Michael Gonzalez

CS 2302 Data Structures

Olac Fuentes

03/25/2019

Lab 4 – B- Trees

For this lab, we have been tasked with implementing various methods for a B-Tree. 1. Compute the height of the tree, 2. Extract the items in the B-tree into a sorted list, 3. Return the minimum element in the tree at a given depth d, 4. Return the maximum element in the tree at a given depth d, 5. Return the number of nodes in the tree at a given depth d, 6. Print all the items in the tree at a given depth d, 7. Return the number of nodes in the tree that are full, 8. Return the number of leaves in the tree that are full, and 9. Given a key k, return the depth at which it is found in the tree, of -1 if k is not in the tree. The biggest challenge that I could see with this assignment is that I would have to figure out how to Traverse the B-tree while assuring that every child of a tree is considered or traversed for the specific method.

My solution for part 1 was to create a method called “height” that used “T” the Root of a B-Tree. B-Trees are known for being self-balancing so that means we can just traverse to any end of the B-Tree and the height would be the same. Here it is done by an if condition and it checks whether the current node is a leaf, if it is it returns 0, if not it makes a recursive call to “T.child[0]” and adds 1 for each layer it goes deeper into.

My solution for part 2 was to create a method called “MakeList” that used a “T” B-Tree. First the method goes into an if statement to check if the current node is a leaf, if it is a leaf then it appends each item in a list outside of the method. Else, it goes into a for loop that goes from 0 to the length of “T.item”. In this for loop it makes a recursive call to “T.child[i]” based on what I is from the for loop, then it appends the list with “T.item[i]”. out of this loop it makes another recursive call again with “T.child[len(T.item)]” to reach the right most side of the B-Tree. And after all calls have been made the List that was appended trough the method now has stored the B-Tree items in sorted order.

My solution for part 3 was to create a method called “MinElementAtDepthD” that used a B-Tree “T” and a number “d”. the method first goes into an if statement to check if d is greater than the height of the tree “height(T)” and if so, it returns -math.inf since it is impossible to reach a depth that does not exist. Next it goes to another if statement that checks if d is zero and if it is it return the first element of the current node “T.item[0]” if not it makes the recursive call to “T.child[0],d-1” to go one layer deeper and towards the left side of the tree since it is where the smallest items are stored. After every call has been made the method returns either the correct small item for the depth d or -math.inf.

My solution for part 4 was a method called “MaximumElementAtDepthD” that used a B-Tree “T” and a number “d”. the method first goes into a if statement to check if d is greater than the height of the tree “height(T)” and if so it return -math.inf since it is impossible to reach a depth that does not exist. Next it goes to another if statement that checks if d is zero and if it is it return the last element of the current node “T.item[len(T.item)-1]” if not it makes the recursive call to “T.child[len(T.item)],d-1” to go one layer deeper and towards the right side of the tree since it is where the larger items are stored. After every call has been made the method returns either the correct large item for the depth d or -math.inf.

My solution for part 5 was a method called “NumNodesAtDepthD” that used a B-Tree “T” and a number “d”. the method first creates a variable called counter and set it to 0. Then it goes into an if statement to check if d is greater than the height of the tree “height(T)” and if so it return -math.inf since it is impossible to reach a depth that does not exist. Next it goes to another if statement that checks if d is zero and if it is it returns the length of “T.item” if not it goes into a for loop that will go to each child of the root, here we add to counter the recursive call to “c,d-1” to go one layer deeper and keep track of the all the nodes at this specific depth. This way it adds all the nodes counted from all the possible children at the specified d. After every call has been made the method returns either the correct number of nodes at depth d or -math.inf.

My solution for part 6 was a method called “PrintAtDepthD” that used a B-Tree “T” and a number “d”. the method first checks the root to see if it is empty and if so return and finish. Then it goes into an if statement that checks if d is zero and if it is it runs a for loop from 0 to the length of the T.item and prints the “root.item[i]” based on what I is from the for loop. if not it goes into a for loop that will go to each child of the root, make the recursive call to “c,d-1” to go one layer deeper and keep track of the all the nodes at this specific depth. Prints all the nodes in order at the specified d. After every call has been made the method will have printed out all items at the given depth d.

My solution for part 7 was to create a method called “FullNodeNum” that used a “T” B-Tree. First the method creates a variable called tot and sets it to 0. Then it goes into an if statement, if T.max\_items == len(T.item) and T.isLeaf is True then it returns 1. If not it goes into an elif, where if T.max\_items == len(T.item) and T.isLeaf == False then it goes to a for loop where for every child in T.child it makes the recursive call to “c” and adds the result from that recursive call to tot or it goes to the else, where it goes to a for loop where for every child in T.child it makes the recursive call to “c” and adds the result from that recursive call to tot goes and after all recursive calls have been made the method returns the total number of full nodes in the B-Tree.

My solution for part 8 was to create a method called “FullNodeLeaf” that used a “T” B-Tree. First the method creates a variable called tot and sets it to 0. Then it goes into an if statement, if T.max\_items == len(T.item) and T.isLeaf is True then it returns 1. If not it goes to the else, where it goes to a for loop where for every child in T.child it makes the recursive call to “c” and adds the result from that recursive call to tot goes and after all recursive calls have been made the method returns the total number of full nodes in the B-Tree.

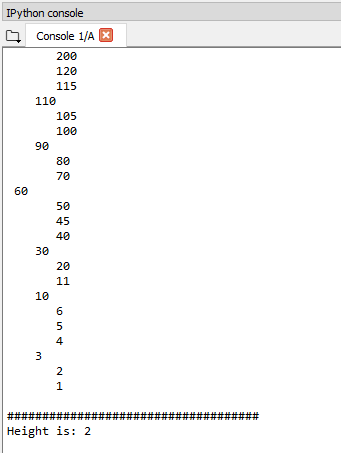
My solution for part 9 was to create a method called “SearchAndDepth” that used a B-Tree “T” and a number “k”. First the method goes into an if statement were checks if k is in T.item and so it returns 0, next it goes into another if statement the checks if it is a leaf and if so it returns negative one for we have not reached the desired depth and there is no more tree to traverse. Next it creates a variable called d which equals the recursive call “T.child[FindChild(T,k)],k”. If d equals to negative one, then it returns a negative one. if not it returns d plus one. After all recursive calls have been made the method returns the depth at which key K is found in the tree T. Item

Experiments/Results

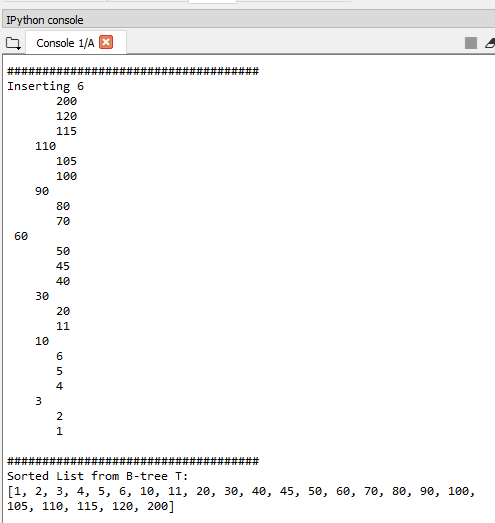
1. Compute the height of the tree

B-Tree

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

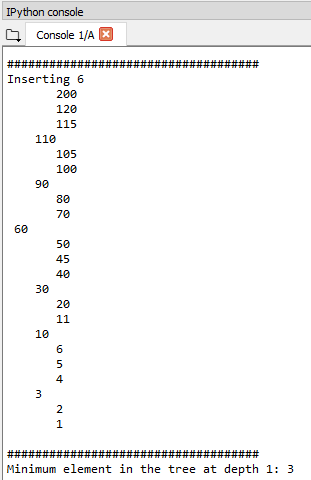


2. Extract the items in the B-tree into a sorted list.



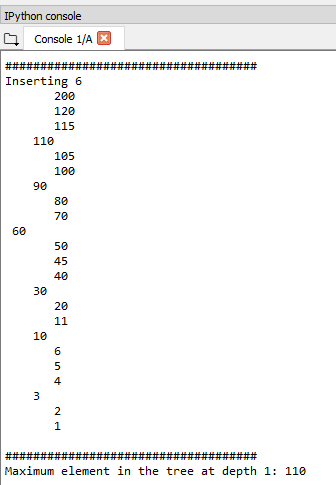
3. Return the minimum element in the tree at a given depth d.

Depth d = 1



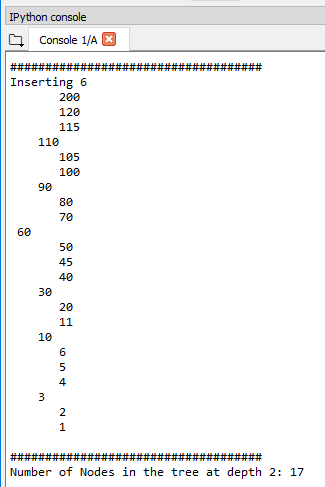
4. Return the maximum element in the tree at a given depth d.

Given Depth d = 1

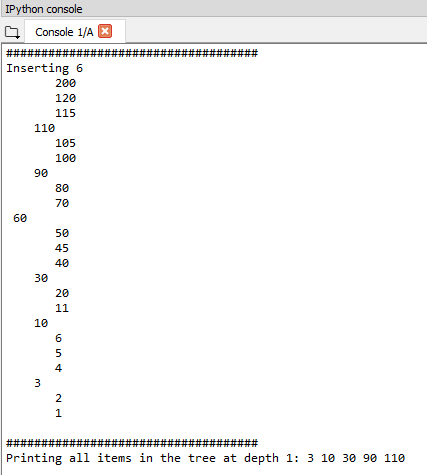


5. Return the number of nodes in the tree at a given depth d.

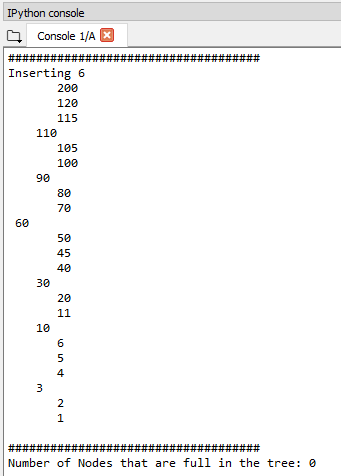
Given Depth d =2



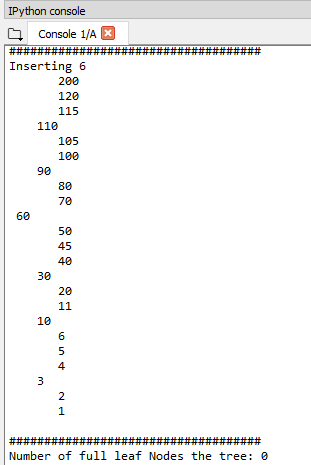
6. Print all the items in the tree at a given depth d.



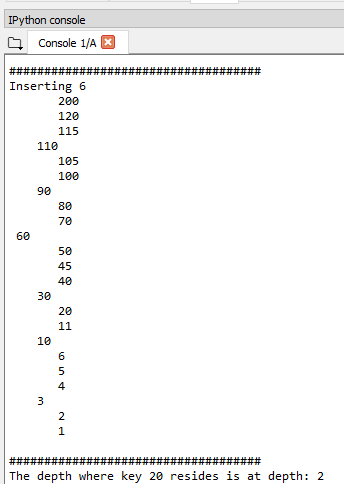
7. Return the number of nodes in the tree that are full.



8. Return the number of leaves in the tree that are full.



9. Given a key k, return the depth at which it is found in the tree, of -1 if k is not in the tree



Conclusion

In conclusion, I learned how to program various methods about B-Trees and how to traverse and modify the items inside the tree in a recursive manner. Aside from the methods, I have now a better appreciation of B-Trees and how efficient and different they are from Binary search trees. I have become more comfortable with coding in python than in lab 3 and I believe that I will be able to learn more from future labs to come.

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.

– Michael Gonzalez

Appendix – code

#Author: Michael Gonzalez

#Course: CS 2302 Data Structures

#Lab 4 B-Trees

#TA: Anindita Nath & Eduardo Lara

#Purpose:the purpose of this lab is to modify the code given in class and implement various methods.

#1. Compute the height of the tree

#2. Extract the items in the B-tree into a sorted list.

#3. Return the minimum element in the tree at a given depth d.

#4. Return the maximum element in the tree at a given depth d.

#5. Return the number of nodes in the tree at a given depth d.

#6. Print all the items in the tree at a given depth d.

#7. Return the number of nodes in the tree that are full.

#8. Return the number of leaves in the tree that are full.

#9. Given a key k, return the depth at which it is found in the tree, of -1 if k is not in the tree.

# Last modified March 24, 2019

import math as math

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 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)

def FD(T,k):

if k in T.item:

return 0

if T.isLeaf:

return -1

if k>T.item[-1]:

d = FD(T.child[-1],k)

else:

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

if k < T.item[i]:

d = FD(T.child[i],k)

if d == -1:

return -1

return d +1

#########################################################################

# LAB Start #

#########################################################################

#1 Compute the height of the tree

def height(T):

if T.isLeaf:

return 0

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

##########################################

#2. Extract the items in the B-tree into a sorted list.

def MakeList(T):

if T.isLeaf:

for t in T.item:

List.append(t)

else:

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

MakeList(T.child[i])

List.append(T.item[i])

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

##########################################

#3. Return the minimum element in the tree at a given depth d.

def MinElementAtDepthD(T,d):

if height(T) < d:

return -math.inf

if d==0:

return T.item[0]

else:

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

##########################################

#4. Return the maximum element in the tree at a given depth d

def MaximumElementAtDepthD(T,d):

if height(T) < d:

return -math.inf

if d==0:

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

else:

return MaximumElementAtDepthD(T.child[len(T.item)],d-1)

##########################################

#5. Return the number of nodes in the tree at a given depth d.

def NumNodesAtDepthD(T,d):

count = 0

if height(T) < d:

return -math.inf

if d==0:

return len(T.item)

else:

for c in T.child:

count += NumNodesAtDepthD(c,d-1)

return count

##########################################

#6. Print all the items in the tree at a given depth d.

def PrintAtDepthD(root, d):

if root is None:

return

if d == 0:

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

print(root.item[i],end=" ")

else:

for c in root.child:

PrintAtDepthD(c, d-1)

##########################################

#7. Return the number of nodes in the tree that are full.

def FullNodeNum(T):

tot = 0

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

return 1

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

for c in T.child:

tot += FullNodeNum(c)

return tot +1

else:

for c in T.child:

tot += FullNodeNum(c)

return tot

##########################################

#8. Return the number of leaves in the tree that are full

def FullNodeLeaf(T):

tot = 0

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

return 1

else:

for c in T.child:

tot += FullNodeLeaf(c)

return tot

##########################################

#9. Given a key k, return the depth at which it is found in the tree,

# of -1 if k is not in the tree

def SearchAndDepth(T,k):

if k in T.item:

return 0

if T.isLeaf:

return -1

d = SearchAndDepth(T.child[FindChild(T,k)],k)

if d == -1:

return -1

return d +1

#########################################

#L = [30, 50, 10, 20, 60, 70, 100, 40, 90, 80, 110, 120, 1, 11 , 3, 4, 5,105, 115, 200, 2, 45,12,13,14]

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

#L = [30, 50, 10, 20, 60, 70, 100, 40,21,22,23,24,25,26,27,19,18,17,11,16,15,14,13,12,9,8,7,6,5,4,3,28,29,31,32,33,34,35,41,42,80,90,95]

#L = [40,50,52,20,10,45,44,43,21,22,23,1,2,3,4,5,6,7,8,9,11,12,13,14,15,16,17,18,19,20,53,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,53,53,53,53,53,53,53,60,60,60,60,60,60,60,30,60,60,22,22,22,22,22,22,22,22,22,22,22,22]

T = BTree()

List = []

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)

#1 Compute the height of the tree

print("Height is:",end=" ")

print(height(T))

print()

#2. Extract the items in the B-tree into a sorted list.

MakeList(T)

print("Sorted List from B-tree T:")

print(List)

print()

#3. Return the minimum element in the tree at a given depth d.

print("Minimum element in the tree at depth 1:",end=" ")

print(MinElementAtDepthD(T,1))

print()

#4. Return the maximum element in the tree at a given depth d

print("Maximum element in the tree at depth 1:",end=" ")

print(MaximumElementAtDepthD(T,1))

print()

#5. Return the number of nodes in the tree at a given depth d.

print("Number of Nodes in the tree at depth 2:",end=" ")

print(NumNodesAtDepthD(T,2))

print()

#6. Print all the items in the tree at a given depth d.

print("Printing all items in the tree at depth 1:",end=" ")

PrintAtDepthD(T,1)

print()

print()

#7. Return the number of nodes in the tree that are full.

print("Number of Nodes that are full in the tree:",end=" ")

print(FullNodeNum (T))

print()

#8. Return the number of leaves in the tree that are full

print("Number of full leaf Nodes the tree:",end=" ")

print(FullNodeLeaf(T))

print()

#9. Given a key k, return the depth at which it is found in the tree,

#of -1 if k is not in the tree

print("The depth where key 20 resides is at depth:",end=" ")

print(SearchAndDepth(T,20))

print()