Lab 4 report

Fernando De Santiago

University of Texas at El Paso

Lab 4 report

Lab #4 focuses on 9 different methods. 1. Being height of the B-tree. 2. Extracts the items in the B-tree into sorted lists(arrays).3 and 4 return the maximum and minimum at a certain depth. 5 returns the number of nodes or items at a given depth. 6 prints all the items at a given depth.7 and 8 returns the full nodes and full leafs. 9. Looks for a key k and returns the depth if found or -1.

# Method 1

Method 1 was already given and implemented in the code so I just reused it. It first checks if the first T is a leaf, however if it is not then it will recursively add 1 and go back to the method and check again till it reaches the end of the recursion

**Method 2**

Method 2 was a bit more challenging and required a list to be implemented. So first it checks if T is a leaf if it is then it returns all the elements in it. After it creates a empty list then create a for look in the range of the length of the child. It recursively adds the items into the list and then checks if the length of the items is ever less than the range of a counter then and if it is then it will append the item at the counter location into the list. At the end it will return the list.

**Method 3 and Method 4**

Method 3 I needed to implement some safety precautions incase the user uses inputs that aren’t allowed. I started by checking if the depth was 0 and if it is then it will return the first item in the list. Else if d < 0 then it will return the statement ’No Items in negative depth’. Else if d>height of the tree then it will return that depth is exceeded. And after checking everything then it will recursively run the method while starting at the first child and going down the depth. Then return the value. Method 4 is the same as number 3 but instead it will look at the last number in the child and the length.

**Method 5**

Method 5 uses the same precautions as 3 and 4 and then we get to the actual method. I create value to 0 and used a for loop for it being in the range of the length of the child. Inside the for loop as a recursive call that would add the total number of items as you went down the depth expected and adds it to the value holder. Then at the end of recursive calls it returns the value holder.

**Method 6**

Method 6 uses the same precautions as before and then begins the method. If T is not a leaf then it goes into a for loop in the range of the length of the child in T. It then prints the items and recursively goes down the tree while going to all the children and printing the items at the depth given.

**Method 7 and Method 8**

Method 7 uses a counter that is initialized to 0. From there we head to and if statement that has the following requirements of if the length items in T is equal to the max items allowed in T, and if it is then it adds one to the counter. Else if T is a leaf then it will return the counter. After those 2 little methods it then goes into a for loop in the range of the length of the child of T which then your counter will equal to the recursive call of the method for each child going down till it reaches the leaf. At the end the counter is returned. Method 8 is similar only that instead you need to check if T is a leaf and the length of the items of T is equal to the max number of items. If it is then it returns 1. Else then if enters a for loop in the range of the length of the child of T which then does the same thing as the counter of method 7 and returns the value for method 8.

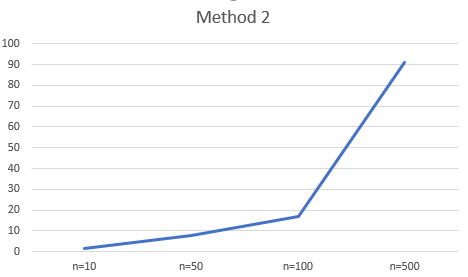
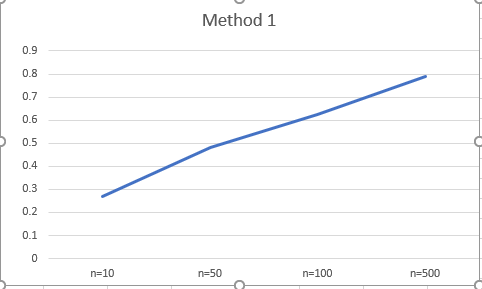
**Method 9**

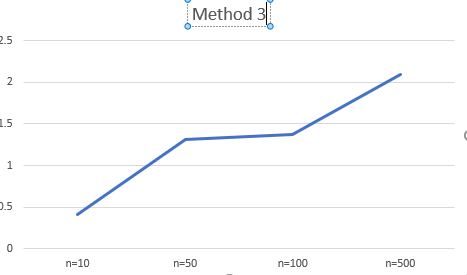
Method 9 uses the values of T and a key for the search. So first I create a temporary variable and use it as an index key. I go into a while loop when b<length of the items of T and the key>item of T at index b which in that while loop just makes the index go up. After it checks with a if statement that checks if the length of items of T is equal to b or if the key<item at b of T. In that if statement it checks with another if statement to see if it’s a leaf and if it is then it returns -1 else then if recursively will add the values returned to a temporary value. It then leads to another if statement and if the temporary value is equal to -1 then it returns -1 else then it will add 1 to f. When it exits the first if statement after the while loop if it didn’t end the if then it will just return 0.

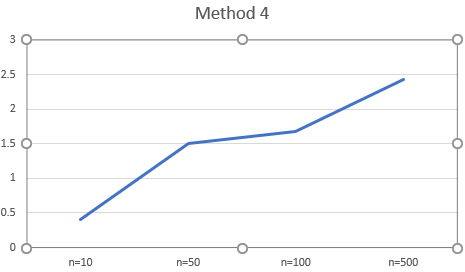
**Conclusion**

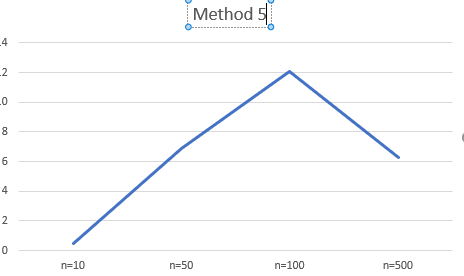
In the end I learned how to navigate a B-Tree while also learning how to do some basic operations. I had some trouble with the B-tree to sorted Array cause I couldn’t get it to display the middle numbers but with some trial and error along with rechecking my solution I was able to figure it out.

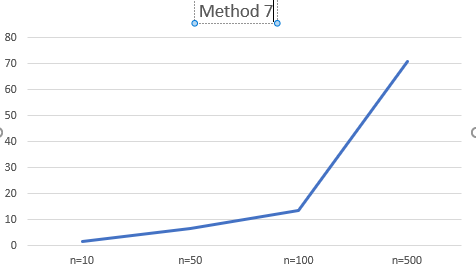
Graphs

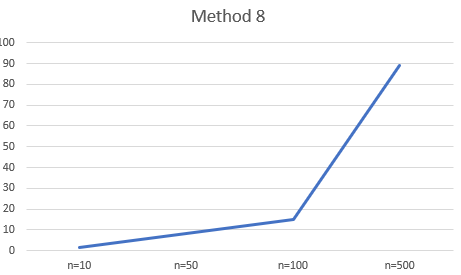


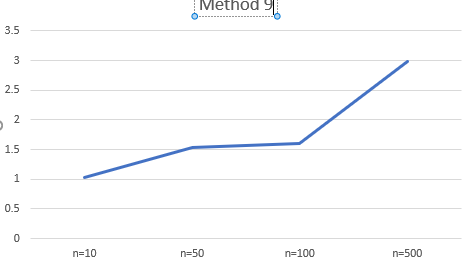


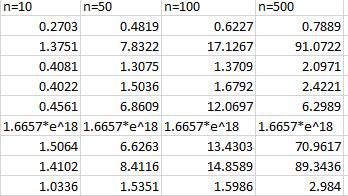












Source code

1# -\*- coding: utf-8 -\*-

"""

Created on Thu Mar 14 22:16:50 2019

@author: Fernando

"""

# Code to implement a B-tree

# Programmed by Olac Fuentes

# Last modified February 28, 2019

#CS2302

#Fernando De Santiago

#LAB3

#Olac Fuentes, Anindita Nath and Maliheh Zargaran

#last edited 3/15/19 21:48:00 PM

#Section M/W 10:30-11:50

#purpose: The purpose of this program is to help us learn how to navigate a B-tree

#while also teaching us how to right methods that require us to know how they work.

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))

#method 1

def height2(T):

if T.isLeaf:#if leaf it'll return 0

return 0

return 1 + height(T.child[0])#recursively goes down tree and counts

#method 2

def BTreeToSA(T):

if T.isLeaf: #if it's leaf it returns all items in that node

return T.item

Array=[]#creates empty array

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

Array+=BTreeToSA(T.child[i])#reursively goes down the tree until it reaches the leafs

if i < len(T.item):#if i is less than the list of items then it

Array.append(T.item[i])#will append the children nodes and root

return Array

#method 3

def minimum(T,d):

if d==0:

return T.item[0]#if depth is zero return the first item on the left

elif d<0:#if depth is less than 0 then it shouldn't expect a number

return 'No items in negative depth'

elif d>height2(T):#if depth is more than height then an error will occur so added this to stop that

return 'depth exceded'

return minimum(T.child[0],d-1)#goes all the way to the left and down the tree depending on the depth

#method 4

def maximum(T,d):

if d==0:#if depth is 0 return number on the far right of the list

return T.item[len(T.item)-1]

elif d<0:#if depth is less than zero than it shouldn't return a number

return 'No items in negative depth'

elif d>height2(T):#if depth is more than height then an error will occur so added this to stop it

return 'depth exceded'

return maximum(T.child[len(T.item)],d-1)#goes all the way to the right and down depending on the depth

#method 5

def NumItemsD(T,d):

if d==0:#if depth is 0 then return the length of the leaf

return len(T.item)

elif d < 0:#if depth is less than 0 then return -1 as a sign or error

return -1

elif T.isLeaf and d>0:#if it's a leaf and depth is greater than zero then an error would occur so return -1

return -1

elif d>height2(T):#if depth is more than the height then error would occur so return string

return 'depth exceded'

add=0#used to add the number

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

add+=NumItemsD(T.child[i],d-1)#recursively adds the numbers in the depth requested depth

return add

#method 6

def printItemsD(T,d):

if d==0:

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

print(T.item[i],' ')#if depth is 0 then it'll return the all the items in the node

elif d>height2(T):

print('depth exceded')#added error message

elif d<0:

print('depth can not be negative')#added error message

else:

if T.isLeaf is False:

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

printItemsD(T.child[i],d-1)#recursively prints items at given depth

#method 7

def FullNodes(T):

counter=0

if len(T.item)==T.max\_items:#if the length of the items is equal to the max add 1

counter+=1

elif T.isLeaf:#if leaf return 0

return counter

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

counter+=FullNodes(T.child[i])#recursively going through the child and checking if it meets max items

return counter

#method 8

def FullLeafs(T):

sum=0

if T.isLeaf and len(T.item) == T.max\_items:#checks if the leaf is a full node or not

return 1

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

sum+=FullLeafs(T.child[i])#recursively going through to find all the leafs and adding 1 to it

return sum

#method 9

def FindDepth(T,k):

b=0

while b<len(T.item) and k>T.item[b]:#going in a loop while using a temp counter and adding 1 to it

b+=1

if b==len(T.item) or k<T.item[b]:#checks to see if b is either the same as the length of the node or if the key is less than the item in the node at location b

if T.isLeaf:

return -1

else:

f=FindDepth(T.child[b],k)#recursively adding to f which will return the depth

if f==-1:

return -1

else:

return f+1

else:

return 0

print(height2(T))#1 works

print(BTreeToSA(T))#2 works

print(minimum(T,2))#3 works

print(maximum(T,2))#4 works

print(NumItemsD(T,0))#5 works

printItemsD(T,1)#6 works

print(FullNodes(T))#7 works

print(FullLeafs(T))#8 fix

print(FindDepth(T,-100))#9

“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.”  
 Fernando De Santiago