Nancy Hernandez

Olac Fuentez

Lab 4 Report

**Introduction**

The point of this lab was to gain experience with B-Trees while using recursion.

**Proposed Solution**

For problem one I decided to just use one of the given methods to calculate the height of the B-tree.

For number two the question asks to create an array from a tree. For this I thought about doing an if and an else statement. If the if statement I will check if it is a leaf. If it is then there will be a for loop that appends each item into an array. If it not a leaf, then it will go into the else statement where it will go into a loop that goes to every child of the root of the tree and goes into a recursive call. Also, inside this loop I append whatever that item is to the array. After the loop there is another recursive call that checks for the last child of the root as it is skipped.

Number three asks to find the smallest number in the tree at a given depth and I accomplish this by checking if it is a leaf first. If it is then I return the first item. If the given depth is 0 then I also return the first item. Else I return a recursive call with the first child and d – 1.

Number four asks to find the largest number in the tree at a given depth. I did this by checking if it is a leaf. If it is then I return the item of the last index. If d is 0 then I also return the item in the last index. Else I return a recursive call with the last child and d – 1.

To find the number of nodes at a certain depth I used an if and else statements. If the depth is 0 then I return 1. Else I create a for loop that increaser a counter by each recursive call. Then I just return the counter.

To print all the nodes at a certain depth I used to if statements. The first checks if the depth is 0. And if it is then it will go into a for loop that goes to each item of the tree and prints it. The second if statement also has a for loop and goes through the length of that depth and has a recursive call that goes through each child and decreases the depth by one.

To count all of the full nodes I made to if statements. The first checks if it is not a leaf. If it is then it goes into a for loop that goes to every child and increases the count every recursive call. The second, checks if the length of the item is equal to the number of max items and increases the count by one. Finally, just returns the count.

The eighth question asks to count the number of full leafs. There is an if statement and an else statement. The if statement checks if the length of the item is equal to the max items. The else statement has a for loop that goes to every child in the tree and increases the count with every recursive call.

Finally, the last question asks to find the depth of a certain number. To do this I used for different if statements and one else statement. The first if checks if the number is the root. If it is then it returns 0. The second if checks if it is a leaf. It is then it also returns 0. The third if checks if the number is more that the last item. If it is then it sets the depth equal to a recursive call.

The else statement contains a for loop that goes through the items. Inside there is an if statement that checks if the number is less than the items. If it is then it will set the depth equal to a recursive call. Finally, the last if checks if the depth is equal to -1 it will return -1. Then I final return statement for depth + 1.

**Setup**

To complete this lab I used an HP Pavilion x360 Convertible with a 2.71 GHz Intel® Core(TM) i5 processor.

**Results**

***Max items:*** 3

***My List:*** 30, 50, 10, 20, 60, 70, 110, 120, 1, 11, 3, 4, 5, 105, 115, 200, 2, 45, 6

***Max element:*** 200

***Min element:*** 1

***Height:*** 2

|  |  |  |  |
| --- | --- | --- | --- |
| **Depths** | **Full Nodes** | **Full Leafs** | **Elements at** |
| 0 | 0 | 0 | 30 |
| 1 | 0 | 0 | 3, 10, 60, 110 |
| 2 | 2 | 2 | 1, 2, 4, 5, 6, 11, 20, 45, 50, 70, 105, 115, 120, 200 |

**A screenshot of a computer screen

Description automatically generatedA screenshot of a computer screen

Description automatically generated**

**Time Analysis**

|  |  |
| --- | --- |
| **Operations** | **Time** |
| Sorting | O(log M/2 N) |
| Find | O(log N) |
| Insert | O((M/log M)\*log N) |
| Delete | O((M/log M)\*log N) |

**Conclusion**

From this lab I learned how recursion can be incorporated when it comes to using B-Tress. I also gained more experience with B-Trees and understand how they work.

**Appendix**

class BTree(object):

# Constructor

def \_\_init\_\_(self, item=[], child=[], isLeaf=True, max\_items=2):

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)

**Academic Dishonesty Statement**

I, Nancy Hernandez, was not involved in any copying from or providing information to another student, possessing unauthorized materials during a test, or falsifying data in laboratory reports. Neither did I participate in any type of collusion involving collaboration with another person to commit an academically dishonest act.