PAOLA TERRAZAS

CS2302

LEARNING TO WORK WITH BINARY TREES AND ALL ITS FUNCTIONS

DR. FUENTES

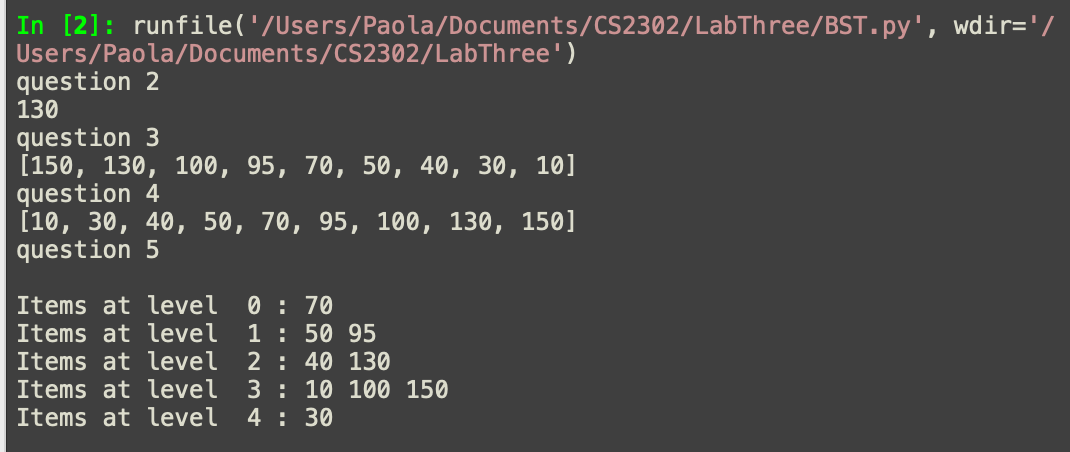
TA - ANINDITA NATH

3.12.2019

REPORT

In our lab 3 assignment, the problem we are trying to solve is figuring out and working with a binary search tree and its operations. Some operation examples are insertion, deletion, search and display. Besides working with a few of its most common operations, we were also asked to display the binary search tree as a figure using matplotlib. I began by looking at the example code provided bst.py and reading every single line of code until I understood exactly what each line was doing. I began by working with number one, which was the most difficult in my opinion. Although I know how binary search trees work perfectly, I still find it difficult to work with the concept on matplotlib. I attempted it several times and was close enough but never got the actual figure that was asked for, therefore I did not include it in my code. For problem number two, I found it very easy.

As we already had the method known as Find(T,k), it was the same concept but we had to rewrite it without recursion which was fairly simple. While T.item is not the key that we are looking for, then it depends if its bigger or smaller than the root. If it’s smaller, then the key is searched on the left side and if its bigger then the key id searched on the right side. For question number three, we needed to build a binary search tree given a sorted list. My method check is T is empty, if it is not then the root is T.item then I declared variables for left side and the right side of the binary search tree. For question number four, we needed to extract the elements in a binary tree into a sorted list. That being the opposite of number three. I created an empty, adding the left side, adding the roots and then adding the right side to the list. For question number five, we had to print the elements in a binary tree ordered by depth. I have three different methods, one that finds the find the depth, one that prints the depth and the last one that iterates through the list to print the order. My list was as follows [70, 50, 95, 130, 150, 40, 10, 30, 100]. Following is the output of each question from my code.



From this project, I learned everything there is to know about binary search trees. I found the concept fairly simple, understandable and easy to work with. The only issues I had with this lab was the concept of drawing in python. That concept is something I still have to research to learn more about. But, other than that, I know how to work with and use a binary search tree very well.

**Source Code**

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 InOrder(T):

if T is not None:

InOrder(T.left)

print(T.item, end = ' ')

InOrder(T.right)

def InOrderD(T,space):

if T is not None:

InOrderD(T.left,space+ ' ')

print(space, T.item)

InOrderD(T.right,space+ ' ')

def Smallest(T):

if T is None:

return None

while T.left is not None:

T = T.left

return T

def SmallestR(T):

if T.left is None:

return T

return SmallestR(T.left)

def Largest(T):

if T.right is None:

return T

else:

return Largest(T.right)

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)

def SumTree(T):

if T is None:

return 0

return T.item + SumTree(T.right) + SumTree(T.left)

def FindDepth(T,k):

# returns the depth of k in BST, or None, if k is not in the tree

if T is None:

return -1

if T.item == k:

return 0

if T.item <k:

d = FindDepth(T.right,k)

else:

d = FindDepth(T.right,k)

if d == -1:

return -1

else:

return d+1

def SumAtDepth(T,d):

if T is None:

return 0

if d ==0:

return T.item

return SumAtDepth(T.left,d-1) + SumAtDepth(T.right,d-1)

# 2) Iterative version of the search operation

def Search(T,k):

while T.item != k:

if T.item > k: # if k is smaller than root searches on left side

T = T.left

elif T.item < k: # if k is bigger than root go to right side

T = T.right

return T

# 3) Building a balanced binary search tree given a sorted list

def SortedTree(T):

if T is None:

return []

elif T is not None:

root = [T.item]

left = SortedTree(T.left)

right = SortedTree(T.right)

return right + root + left # all together the left side the right side and root

# 4) Extracting the elements in a binary search tree into a sorted list

def SortedList(T):

if T is None:

return []

array = [] # creates empty list

i = [T.item] # puts T.item into empty list

array = array + SortedList(T.left) # puts T.left in list

array = array + i # adds root to list

array = array + SortedList(T.right) # adds T.right

return array

# 5) Printing the elements in a binary tree ordered by depth

def findDepth(T): # same as my FindDepth method but without key as parameter

if T is None:

return 0

else:

left = findDepth(T.left)

right = findDepth(T.right)

if left > right:

return left + 1

return right + 1

def printDepth(T, i):

if T is None:

return

if i == 0:

print(T.item, end= ' ')

elif i > 0:

printDepth(T.left, i-1)

printDepth(T.right, i-1)

def OrderedByDepth(T):

h = findDepth(T)

i = 0

while i < h:

print() # println in java

print("Items at level ", i, ":", end= ' ') # prints

printDepth(T, i)

i = i + 1

T = None

A = [70, 50, 95, 130, 150, 40, 10, 30, 100]

for a in A:

T = Insert(T,a)

#print(PrintInOrder(T, [], 0))

#InOrderD(T,'')

#print(Print(T))

#print(Smallest(T).item)

#print(Largest(T).item)

#print(SumTree(T))

print("question 2")

print(Search(T,130).item)

# question 3

print("question 3")

list = SortedTree(T)

print(list)

print("question 4")

# question 4

Tree = SortedList(T)

print(Tree)

# question 5

print("question 5")

OrderedByDepth(T)