CS 2302

Lab 4 Report

Author: David Ayala

ID: 80617838

Instructor: Olac Fuentes

TA: Anindita Nath

**Introduction**

The purpose of this lab was to Compute the height of the tree, Extract the items in the B-tree into a sorted list. Return the minimum element in the tree at a given depth d. Return the maximum element in the tree at a given depth d. Return the number of nodes in the tree at a given depth d. Print all the items in the tree at a given depth d. Return the number of nodes in the tree that are full. Return the number of leaves in the tree that are full. Given a key k, return the depth at which it is found in the tree, of -1 if k is not in the tree.

**Proposed Solution & Design Implementation**

**Height Of Tree**

There’s not much to say about this except that professor Fuentes gave us this one on the code that he posts on his website.

**Extract Items To Sorted List**

In this one I understood that the goal was to extract the values from the B-Tree into a sorted list. So I knew that I would have transverse the list and then extracted the value. Since the B-tree already sorts the values all I had to do was make sure that I transverse the in the correct order in order to accomplish this task.

**Smallest At Depth**

So since I knew that I was looking for the smallest value at a certain depth I knew that all I had to do was transverse the B-tree to the given depth. Then from there it was all a matter of know which side to go to because the B-tree is already structed to have the values sorted.

**Largest At Depth**

So since I knew that I was looking for the largest value at a certain depth I knew that all I had to do was transverse the B-tree to the given depth. Then from there it was all a matter of know which side to go to because the B-tree is already structed to have the values sorted.

**Nodes At Depth**

So I knew that I was looks for nodes at the depth given, I also had to traverse the B-tree until the given depth. I need a statement to find each node as I transverse the B-tree. Once I find a node, I would then have add 1 to the counter and once there were no more node I would then return counter.

**Print At Depth**

So since I knew that I was going to print the values at a certain depth I knew that all I had to do was transverse the B-tree to the given depth. Then from there it was all a matter of going traversing the B-tree and at each value simply printing out the values.

**Number Of Full Nodes**

I first had to understand what a full node was and then from there I knew that I would have to transverse the B-tree. While traversing the B-tree I would need a statement that would check to see if there was a full node and if so add 1 to counter. Once I go through the whole tree, I return count.

**Number Of Full Leaves**

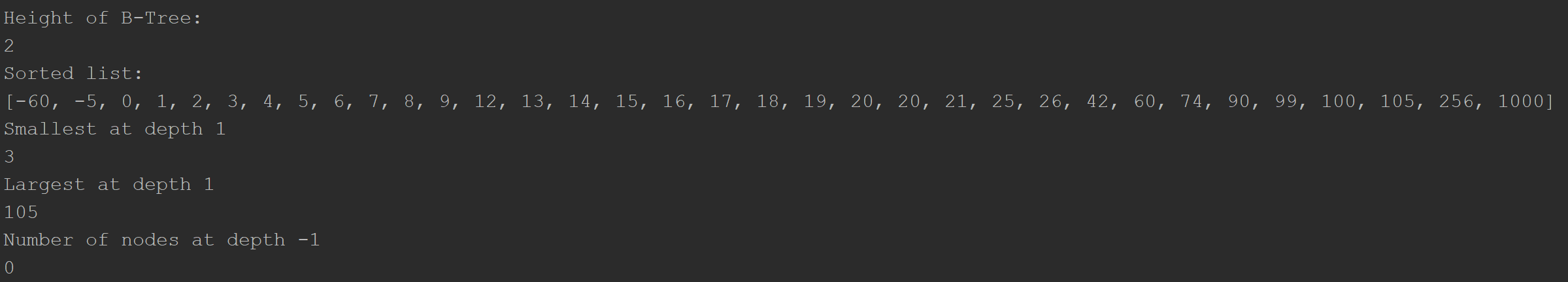
I understood that I would need to traverse the B-tree until I reach the leaves and once there I would have to compare the number of leaves to see if it was full or not. If the leaf was full I would add 1 to count and once I went through the whole tree I would return count.

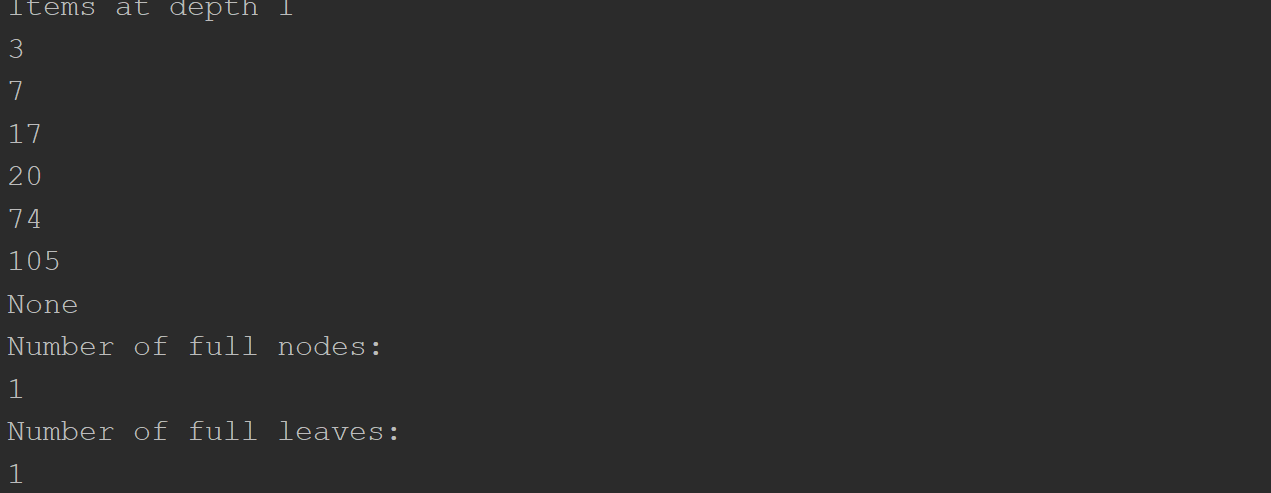
**Find Depth Of Element**

So I knew that I was looks for the depth of a given value, so I would have to traverse the B-tree and while I was doing that compare the values in the tree to value that was given, add one to counter if the value was not there. If I found the value, I would then return count and if the value was never found then I would return -1.

**Experimental Results**

|  |  |
| --- | --- |
| **List made into tree size** | **Runtime** |
| [] (zero) | 0.0 |
| [1, 2, 3, 4, 5, 6] (six) | 0.0 |
| [-1, -2, -3, -4, -5, -6](six) | 0.0 |
| [30, 50, 10, 20, 60, 70, 100, 40, 90, 80, 110, 120, 1, 11 , 3, 4, 5, 105, 115, 200, 2, 45, 6] (twenty-three) | 0.004016876220703125 |
| [42, 3, 17, 19, 7, 25, 14, 8, 5, 16, 15, 0, 20, 4, 18, 13, 9, 20, -60, 12, 21, 2, 1, -5, 6] (twenty-four) | 0.0050220489501953125 |
| [42, 3, 17, 19, 7, 25, 14, 8, 5, 16, 15, 0, 20, 4, 18, 13, 9, 20, -60, 12, 21, 2, 1, -5, 6, 90, 105, 256, 1000, 60, 74, 99, 100, 26](thirty-four) | 0.008014440536499023 |





**Conclusion**

This lab allowed us to learn how to transverse a B-tree and how to obtain certain values in the tree. Basically, this whole lab was about B-trees and the different things we could do with them particularly the questions that he asked us to do (Compute the height of the tree, Extract the items in the B-tree into a sorted list. Return the minimum element in the tree at a given depth d. Return the maximum element in the tree at a given depth d. Return the number of nodes in the tree at a given depth d. Print all the items in the tree at a given depth d. Return the number of nodes in the tree that are full. Return the number of leaves in the tree that are full. Given a key k, return the depth at which it is found in the tree, of -1 if k is not in the tree.) This lab also will give us some ideas on how to do the next lab.

**Appendix**

# Course:CS 2302 MW 1:30-2:50, Author:David Ayala  
# Assignment:Lab #4, Instructor: Olac Fuentes  
# Teaching Assistant: Maliheh Zargaran, Date of last Modification: 3/24/2019  
# Purpose of program: Compute the height of the tree, Extract the items in the B-tree into a sorted list.  
# Return the minimum element in the tree at a given depth d.  
# Return the maximum element in the tree at a given depth d.  
# Return the number of nodes in the tree at a given depth d.  
# Print all the items in the tree at a given depth d.  
# Return the number of nodes in the tree that are full.  
# Return the number of leaves in the tree that are full.  
# Given a key k, return the depth at which it is found in the tree, of -1 if k is not in the tree.  
import time  
  
start = time.time()  
  
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 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 HeightOfTree(T):  
 if T.isLeaf:  
 return 0  
 return 1 + HeightOfTree(T.child[-1])  
  
def ExtractItemsToSOrtedList(T, List):  
 if T.item is None:  
 return None  
 else:  
 if T.isLeaf:  
 for temp in T.item:  
 List += [temp]  
 else:  
 for i in range(len(T.item)):  
 ExtractItemsToSOrtedList(T.child[i], List)  
 List += [T.item[i]]  
 ExtractItemsToSOrtedList(T.child[len(T.item)], List)  
  
  
def SmallestAtDepth(T, d):  
 if not T.item:  
 return None  
 if d == 0:  
 return T.item[0]  
 if T.isLeaf or d < 0:  
 return None  
 return SmallestAtDepth(T.child[0], d - 1)  
  
def LargestAtDepth(T, d):  
 if not T.item:  
 return None  
 elif d == 0:  
 return T.item[-1]  
 elif T.isLeaf:  
 return None  
 elif d < 0:  
 return None  
 return LargestAtDepth(T.child[-1], d - 1)  
  
def NodesAtDepth(T, depth):  
 if not T.item:  
 return 0  
 elif depth == 0:  
 return 1  
 elif T.isLeaf:  
 return 0  
 elif depth < 0:  
 return 0  
 count = 0  
 for i in range(len(T.child)):  
 count += NodesAtDepth(T.child[i], depth - 1)  
 return count  
  
  
  
def PrintAtDepth(T, depth):  
 if depth == 0:  
 for i in range(len(T.item)):  
 print(T.item[i])  
 else:  
 for i in range(len(T.child)):  
 PrintAtDepth(T.child[i], depth - 1)  
  
def NumberOfFullNodes(T):  
 if len(T.item) >= T.max\_items:  
 return 1  
 if T.isLeaf:  
 return 0  
 count = 0  
 for i in range(len(T.child)):  
 count += NumberOfFullNodes(T.child[i])  
 return count  
  
  
  
def NumberOfFullLeaves(T):  
 if len(T.item) >= T.max\_items:  
 if T.isLeaf:  
 return 1  
 count = 0  
 for i in range(len(T.child)):  
 count += NumberOfFullLeaves(T.child[i])  
 return count  
  
def FindDepthOfElement (T, k):  
 i = 0  
 while i < len(T.item) and T.item[i] < k:  
 i += 1  
  
 if len(T.item) == i:  
 if T.isLeaf:  
 return -1  
 else:  
 depth = FindDepthOfElement(T.child[i], k)  
 if depth >= 0:  
 return depth + 1  
 return -1  
 elif T.item[i] > k:  
 if T.isLeaf:  
 return -1  
 else:  
 depth = FindDepthOfElement(T.child[i], k)  
 if depth >= 0:  
 return depth + 1  
 return -1  
 return 0  
  
# List = []  
# List = [1, 2, 3, 4, 5, 6]  
# List = [-1, -2, -3, -4, -5, -6]  
# List = [30, 50, 10, 20, 60, 70, 100, 40, 90, 80, 110, 120, 1, 11 , 3, 4, 5, 105, 115, 200, 2, 45, 6]  
# List = [42, 3, 17, 19, 7, 25, 14, 8, 5, 16, 15, 0, 20, 4, 18, 13, 9, 20, -60, 12, 21, 2, 1, -5, 6]  
List = [42, 3, 17, 19, 7, 25, 14, 8, 5, 16, 15, 0, 20, 4, 18, 13, 9, 20, -60, 12, 21, 2, 1, -5, 6, 90, 105, 256, 1000,  
 60, 74, 99, 100, 26]  
  
T = BTree()  
for i in List:  
 print('Inserting', i)  
 Insert(T, i)  
 PrintD(T, '')  
 # Print(T)  
 print('\n####################################')  
  
print("Height of B-Tree:")  
print(HeightOfTree(T))  
List2 = []  
ExtractItemsToSOrtedList(T, List2)  
print("Sorted list:")  
print(List2)  
  
print("Smallest at depth 1")  
print(SmallestAtDepth(T, 1))  
  
print("Largest at depth 1")  
print(LargestAtDepth(T, 1))  
  
print("Number of nodes at depth -1")  
print(NodesAtDepth(T, -1))  
  
print("Items at depth 1")  
print(PrintAtDepth(T, 1))  
  
print("Number of full nodes:")  
print(NumberOfFullNodes(T))  
  
print("Number of full leaves:")  
print(NumberOfFullLeaves(T))  
  
print("Depth of -1:")  
print(FindDepthOfElement(T, -1))  
print("Depth of 1:")  
print(FindDepthOfElement(T, 1))  
print("Depth of 24:")  
print(FindDepthOfElement(T, 24))  
  
elapsed\_time = time.time()-start  
print(elapsed\_time)