Erick Perchez

80582912

CS 2302 Data Structures

3/24/2019

Lab 4

**Introduction:** In this lab we were given one file that defined a binary tree. We were supposed to create a tree, manipulate it, and extract information from it to display.

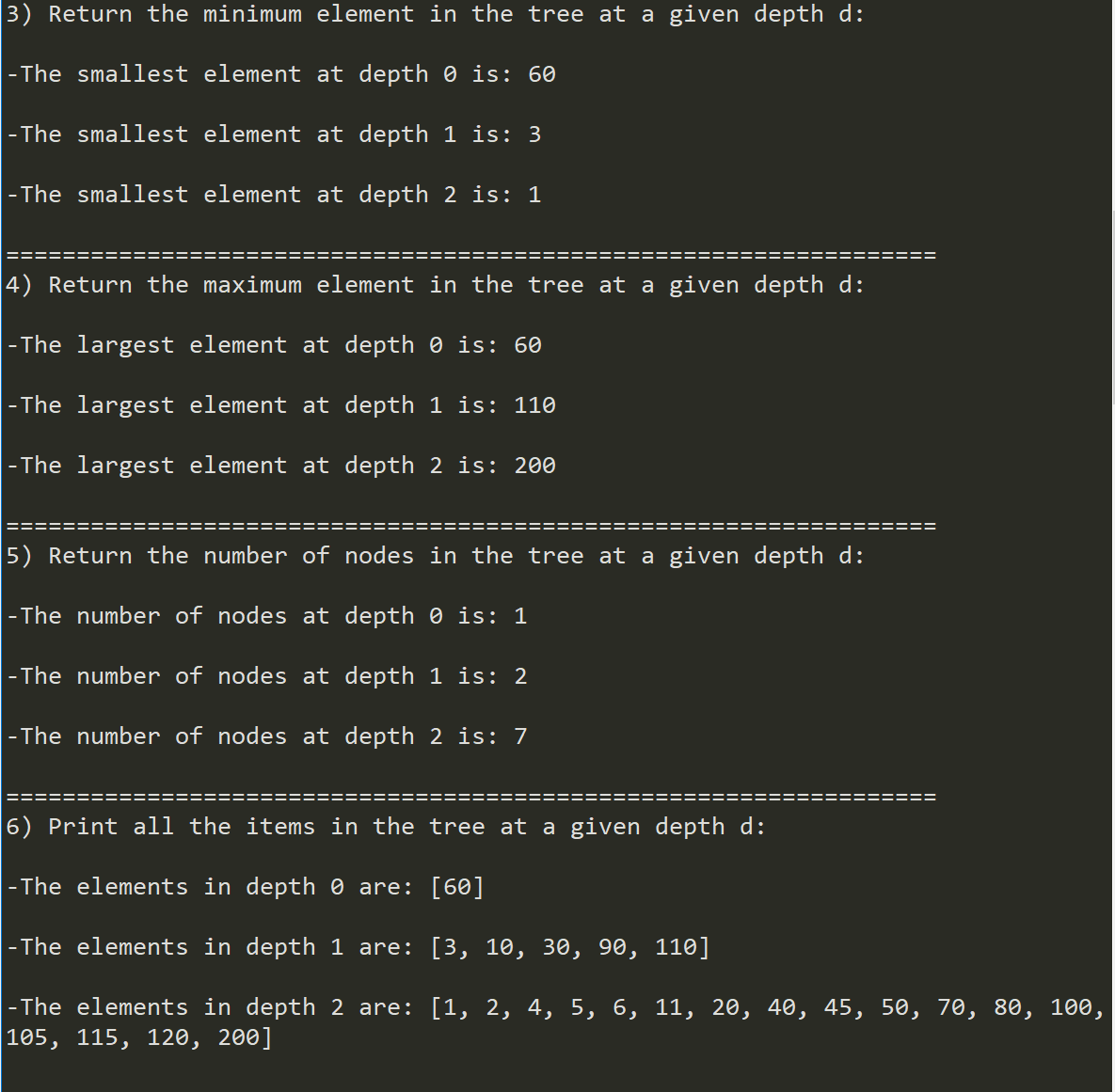
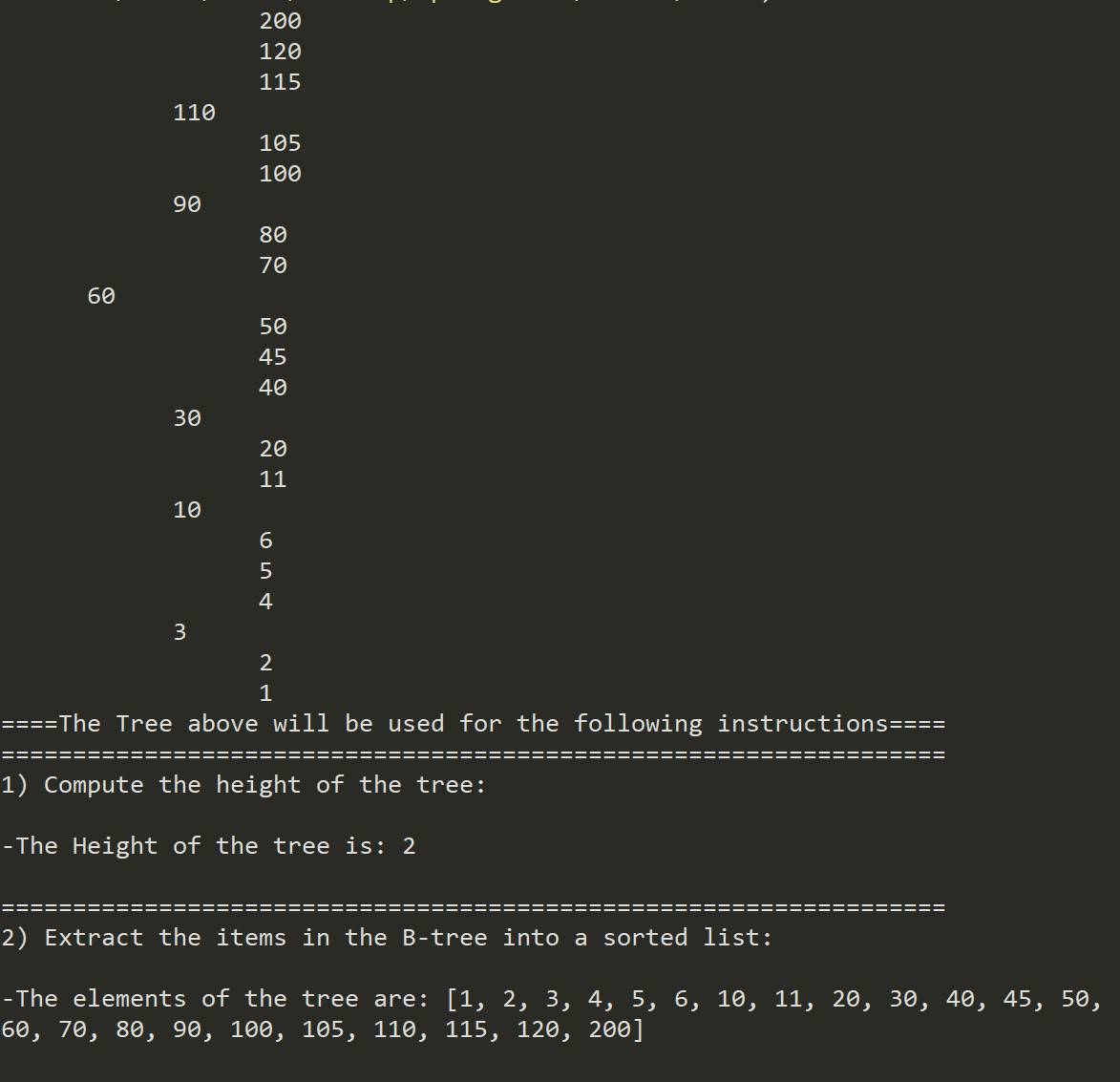
**Proposed solution:**

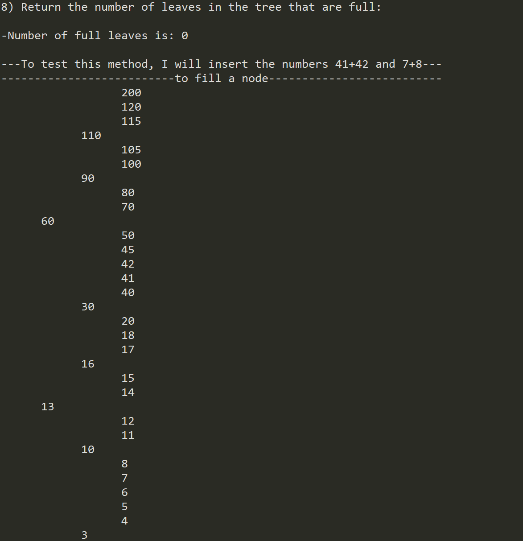
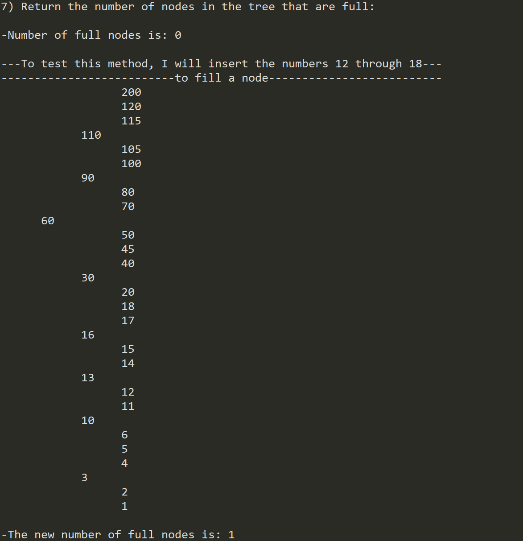
1. Computing the height of the tree- I simply created a makeshift counter in the return statement that also was a recursive call. As soon as it reached a leaf, it would return a 0.

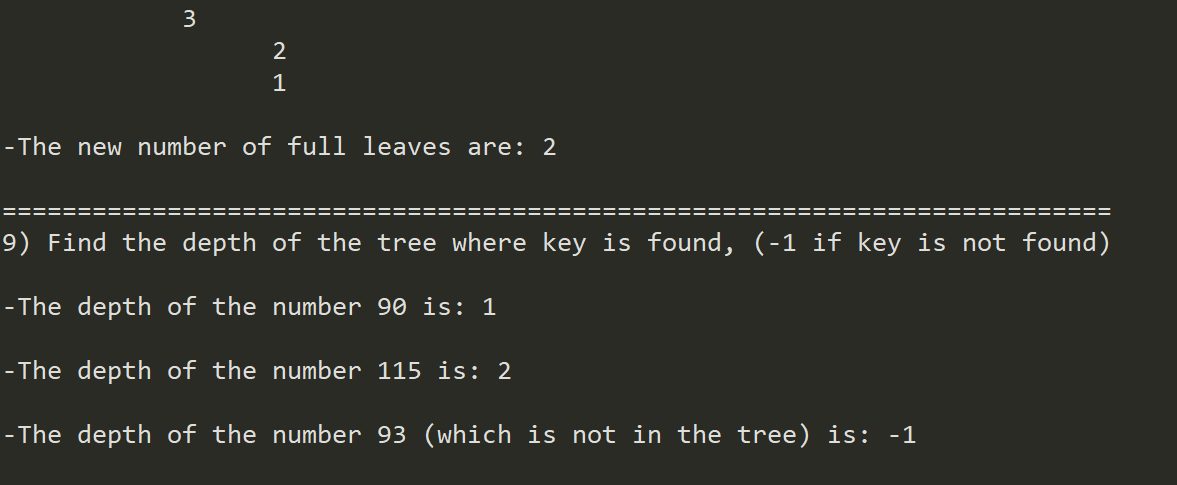
O(log(n))

1. Extract the items in The B-Tree- To create this method I started with the base case, if T was a leaf, then I would append all of the elements to a list. I also created a for loop that makes a recursive call until it reaches the children (base case) then appends those elements, then the parents, then the right children. At the end, I appended the very last child to the list then that returns the list.
2. Return the minimum element at given depth- I created a base case that if d was 0, that meant we had reached the desired depth, which would return the lowest element in index 0. In the case that a leaf is reached, I printed the smallest element of all the leaves. My return statement is a recursive call that calls the very first child and the depth is reduced by 1.
3. Return the maximum element at given depth- I created a base case that if d was 0, that meant we had reached the desired depth, which would return the largest element in index -1. In the case that a leaf is reached, I printed the largest element of all the leaves. My return statement is a recursive call that calls the last child and the depth is reduced by 1.
4. Return number of nodes in the tree at a given depth- To start, I created the base case like the previous two that determined when the desired depth was reached. If it was reached, then it would return 1 as only one node was visited. I would return 0 if T was a leaf. I then initiated a count variable to be used in a for loop. I added that count to a recursive call that went through every child and reduced the depth by 1. I would finally return the count.
5. Print all items at a given depth- As the previous methods, my base case was that if d was 0, I added a conditional to where if T.item was not None, it would return T.item. Else, a list would be created and a for loop would run through the children and the elements at the desired depth would be appended to the list. I made a recursive call in the for loop and finally returned the list that would then be used to print the elements at the depth desired.
6. Return the number of full nodes- For this method, I used a given method called IsFull that determines if a node is full. My base case was IsFull and it would return 1, if a leaf was reached, it would return 0. Then a count was initiated and used in a for loop that made a recursive call to see if the children were nodes, the count was then added to that. Finally, count would be returned.
7. Return the number of full leaves- As a base case I checked it T was a leaf, inside that conditional was another that checked if the node was full with the method IsFull. If that was the case, it would return 1. Then a count was also initiated and used in a for loop that ran for all the children. The count would be added to a recursive call. Finally, count would be returned.
8. Find the depth of a given key- This method was the trickiest one, I initially check if k was in T.item, if that was the case, I would return 0. If a leaf was reached, that would mean that the key was not in the tree so it would return a -1. Then a for loop was used to run through the children. In that for loop, to find the depth, I used a method called FindChild that was given to us that gave me the child in which the key was located in conjunction with a recursive call. Finally, depth would be checked, if it was -1 then it would return -1. If it was anything other than -1, it returns the depth plus the current depth which was 1.

**Experimental Results:**







**Conclusion:** I learned a lot on this lab. While learning to navigate and manipulate a B tree, I managed to work on my recursive skills and improve them accordingly.

**Appendix:**

# Code to implement a B-tree

# Programmed by Olac Fuentes

# Last modified February 28, 2019

'''

Course: CS2302

Author: Erick Perchez

Assignment: Lab 2

Instructor: Dr. Fuentes

TA: Andita Nath

Date: 03/20/2019

Purpose: To make, manipulate, extract from, and understand B trees and their

functions

'''

class BTree(object):

# Constructor

def \_\_init\_\_(self,item=[],child=[],isLeaf=True,max\_items=5):

self.item = item

self.child = child

self.isLeaf = isLeaf

if max\_items <3: #max\_items must be odd and greater or equal to 3

max\_items = 3

if max\_items%2 == 0: #max\_items must be odd and greater or equal to 3

max\_items +=1

self.max\_items = max\_items

def FindChild(T,k):

# Determines value of c, such that k must be in subtree T.child[c], if k is in the BTree

for i in range(len(T.item)):

if k < T.item[i]:

return i

return len(T.item)

def InsertInternal(T,i):

# T cannot be Full

if T.isLeaf:

InsertLeaf(T,i)

else:

k = FindChild(T,i)

if IsFull(T.child[k]):

m, l, r = Split(T.child[k])

T.item.insert(k,m)

T.child[k] = l

T.child.insert(k+1,r)

k = FindChild(T,i)

InsertInternal(T.child[k],i)

def Split(T):

#print('Splitting')

#PrintNode(T)

mid = T.max\_items//2

if T.isLeaf:

leftChild = BTree(T.item[:mid])

rightChild = BTree(T.item[mid+1:])

else:

leftChild = BTree(T.item[:mid],T.child[:mid+1],T.isLeaf)

rightChild = BTree(T.item[mid+1:],T.child[mid+1:],T.isLeaf)

return T.item[mid], leftChild, rightChild

def InsertLeaf(T,i):

T.item.append(i)

T.item.sort()

def IsFull(T):

return len(T.item) >= T.max\_items

def Insert(T,i):

if not IsFull(T):

InsertInternal(T,i)

else:

m, l, r = Split(T)

T.item =[m]

T.child = [l,r]

T.isLeaf = False

k = FindChild(T,i)

InsertInternal(T.child[k],i)

def Search(T,k):

# Returns node where k is, or None if k is not in the tree

if k in T.item:

return T

if T.isLeaf:

return None

return Search(T.child[FindChild(T,k)],k)

def PrintDescending(T):

if T.isLeaf:

for i in range(len(T.item)-1,-1,-1):

print(T.item[i], end =' ')

else:

for i in range(len(T.item)-1,-1,-1):

PrintDescending(T.child[i])

print(T.item[i], end = ' ')

PrintDescending(T.child[0])

def Print(T):

# Prints items in tree in ascending order

if T.isLeaf:

for t in T.item:

print(t,end=' ')

else:

for i in range(len(T.item)):

Print(T.child[i])

print(T.item[i],end=' ')

Print(T.child[len(T.item)])

def PrintD(T,space):

# Prints items and structure of B-tree

if T.isLeaf:

for i in range(len(T.item)-1,-1,-1):

print(space, space,T.item[i])

else:

PrintD(T.child[len(T.item)],space+' ')

for i in range(len(T.item)-1,-1,-1):

print(space, space ,T.item[i])

PrintD(T.child[i],space+' ')

def SearchAndPrint(T,k):

node = Search(T,k)

if node is None:

print(k,'not found')

else:

print(k,'found',end=' ')

print('node contents:',node.item)

def Smallest(T):

if T.isLeaf:

return T.item[0]

while T.isLeaf is not True:

return Smallest(T.child[0])

def Largest(T):

if T.isLeaf:

return T.item[-1]

while T.isLeaf is not True:

return Largest(T.child[-1])

''' #1

========================================================

'''

#Compute the height of the tree

def Height(T):

if T.isLeaf:

return 0

#uses child 0 to determine the height

return Height(T.child[0]) + 1

''' #2

========================================================

'''

def TreeToList(T):

if T.isLeaf:

#appends all leaves to the list

for i in range(len(T.item)):

List.append(T.item[i])

return

for i in range(len(T.item)):

#starts with first child to append in order

TreeToList(T.child[i])

List.append(T.item[i])

#recursive call for the last child

TreeToList(T.child[-1])

return List

''' #3

========================================================

'''

def MinElement(T,d):

#if d is 0, we found our depth, so print element at

#index 0

if d == 0:

return T.item[0]

#in case we reach a leaf, i print the lowest element

#as there is nothing smaller than it

if T.isLeaf:

return T.item[0]

#recursive call calls the first child as that is where

#the smallest element is

return MinElement(T.child[0], d-1)

''' #4

========================================================

'''

def MaxElement(T,d):

#if d is 0, we found our depth, so return last element

if d == 0:

return T.item[-1]

#in case we reach a leaf, i return the largest element

if T.isLeaf:

return T.item[-1]

#recursive call calls the last child as that is where

#the largest element is

return MaxElement(T.child[-1], d-1)

''' #5

========================================================

'''

def NumNodes(T,d):

#reached desired depth, returns 1 as there is only 1 node

#that was visited

if d == 0:

return 1

if T.isLeaf:

return 0

count = 0

for i in range(len(T.child)):

#adds the number of children to count

count += NumNodes(T.child[i] ,d-1)

return count

''' #6

========================================================

'''

def PrintAtDepth(T,d):

if d == 0:

if T.item is not None:

return T.item

else:

L=[]

for i in range(len(T.child)):

L += PrintAtDepth(T.child[i], d-1)

return L

''' #7

========================================================

'''

def FullNodes(T):

if IsFull(T):

return 1

if T.isLeaf:

return 0

count = 0

for i in range(len(T.child)):

count += FullNodes(T.child[i])

return count

''' #8

========================================================

'''

def FullLeaves(T):

#checks if node of leaves is full, returns 1 if its full

if T.isLeaf:

if IsFull(T):

return 1

count = 0

#recursive call to check the leaves of the children to see

#if they are full

for i in range(len(T.child)):

count += FullLeaves(T.child[i])

return count

''' #9

========================================================

'''

def FindDepth(T,k):

#if k is in T.item, then the depth where it is at returns 0

if k in T.item:

return 0

#if the leaf is reached, K is not in the tree so it returns -1

if T.isLeaf:

return -1

for i in range(len(T.child)):

#using FindChild, we go to the desired child to find the

#element faster

depth = FindDepth(T.child[FindChild(T,k)],k)

#if depth is -1, then k is not in the tree

if depth == -1:

return -1

#returns the depth plus the current depth

else:

return depth+1

'''

========================================================

'''

L = [30, 50, 10, 20, 60, 70, 100, 40, 90, 80, 110, 120, 1, 11 , 3, 4, 5,105, 115, 200, 2, 45, 6]

T = BTree()

for i in L:

#print('Inserting',i)

Insert(T,i)

# Print(T)

List = []

#Insert(T,300)

#Insert(T,301)

PrintD(T,' ')

print('====The Tree above will be used for the following instructions====')

print('==================================================================')

print('1) Compute the height of the tree: ')

print()

print('-The Height of the tree is:' , Height(T))

print()

print('==================================================================')

print('2) Extract the items in the B-tree into a sorted list: ')

print()

print('-The elements of the tree are:', TreeToList(T))

print()

print('==================================================================')

print('3) Return the minimum element in the tree at a given depth d: ')

print()

print('-The smallest element at depth 0 is:', MinElement(T,0))

print()

print('-The smallest element at depth 1 is:', MinElement(T,1))

print()

print('-The smallest element at depth 2 is:', MinElement(T,2))

print()

print('==================================================================')

print('4) Return the maximum element in the tree at a given depth d: ')

print()

print('-The largest element at depth 0 is:', MaxElement(T,0))

print()

print('-The largest element at depth 1 is:', MaxElement(T,1))

print()

print('-The largest element at depth 2 is:', MaxElement(T,2))

print()

print('==================================================================')

print('5) Return the number of nodes in the tree at a given depth d: ')

print()

print('-The number of nodes at depth 0 is:',NumNodes(T,0))

print()

print('-The number of nodes at depth 1 is:',NumNodes(T,1))

print()

print('-The number of nodes at depth 2 is:',NumNodes(T,2))

print()

print('==================================================================')

print('6) Print all the items in the tree at a given depth d: ')

print()

print('-The elements in depth 0 are:', PrintAtDepth(T,0))

print()

print('-The elements in depth 1 are:', PrintAtDepth(T,1))

print()

print('-The elements in depth 2 are:', PrintAtDepth(T,2))

print()

print('==================================================================')

print('7) Return the number of nodes in the tree that are full: ')

print()

print('-Number of full nodes is:',FullNodes(T))

print()

print('---To test this method, I will insert the numbers 12 through 18---')

print('--------------------------to fill a node--------------------------')

Insert(T,12)

Insert(T,13)

Insert(T,14)

Insert(T,15)

Insert(T,16)

Insert(T,17)

Insert(T,18)

PrintD(T,' ')

print()

print('-The new number of full nodes is:',FullNodes(T))

print()

print('==================================================================')

print('8) Return the number of leaves in the tree that are full: ')

print()

print('-Number of full leaves is:',FullLeaves(T))

print()

print('---To test this method, I will insert the numbers 41+42 and 7+8---')

print('--------------------------to fill a node--------------------------')

Insert(T,41)

Insert(T,42)

Insert(T,7)

Insert(T,8)

PrintD(T,' ')

print()

print('-The new number of full leaves are:',FullNodes(T))

print()

print('==========================================================================')

print('9) Find the depth of the tree where key is found, (-1 if key is not found)')

print()

print('-The depth of the number 90 is:', FindDepth(T,90))

print()

print('-The depth of the number 115 is:', FindDepth(T,115))

print()

print('-The depth of the number 93 (which is not in the tree) is:', FindDepth(T,93))

print()

“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”

