Lab 4 Report

Problem:

Understanding how a b-tree works through many different subproblems. Through three basic problems. Which are the full traversal of the b-tree. Finding the depth of the tree. Finding a certain element in the tree

Proposed solution:

The first problem was finding the height of the tree. Which is traversing the tree and adding one to a counter until you reach a leaf node. This runs in O(logn). Then we had to make a list from the b-tree. Unfortunately, I could not get that to work. This would run in O(nlogn). My idea consisted of traversing the b-tree until you reached a leaf and then insert the item array. The next problem is to find a minimum at a certain depth. You need to traverse the b-tree from the left-most side of the tree until you get to the desired depth then return item at index 0. This runs in O(logn). Then we were asked to find the largest at depth. Which means to traverse the right most until reaching the desired depth and returning the item at index -1. It runs in O(logn). The next problem was to print at depth which traverses the tree and gets the desired depth. Then you traverse the two sides of the tree and prints the items in each node of the depth. It runs in O(n). Then we look for the number of full leaves in the b-tree. This runs in O(n). Because you have to traverse the tree until it’s a leaf then check if the number of items is the same as the number of max items. This runs in O(n).The same could be said for the number of full nodes the difference is that this counts both full nodes and leaves. Runs in O(n). The next one is to count the number of items in a depth of the b-tree. You traverse the tree until you find the depth then return the length of the items array and add that for each item array in the desired depth. This runs in O(n). Finally, we have to find the depth of a certain item and return it. You have to add one to a counter and return it as soon as you find the item

Conclusion:

A b-tree traversal is a mix between the list traversal and an array traversal. It seems complicated at first but you can use recursion to make it simpler.

Appendix:

#Andres Arellanes Professor:Dr.Olac Fuentes TA:Anindita Nath CS2302 MW 1:30-2:50 03/24/19 Lab4:BTrees

# The purpose of this lab was to help us understand how b-trees work. How they are traversed and

# understand the three different types of problems

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 InsertLeaf(T,i):

T.item.append(i)

T.item.sort()

def height(T):

#returns the height of the b-tree

if T.isLeaf:

return 0

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

def minimumAtDepth(T,d):

# looks for the minimum in a b-tree

if d ==0:

return T.item[0]

if T.isLeaf:

print("item not found")

return None

else:

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

def maximumAtDepth(T,d):

# looks for the maximum number in the tree

if d==0:

return T.item[-1]

if T.isLeaf:

print("item not found")

return None

else:

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

def PrintAtDepth(T,d):

# Prints items in a tree at a certain depth

if d == 0:

print(T.item)

if T.isLeaf:

return -1

else:

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

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

def numFullLeaves(T):

#counts the number of full leaves

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

if T.isLeaf and len(T.item) == T.max\_items:

return 1

else:

return 1 + numFullLeaves(T.child[i])

return -1

def numFullNodes(T):

#counts the number of full nodes

if len(T.item) == T.max\_items:

return 1

if T.isLeaf:

return 0

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

return 1 + numFullNodes(T.child[i])

def NumsAtDepth(T,d):

# counts how many numbers are at a certain depth

if d == 0:

return len(T.item)

if T.isLeaf:

return -1

else:

count = 0

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

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

return count

def SearchDepth(T,k):

#Looks for a number and returns the depth where it is found

if k in T.item:

return 0

else:

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

return 1 + SearchDepth(T.child[i],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 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 IsFull(T):

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

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 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 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 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, 80,81,82, 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(" ")

print(height(T))

print(minimumAtDepth(T,1))

print(maximumAtDepth(T,1))

PrintAtDepth(T,1)

print(" ")

print(NumsAtDepth(T,2))

print(numFullNodes(T))

print(numFullLeaves(T))

print(SearchDepth(T,1))

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