Ivan Perez
80602918
Lab 4 Report

Introduction

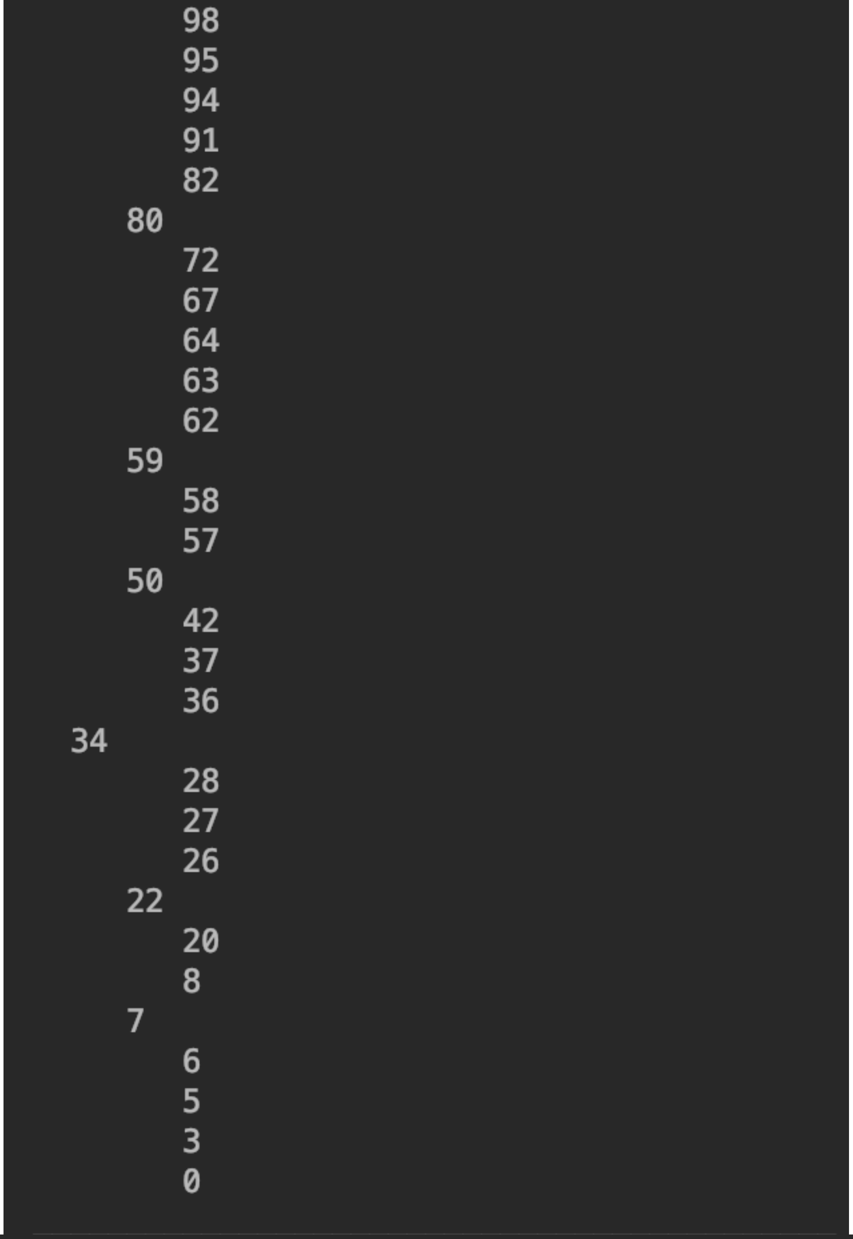
Using the object Btree, we are to create methods that increase the usability of Btrees. The methods to be created are finding the height, making a Btree into a sorted list, finding the minimum and maximum at a depth, the number of nodes at a depth, a print items at a depth, how many full leaves and nodes there are, and finding the depth of a certain element.

Proposed solution design and implementation

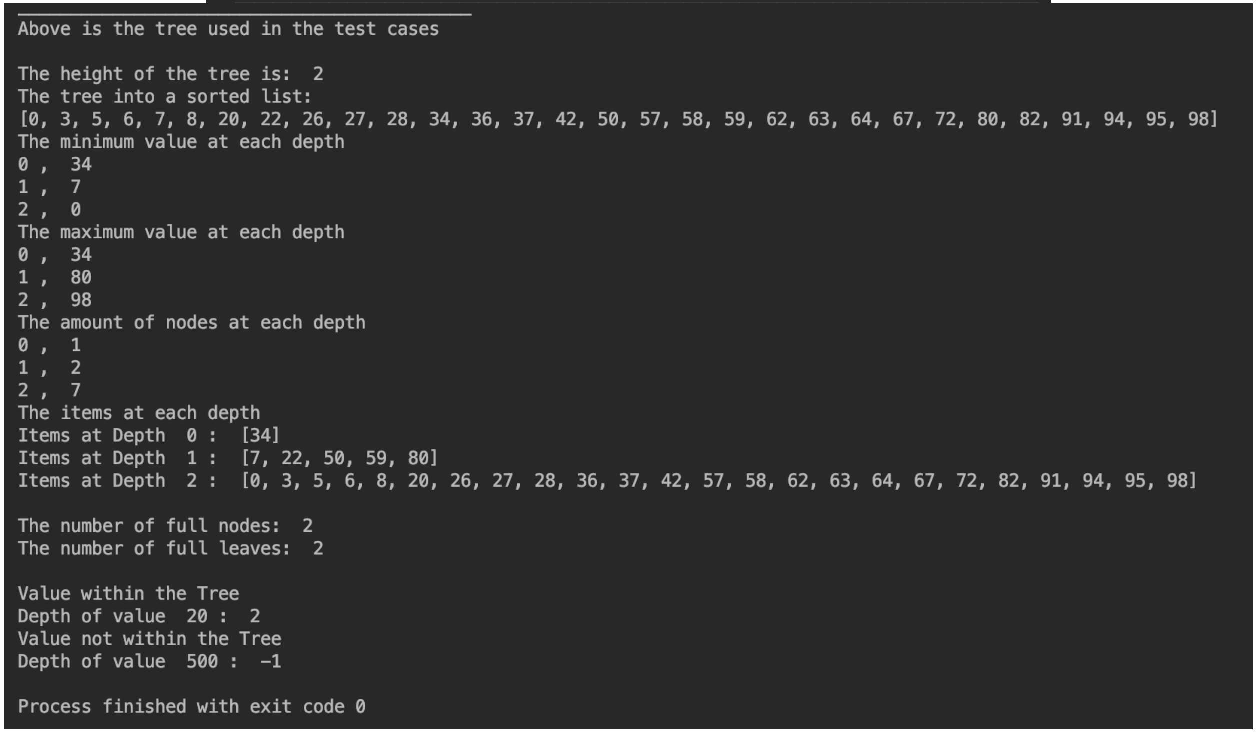
Btrees are self- balancing trees, meaning that for the height method I simply went down one side of the tree and counted the heights it went through recursively. Btrees have nodes and their children, By going through the leaf children first and placing those values into a sorted list, we can place the parents once each of its children is placed. Finding the minimum and maximum values of a depth just required us to go to the far left of the depth and retrieve the minimum while far right for the maximum. The number of nodes at a depth would just have the code go through the list and go the wanted depths parents and count the children those nodes have for the nodes at the depth wanted and return recursively. Printing items at a depth required 2 methods, one that retrieves the values recursively and the other to print the values in the list that the previous method returned. The recursive method goes to the depth needed and returns the list of items at depth and the method that called it would add it to the list it currently has. Full leaves and Full nodes were the same except that Full leaves would check to see if its a leaf first. The code would traverse through each depth and return the value of how many nodes or leaves were full. Finding the depth of the element just checked if the value was in current node. If not it would add 1 for each traversal it needed to by calling the Findchild method to traverse only the needed part of the tree. If the value wasn't found it would return -1 and each call would first check if its -1 before adding 1 for depth its created in.

Experimental results

I imported the random tool and created a sample list of size 30. From there I created a Btree object and used a for loop to input the values in the tree. I then just ran every code by printing a suitable statement in the command prompt and calling the method with the tree to see if it printed the correct values. I tried different list lengths and stuck with 30 because I felt it was a good size to demonstrate the values. Below are the results of the tests I did.



```
        98
        95
        94
        91
        82
    80
        72
        67
        64
        63
        62
    59
        58
        57
    50
        42
        37
        36
34
        28
        27
        26
    22
        20
        8
    7
        6
        5
        3
        0
```



```
__________________________________________
Above is the tree used in the test cases

The height of the tree is:  2
The tree into a sorted list:
[0, 3, 5, 6, 7, 8, 20, 22, 26, 27, 28, 34, 36, 37, 42, 50, 57, 58, 59, 62, 63, 64, 67, 72, 80, 82, 91, 94, 95, 98]
The minimum value at each depth
0 ,  34
1 ,  7
2 ,  0
The maximum value at each depth
0 ,  34
1 ,  80
2 ,  98
The amount of nodes at each depth
0 ,  1
1 ,  2
2 ,  7
The items at each depth
Items at Depth  0 :  [34]
Items at Depth  1 :  [7, 22, 50, 59, 80]
Items at Depth  2 :  [0, 3, 5, 6, 8, 20, 26, 27, 28, 36, 37, 42, 57, 58, 62, 63, 64, 67, 72, 82, 91, 94, 95, 98]

The number of full nodes:  2
The number of full leaves:  2

Value within the Tree
Depth of value  20 :  2
Value not within the Tree
Depth of value  500 :  -1

Process finished with exit code 0
```

Conclusions

I learned how to traverse a Btree and a more in depth understanding of Btrees. Btrees seemed extremely complicated but seeing how it works makes it an efficient way to store items. I learned how to read a Btree by extracting elements of a Btree into a sorted list.

Appendix

```
################################################################
################
#
# Course: CS2302 (MW - 1:30pm)
# Author: Ivan Perez
# Assignment: Lab 4 B-Trees
# Instructor: Olac Fuentes
# T.A.: Anindita Nath
# Last Modified: March 25, 2019
# Purpose: This code contains the object Btree along with many
of the essential
# methods to utilize it. Besides key ones like insert and
search, teh ones required
# to make for this lab are height computation, extract to sorted
list, find minimum
# and maximum element at given depth, number of nodes in a
depth, printing all the
# items in a given depth, the number of full nodes and leaves of
a tree, and what
# depth a value is in.
#
################################################################
################

import math
import random

############# Below is the code provided to us by Mr. Fuentes
################
# 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 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=' ')
```

```python
        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)


################################################################
################
# Everything below is what was required by the assignment and
created by myself
# Each method is in order of the worksheet


def Height(T):
    # Prints the height of the tree
    if T.isLeaf:
        return 0
    return 1 + Height(T.child[0])


def TreeToList(T):
    # Makes a sorted list from the tree
    if T is None:
        return []
    if T.isLeaf:
        return T.item
    l = []
    n = 0
    # goes through the trees children
    for i in range(len(T.child)):
        if n < len(T.item): # if statement checks where to place
parent
            if T.item[n] < T.child[i].item[0]:
                l += [T.item[n]]
                n += 1
        l += TreeToList(T.child[i]) # recursively creates list
    return l

def MinAt_d(T,d):
    # Finds the minimum value at a given depth
    if d == 0:
        return T.item[0]
    if T.isLeaf:
        print('depth does not exist')
        return math.inf
    return MinAt_d(T.child[0], d-1)


def MaxAt_d(T,d):
    # Finds the maximum value at a given depth
    if d == 0:
        return T.item[len(T.item)-1]
    if T.isLeaf:
        print('depth does not exist')
        return -math.inf
    return MaxAt_d(T.child[len(T.child)-1], d-1)


def NumNodesAt_d(T, d):
    # returns the number of nodes at a given depth
    if T is None:
        return 0
    if d == 0:
        return 1
    if d == 1:
        return len(T.child)
    n = 0
    for i in range(len(T.child)):
        n += NumNodesAt_d(T.child[i], d-1)
    return n


def ItemsAt_d(T, d):
    # Gets the items at a given depth and returns a string of
them
    if T is None:
        return []
    if d == 0:
        return T.item
    if T.isLeaf:
        return []
    l = []
    for i in range(len(T.child)):
        l += ItemsAt_d(T.child[i], d-1)
    return l

def PrintItemsAt_d(T, d):
    # Prints the items received from calling the method
ItemsAt_d
    l = ItemsAt_d(T, d)
    print('Items at Depth ',d,': ', l)


def FullNodes(T):
    # Returns the amount of nodes that are full
    if T is None:
        return 0
    if not T.isLeaf:
        sum = 0
        if len(T.item) == T.max_items:
            sum +=1
        for i in range(len(T.child)):
            sum += FullNodes(T.child[i])
        return sum
    else: # when it reaches the end of the list
        if len(T.item) == T.max_items:
            return 1
        else:
            return 0


def FullLeaves(T):
    # returns the amount of leaves that are full
    if T is None:
        return 0
    if T.isLeaf: # checks if list before doing anything
        if len(T.item) == T.max_items:
            return 1
        else:
            return 0
    sum = 0
    for i in range(len(T.child)): # iterates
        sum += FullLeaves(T.child[i])
    return sum


def FindDepthOf_k(T, k):
    # Searches for the value k and returns the depth or -1 if
not found
    if T is None:
        return -1
    if k in T.item:
        return 0
    if T.isLeaf:
        return -1
    x = FindDepthOf_k(T.child[FindChild(T, k)], k)
    if x == -1: # if value isnt found, checks to continue to
return -1
        return -1
    else:
        return 1 + x


#___________________ Test Code __________ __________#

T = BTree()
l = random.sample(range(100), 30)

for i in range(len(l)):
    Insert(T, l[i])

PrintD(T, ' ')
print('___________________________________________')
print('Above is the tree used in the test cases')
print()
print('The height of the tree is: ', Height(T))
print('The tree into a sorted list: ')
print(TreeToList(T))
print('The minimum value at each depth')
for i in range(Height(T)+1):
    print(i, ', ', MinAt_d(T,i))

print('The maximum value at each depth')
for i in range(Height(T)+1):
    print(i, ', ', MaxAt_d(T,i))
print('The amount of nodes at each depth')
for i in range(Height(T)+1):
    print(i, ', ', NumNodesAt_d(T,i))

print('The items at each depth')
for i in range(Height(T)+1):
    PrintItemsAt_d(T,i)


print()
print('The number of full nodes: ', FullNodes(T))
print('The number of full leaves: ', FullLeaves(T))

print()
print('Value within the Tree')
print('Depth of value ', l[10], ': ', FindDepthOf_k(T, l[10]))
print('Value not within the Tree')
print('Depth of value ', 500, ': ', FindDepthOf_k(T, 500))
```

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

Ivan Perez