**CS2302 - Data Structures**

**Spring 2019**

**Lab Report 4**

March 25, 2019

Sebastian Gomez

**Introduction:**

Using the given code for performing and creating B-Trees, elaborate 9 different methods to do different operations that works with the given code.

**Implementation:**

1. For problem number 1, in order to get the height of the tree, I needed to check the levels of such tree. If it was only a leaf, then automatically the height of the tree was zero, if not, it could check any child because the children are balanced, so I added one and called the method again with parameter the last child to continue to add until there is no more levels.
2. For problem number 2, I had to put the items of the B-Tree to a list, that needed to be sorted. For this problem I created an empty list, I checked if the given tree was a leaf, if yes, then it would populate the list with the items of the B-tree, if not, then it would go to the left child and add its items, then I added the root, and then went to the right child, but for this one I could populate the list with every item of the B-tree, it did return a sorted list but not with every element of the list.
3. For problem number 3, we were asked to find the minimum element at a given depth, so for this problem I created a method with 3 parameters. The first one was the Tree, the second one was the expected depth, and I added a third one to check from the beginning if the depth given was valid or not which was the height of the tree. As said, first I checked if the depth was bigger than the height, if yes, then automatically returned -1 to indicate that is not valid, if it was less then I checked if the depth was zero, if yes, then it would return the item at index zero. If the depth was greater than zero, then it would go to the left child, because we know that in there is where the minimum element is, and reduced the depth by one so it eventually it would be zero in order for it to return the minimum element at the given depth.
4. For problem number 4, which was to find the maximum element, I basically did the same as for problem number 3, the difference was that instead of going to the left child, I would go to the right, and instead of returning the item at index zero, I returned the item at index negative one in order to return the last element, therefore, the maximum element at given depth.
5. For problem number 5, we were asked to return the number of nodes at given depth. For this problem, I still used the height as a parameter to check if the depth was valid or not, if it was and if it was zero, then I would just return the length of the tree, if the depth was bigger, I just went to the left child and decremented the depth by one, and added the result of the length of the right child and also decremented the depth by one. For this method, for depth zero and one, it worked, but for depth two only returned the number of nodes at the left child.
6. For problem number 6, the task was to print at a given depth. I used the same logic of problem number 5, and still used the height as my verifier. When the depth was zero or reached zero, I created a for loop to print the elements of the tree at the given depth, if it was bigger than zero, it would reach the else statement where it would go to another for loop which was from zero to the length of the child, and inside the loop, I called the method again and again until it was one less than the length of the child, I did so in order to print at depth one and two, or any other depths if added more items.
7. For problem number 7, we were asked to create a method to know what nodes of the tree were full. In other words, if a node of the tree had 5 elements, then it was full because if you added one more, it would split it and balance the tree. For this method, we only cared about the nodes who were not leaves, so if it was a leaf, it returned zero, if not, I created and if statement, on which I used the given method IsFull(T), that if it got in, it would add one and then add the method itself with parameter the left child plus the method itself with parameter the right child. For this case I did not know why it always returned zero.
8. For problem number 8 we were asked to find what leaves were full, so I started my method by checking if it was a leaf, if it was then it would check if the leaf was full using the method IsFull, if yes, it would add one and add when the method called the left child and the right child. As well, I always got zero as an answer and I do not know why because I do not know what is wrong with the code.
9. For the last problem we needed to return the depth at which the given key was found. I started by checking if it was a leaf, if the key was in the tree then it would return zero because a leaf has no more levels. Then if it was not a leaf, it would check if the key was in the root, if not it would check if the key was less than, if yes it would go to the left child, if not, it would go to the right child. For both I assigned a variable d, representing depth that it would increment every time the method called itself, if it reached negative one, it meant that the key was not found, if it was found, I returned d plus one. The problem that I found with these problem was at depth 2 because with depth zero and one it returned all of them correctly, but for depth two it only worked for the left child and right child when the key was at the at the most left side or the most right side.

**Running Times:**

Problem 1 – O(Log(n))

Problem 2 – O(n)

Problem 3 – O(Log(n))

Problem 4 – O(Log(n))

Problem 5 – O(n)

Problem 6 – O(n)

Problem 7 – O(n)

Problem 8 – O(n)

Problem 9 – O(Log(n))

**Conclusions:**

As said in class, B-Tree is the hardest subject of the semester and it fact it was a little bit difficult to code because even though it only has a left and right child, by having several items in it, and that it has the ability to split and more factors that make it unique, it was a little bit harder to know how to access everything, I hope that I have learned enough for the exam.

**I, Sebastian Gomez, certify that this project is entirely my own work, I wrote, debugged, and tested the code being presented, performed 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.**

**Appendix:**

"""

@author: Sebastian Gomez

Course: Data Structure 2302

Assignment: Lab 4

Instructor: Olac Fuentes

T.A: Anindita Nath and Maliheh Zargaran

Purpose: Create operations for a B-Tree

"""

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

else:

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

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 BTToList(T):

sort = []

if T.isLeaf:

sort = [T.item]

else:

sort = BTToList(T.child[0]) + [T.item] + BTToList(T.child[-1])

return sort

def MinElementAtDepth(T,d,h):

if d > h:

return -1

if d == 0:

return T.item[0]

else:

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

def MaxElementAtDepth(T,d,h):

if d > h:

return -1

if d == 0:

return T.item[-1]

else:

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

def NumOfNodesAtDepth(T,d,h):

if d > h:

return -1

if d == 0:

return len(T.item)

else:

num = NumOfNodesAtDepth(T.child[0],d-1,h) + NumOfNodesAtDepth(T.child[-1],d-1,h)

return num

def PrintAtDepth(T,d,h):

if d > h:

return -1

if d == 0:

for t in T.item:

print(t,end=' ')

else:

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

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

def NumOfNodesFull(T):

if T.isLeaf:

return 0

else:

if IsFull(T):

return 1 + NumOfNodesFull(T.child[0]) + NumOfNodesFull(T.child[-1])

else:

return 0

def NumOfLeavesFull(T):

if T.isLeaf:

if IsFull(T):

return 1 + NumOfLeavesFull(T.child[0]) + NumOfLeavesFull(T.child[-1])

else:

return 0

else:

return 0

def FindAtDepth(T,k):

if T.isLeaf:

if k in T.item:

return 0

else:

return -1

if k in T.item:

return 0

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

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

else:

d = FindAtDepth(T.child[0],k)

if d == -1:

return -1

else:

return d + 1

L = [30, 50, 10, 20, 60, 70, 100, 40, 90, 80, 110, 120,123,122,121,201,202,205,206,53, 1, 11 , 3, 4, 5,105, 115, 200, 2, 45, 6,51,52,116]

T = BTree()

for i in L:

print('Inserting',i)

Insert(T,i)

PrintD(T,'')

Print(T)

print('\n####################################')

h = height(T)

print(h)

sortedList = BTToList(T)

i = 0

while i < len(sortedList):

print(sortedList[i], end = ' ')

i +=1

print()

print('Minimum Element at Depth: ',MinElementAtDepth(T,1,h))

print('Max Element at Depth: ',MaxElementAtDepth(T,1,h))

print('Number Of Leaves Full: ',NumOfLeavesFull(T))

print('Number of Nodes Full: ',NumOfNodesFull(T))

print('Number of Nodes at Depth: ',NumOfNodesAtDepth(T,1,h))

PrintAtDepth(T,1,h)

print()

print(FindAtDepth(T,202))