Marco Valerio

CS 2302

Olac Fuentes

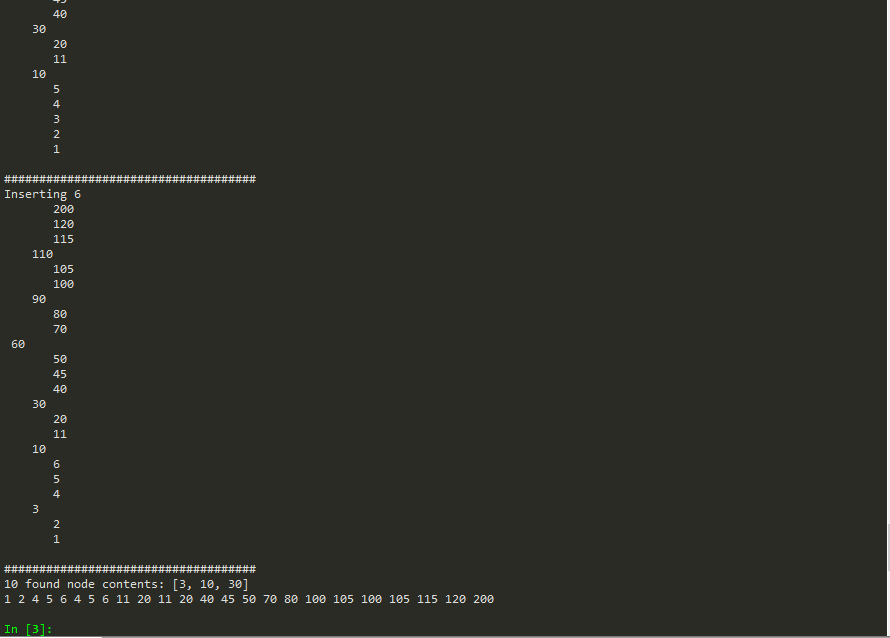
Lab 4 Report

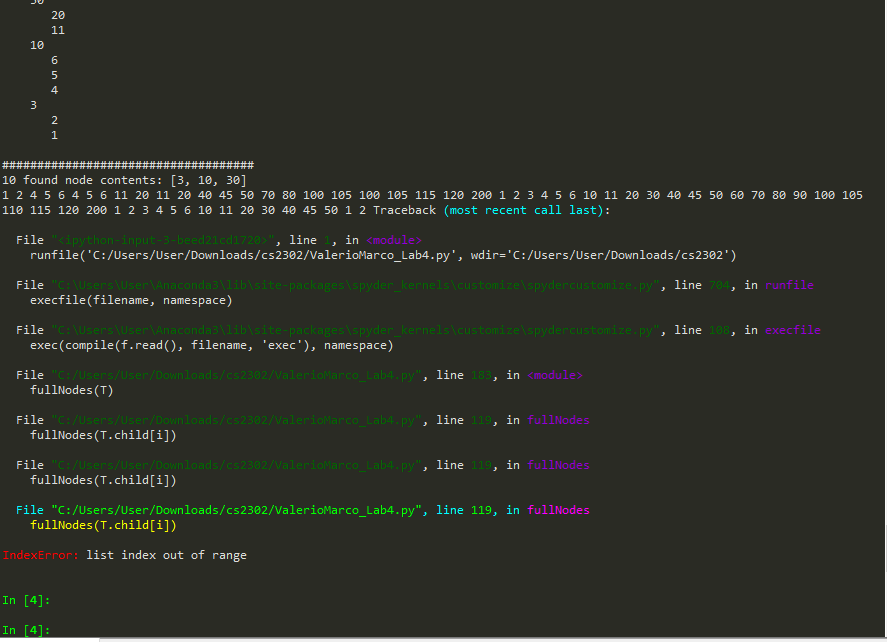
03/19/2019

***Introduction:*** The focus of lab 4 was to continue practice with binary trees and B-trees. To my understanding B-trees are similar to binary trees, but each child is a different array of items; they are like data structures made of data structures. Much like the previous lab, this lab tests the students understanding and implementation of b-trees in regard to maneuvering, locating and extracting data.

***Proposed Solution:*** A majority of my methods in this lab are attempts because I was unable to get them to properly work, but I believe the attempt is close. Method 1, to compute the height of the tree I intended to run through each path and use a counter to keep track of each level it has passed. For the methods that run through the b-tree and returns the contents at that depth whether it be minimum, maximum and all keys. I basically attempted to run through each level until ‘d’ is 0, ‘d’ was the desired depth and it would search for either item[0] for the min number or item[-1] for the max number, or it would go through each item with a for loop checking every item[i]. My attempts to check whether if the nodes or leaves were full were to run through each path and check the nodes and leaves using the ‘isFull’ method.

***Experimental Results:*** For the experiments regarding depth, I tested each depth of the tree and an out of bounds number, this method did not work it kept giving me an ‘index out of range’ message instead of the expected result which was the contents of the tree. Most of my methods were not functional, so I was unable to conduct meaningful experiments.





***Conclusion:*** B-trees are much more difficult to work with than typical binary trees. In the previous lab my understanding of the topic was better than this one. Most likely because I missed the day which this data structure was introduced. I attempted to use similar algorithms that I knew worked for binary trees.

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

"""

Prof: Olac Fuentes

Due: 03/15/19

Section MW 10:30-11:50

Lab 4

@author: Marco Valerio

"""

# 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 DepthFinder(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 DepthFinder(T.child[FindChild(T,k)],k)

"""

Print the elemnts at depth 'd' ,uses the for loop to get every possible child until reaches the length of the tree

"""

def printAt(T,d):

if d == 0:

if T.isLeaf:

for t in T.item:

print(t,end=' ')

else:

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

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

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

def fullNodes(T):

counter = 0

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

if IsFull(T):

counter = counter + 1

Print(T)

fullNodes(T.child[i])

print(counter)

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)

"""

This method should go through each level until d is 0 then print the right most item

"""

def maxElm(T,d):

if T is None:

return None

if d==0:

print(T.item[-1])

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

"""

This method should go through each level until d is 0 then print the left most item

"""

def minElm(T,d):

if T is None:

return None

if d == 0:

print(T.item[0])

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

c = 0

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)

printAt(T,2)

DepthFinder(T,1)

***Academic Statement:*** """

I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, preformed 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."""