checked if my number was bigger or smaller if not return the address of the root. Once it finds it returns a statement and the number searched. This should take at most least O(logn) because you are only looking at half of the elements in T. In the extract method where we get a tree and make it an ordered array. I had a condition when it was an empty node return none otherwise make a recursive call on the right side and on the left side. Then you return the left side plus the current node item plus the right side. This is in done in O(n) time.

The build tree method works in similar fashion instead of checking if the tree is empty you check the list length when it is one you return the item at index 0 as a node once you find the middle. Otherwise you recursively call it on the left side of the list and the right side it should run in O(n). I kept getting a error with length of middle. For printing the depth I used a separate method to find the depth and then to iterate through I used a loop to go up to the depth.

Conclusion:

I learned that recursion is simplest case should be your base case and the recursive case should be how you can get to a base case.

Appendix

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 Search(T,k):

temp = T

while temp is not None:

if temp.item < k:

T = T.right

if temp.item > k:

T = T.left

if T.item == k:

print(k, 'found')

return T

def extract(T):

if T is None:

return []

if T is not None:

small = extract(T.left)

large = extract(T.right)

return small + [T.item] + large

def depth(T,d):

if T is None:

return None

if d==0:

return T.item

left = depth(T.left,d-1)

right = depth(T.right,d-1)

return left, right

def printbydepth(T,d):

for i in range(d):

print("keys at depth ", i ,depth(T,d))

T = None

A = [70, 50, 90, 130, 150, 40, 10, 30, 100, 180, 45, 60, 140, 42]

for a in A:

T = Insert(T,a)

Search(T,50)

print(extract(T))

printbydepth(T,2)

Academic Agreement:

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.

Andres Arellanes