Nancy Hernandez

Olac Fuentez

Lab 3 Report

**Introduction**

The objective of this lab is to learn how recursion works with Binary Search Trees

**Proposed Solution**

For number two I thought about checking whether the number is in the list or not. Then I thought about what to do when I do find the number and then what to do if I did find the number in the list. If the number happen to be less than the item then it would move on to the left side of the tree. If not then it would move on to the right side of the tree.

For number three I first checked if the list was sorted. Then I calculated the middle and the root. Then I did two recursive calls that would find the middle of each side (left and right) of the tree and find the root. And finally return the root.

For number four is first checked if the tree was empty then I set the index equal to a recursive call for the left side of the tree. I then did the same this but this time for the right side of the tree. And finally returned the index

For number five I first checked if the tree was empty then I checked if the depth was 0 and if it was then I would just print that item. Then if the left and right side of the tree are not empty then I would increase the depth by one, move to the left side and print that item. If the right side is empty and the left isn’t then I would increase the depth by one and go to the right side. If the left side is not empty and the right is then I would also increase the depth by one and move to the left side. Finally make two recursive calls one for the left and one for the right. And then just return the depth

**Setup**

To complete this lab I used an HP Pavilion x360 Convertible with a 2.71 GHz Intel® Core(TM) i5 processor.

**A screenshot of a computer screen

Description automatically generatedA screenshot of a computer screen

Description automatically generatedResults**

**Time Analysis**

|  |  |
| --- | --- |
| **Methods** | **Running time** |
| **2** | **O(n)** |
| **3** | **O(n)** |
| **4** | **O(n)** |
| **5** | **2n - 1** |

**Conclusion**

from this lab I learned how to use recursion when it comes to using binary search trees. I also learned how O(n) time is

**Appendix**

# Code to implement a B-tree

# Programmed by Olac Fuentes

# Last modified February 28, 2019

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 height(T):

if T.isLeaf:

return 0

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

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 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,T.item[i])

else:

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

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

print(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)

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)

PrintD(T,'')

#Print(T)

print('\n####################################')

SearchAndPrint(T,60)

SearchAndPrint(T,200)

SearchAndPrint(T,25)

SearchAndPrint(T,20)

print(height(T))

**Academic Dishonesty Statement**

I, Nancy Hernandez, was not involved in any copying from or providing information to another student, possessing unauthorized materials during a test, or falsifying data in laboratory reports. Neither did I participate in any type of collusion involving collaboration with another person to commit an academically dishonest act.