PAOLA TERRAZAS

CS2302

B-TREES

DR. FUENTES

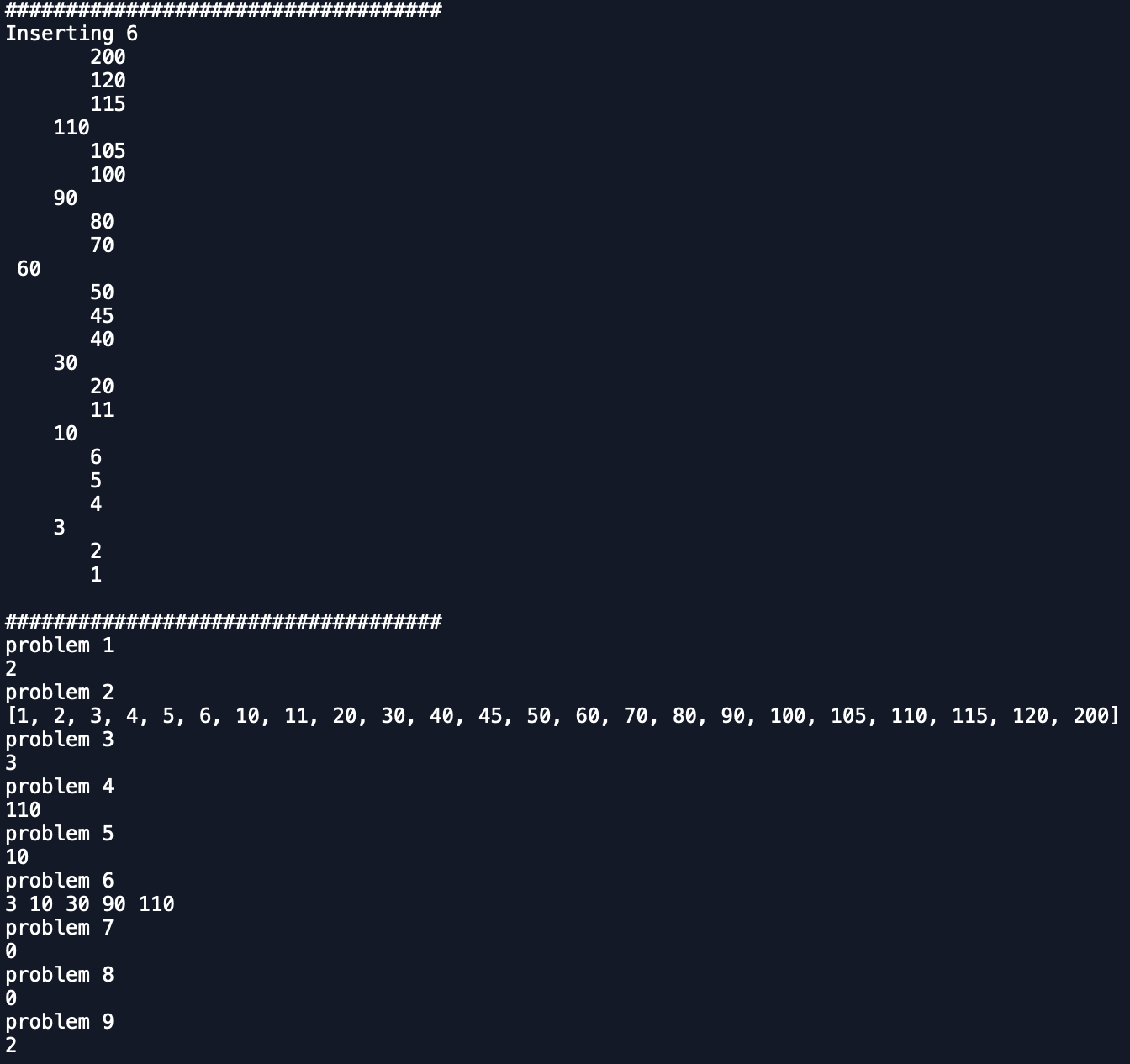
TA – ANINDITA NATH

3.25.2019

LAB 4 REPORT

For this report, we were asked to answer nine problems on B-Trees. Problem number one asked to compute the height of the tree, which was a very simple computation. I created an if statement if T.isLeft return 0, else return 1 plus the recursive call of the method at T.child[0]. We add one because iterator begins counting at 0. Problem number two asked to extract the items in the B-tree into a sorted list. If T is a leaf, it returns that item, else I created an empty list named List for iteratively add every child. For problem number three and four, we needed to return the minimum and maximum elements in the tree at a given depth d. I used the same algorithm for both of them, for smallest we access it using position 0 and for largest we access it using position -1. For problem number five, we needed to return the number of nodes in the tree at the given depth d, I used recursion to go through the nodes at the specific depth subtracting one until reaching the base case of d is equal to 0. For the sixth problem, we were asked to print all the items in the tree at a given depth d. I created a loop that goes through every child. For problems seven and eight, we needed to return the number of nodes and leaves in the tree that are full. I used a similar algorithm for both, creating a count variable that iterates through the tree, adding to the count. For the last problem, we were asked to return the depth at which k is found in the tree, -1 if k is not in the tree. I created an if statement that checks if the k is the current node then you return 0, two other if statements to check if the key is greater than or less than the last item. If the key is not found, you return -1.

Below are my results.



In this lab, I learned about B-Trees and all of its functions. I learned how to get minimum and maximum values in a B-Tree, print all the values, print full leaves and so much more.

#SOURCE CODE

# PAOLA TERRAZAS

# CS2302

# BTREE

# DR. FUENTES

# TA - ANINDITA NATH

# 3.24.2019

# LEARNING TO WORK WITH B TREES AND ALL OF ITS FUNCTIONS

import 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 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)

def Smallest(T):

if T.isLeaf:

return T.item[0]

return Smallest(T.child[0])

def NumItems(T):

if T.isLeaf:

return len(T.item)

count = len(T.item)

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

count += NumItems(T.child[i])

return count

def Largest(T):

if T.isLeaf:

return T.item[-1]

return Largest(T.child[-1].child[-1])

# 1) compute the height of tree

def TreeHeight(T):

if T.isLeaf:

return 0

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

# 2) extract items in tree into sorted list

def TreetoList(T):

if T.isLeaf:

return T.item

else:

List = []

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

List = List + TreetoList(T.child[i]) + [T.item[i]]

return List + TreetoList(T.child[-1])

# 3) min element in tree at given depth d

def SmallestAtDepthD(T,d):

if d == 0:

return T.item[0]

if d > 0 and T.isLeaf:

return -math.inf

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

# 4) max element in tree at given depth d

def LargestAtDepthD(T,d):

if d == 0:

return T.item[-1]

if d > 0 and T.isLeaf:

return -math.inf

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

# 5) return the number of nodes in tree at given depth d

def NodesAtDepthD(T,d):

if d==0:

return len(T.item)

if not T.isLeaf:

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

# 6) print all items in tree at given depth d

def PrintAtDepth(T,d):

if d==0:

for i in T.item:

print(i,end=' ')

if not T.isLeaf:

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

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

# 7) return number of nodes in tree that are full

def fullNodes(T):

count = 0

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

count = count + 1

return count

# 8) return number of leaves in tree that are full

def fullLeaves(T):

count = 0

if not T.isLeaf:

return 0

else:

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

count = count + 1

return count

# 9) return the depth at which k is found in the tree, -1 if k is not in tree

def FindDepthK(T, k):

if k in T.item:

return 0

depth = 0

if k > T.item[len(T.item)-1]:

depth = FindDepthK(T.child[len(T.item)], k)

elif k < T.item[0]:

depth = FindDepthK(T.child[0], k)

if depth >= 0:

return 1 + depth

else:

return -1

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("LargestAtDepthD")

#print(LargestAtDepthD(T,2))

#print(height(T))

#print(Largest(T))

#print(Smallest(T))

#print("NumItems")

#print(NumItems(T))

#print(T.item[0])

#print(T.child[0].item[2])

#print(len(T.child[1].item))

#print(T.child[0].child[1].item)

#print(T.child[0].child[2].item)

print("problem 1")

print(TreeHeight(T))

print("problem 2")

print(TreetoList(T))

print("problem 3")

print(SmallestAtDepthD(T, 1))

print("problem 4")

print(LargestAtDepthD(T, 1))

print("problem 5")

print(NodesAtDepthD(T,2))

print("problem 6")

PrintAtDepth(T,1)

print()

print("problem 7")

print(fullNodes(T))

print("problem 8")

print(fullLeaves(T))

print("problem 9")

print(FindDepthK(T, 70))