**Lab 4**

Height method given meets the requirements of Depth, there the method uses recursion to move through one side of the B-tree in order to measure its depth. Extract(T, A) is supposed to extract the elements of the tree in to a list. MinItemAtDepth(T) is used to find the minimum element on the tree by recursively getting to the left side leaf node and then returning the first value of the list. MaxItemAtDepth(T), similar to the previous, it goes to the right and returns the last element of the list to the right. NumNodesAtDepth(T,depth) recursively gets to the appropriate depth and counts the nodes in the variable temp, to return the total amount of nodes at the given depth. PrintAtDepth(T, depth) will get to the appropriate depth and then recursively print the elements in the depth. FullNodes(T) will traverse the tree in search for full nodes and add 1 to the x variable which will return the total amount of full nodes. FullLeaf(T) will only add 1 to the counter if the node is a Leaf. KeyAtDepth(T, k) will check for k in the list first then it will check if the node is a Leaf, if so it will return -1, otherwise it will recursively traverse the tree in search for the k key.

**Experimental Results**

|  |  |
| --- | --- |
| Inputs |  |
| A1 = [30, 50, 10, 20, 60, 70, 100, 40, 90, 80, 110, 120, 1, 11 , 3, 4, 5,105, 115, 200, 2, 45, 6,222,72,45,516,51,22,18,46,99,252,282,300,65,80]  Length = 23 | 516  300  282  252  222  200  120  115  110  105  100  99  90  80  80  72  70  65  60  51  50  46  45  45  40  30  22  20  18  11  10  6  5  4  3  2  1 |
| A2 = [30, 50, 10, 20, 60, 70, 100, 40, 90, 80, 110, 120, 1, 11, 3, 4, 5, 105, 115, 200, 2, 45, 6, 222, 72, 45, 516, 51, 22, 18, 46, 99, 252, 282, 300, 65, 80, 77, 54, 9, 68, 69, 94, 75, 97, 36, 81, 31, 39, 74, 66, 8, 62, 85, 59, 42, 86, 37, 95, 43, 49, 21, 27, 58, 91, 63, 41, 15, 24, 53, 0, 96, 79, 29, 55, 14, 16, 89, 82, 32, 88, 64, 26, 84, 52, 56, 12, 7, 98, 35, 61, 28, 67, 57, 92, 13, 17, 87, 76, 71, 78, 47, 19, 48, 23, 83, 73]  Length = 107 | 516  300  282  252  222  200  120  115  110  105  100  99  98  97  96  95  94  92  91  90  89  88  87  86  85  84  83  82  81  80  80  79  78  77  76  75  74  73  72  71  70  69  68  67  66  65  64  63  62  61  60  59  58  57  56  55  54  53  52  51  50  49  48  47  46  45  45  43  42  41  40  39  37  36  35  32  31  30  29  28  27  26  24  23  22  21  20  19  18  17  16  15  14  13  12  11  10  9  8  7  6  5  4  3  2  1  0 |
| A3 = [71, 92, 73, 107, 117, 196, 90, 15, 10, 19, 198, 35, 193, 140, 21, 174, 16, 30, 54, 102, 66, 40, 86, 175, 25, 103, 157, 9, 143, 12, 63, 109, 72, 61, 185, 176, 150, 11, 164, 48, 104, 80, 82, 169, 149, 101, 44, 136, 151, 84, 70, 178, 173, 29, 20, 137, 26, 129, 38, 123, 187, 199, 31, 87, 0, 36, 5, 75, 148, 192, 42, 168, 51, 139, 127, 122, 23, 7, 158, 14, 130, 153, 144, 53, 60, 3, 116, 180, 194, 170, 39, 2, 167, 113, 121, 56, 126, 58, 81, 190, 89, 111, 22, 59, 186, 27, 62, 188, 132, 88, 52, 94, 8, 34, 65, 106, 4, 97, 177, 74, 184, 79, 37, 64, 124, 17, 159, 105, 128, 32, 171, 67, 115, 146, 160, 6, 46, 95, 69, 91, 152, 28, 13, 68, 147, 96, 145, 49, 100, 141, 55, 161, 134, 179, 112, 57, 77, 119, 135, 197, 131, 120, 191, 1, 200, 99, 50, 163, 110, 76, 182, 189, 154, 33, 138, 166, 172, 43, 85, 155, 133, 108, 18, 93, 98, 118, 83, 165, 41, 183, 125, 156, 195, 181, 47, 24, 114, 78, 162, 142]  Length = 200 |  |

|  |  |  |
| --- | --- | --- |
| Methods and input | | Output |
| Depth | | 3 |
| A1 | |
| A2 | | 4 |
| A3 | | 4 |
|  |  | |
| MinItemAtDepth | | 3 |
| A1 = 1 | |
| A2 = 3 | | 0 |
| A3 = 2 | |  |
| MaxItemAtDepth | | 200 |
| A1 = 1 | |
| A2 = 3 | | 516 |
| A3 = 2 | |  |
| NumNodesAtDepth | | 2 |
| A1 = 1 | |
| A2 = 3 | | 24 |
| A3 =2 | |  |
| PrintAtDepth | | 3 10 30 45 90 110 200 |
| A1 = 1 | |
| A2 = 3 | | 0 1 2 4 5 7 8 9 11 12 13 14 16 17 18 19 21 22 23 26 27 28 29 31 32 35 36 37 40 41 42 43 45 46 47 48 49 50 52 53 54 56 57 58 59 61 62 63 65 66 67 69 70 71 73 74 75 77 78 79 80 81 82 83 84 86 87 88 89 91 92 94 96 97 98 100 105 115 120 222 252 282 300 516 |
| A3 = 2 | | 3 9 15 19 29 35 39 49 53 61 65 74 80 86 96 101 107 111 123 127 130 136 139 147 150 153 160 164 169 178 182 185 193 197 |
| FullNodes | | 2 |
| A1 | |
| A2 | | 5 |
| A3 | | 11 |
| FullLeaf | | 2 |
| A1 | |
| A2 | | 5 |
| A3 | | 11 |
| KeyAtDepth | | -1 |
| A1 = 5 | |
| A2 = 84 | | 3 |
| A3 = 40 | | -1 |

**Conclusion**

The B-Tree structure gives an ordered access to a large amount of numerical information efficiently. And offers access to every section of the tree individually. Traversing the tree can be a little confusing at first because of the lists that are on each node, but once t is understood, the tree becomes a simple tool for organizing information.

**Source Code**

import math

import random

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 1

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 Extract(T, A):

if T.isLeaf:

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

A.append(T.item[i])

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

Extract(T.child[i],A)

def MinItemAtDepth(T,depth):

if depth == 0:

return T.item[0]

if T.isLeaf:

return math.inf

else:

return MinItemAtDepth(T.child[0],(depth-1))

def MaxItemAtDepth(T,depth):

if depth == 0:

return T.item[-1]

if T.isLeaf:

return math.inf

else:

return MaxItemAtDepth(T.child[-1],depth-1)

def NumNodesAtDepth(T, depth):

if depth == 0:

return 1

else:

temp = 0

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

temp += NumNodesAtDepth(T.child[i],depth -1)

#print(temp)

return temp

def PrintAtDepth(T, depth):

if depth == 0:

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

print(T.item[i], end =' ')

else:

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

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

def FullNodes(T):

if IsFull(T):

return 1

x = 0

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

x += FullNodes(T.child[i])

return x

def FullLeaf(T):

if T.isLeaf:

if IsFull(T):

return 1

return 0

x = 0

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

x += FullLeaf(T.child[i])

return x

def KeyAtDepth(T, k):

if k in T.item:

return 0

if T.isLeaf:

return -1

if k > T.item[-1]:

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

else:

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

if k < T.item[i]:

d = KeyAtDepth(T.child[i],k)

if d == -1:

return -1

return d+1

L = []

T = BTree()

depth = 2

for i in L:

#print('Inserting',i)

Insert(T,i)

#PrintD(T,'')

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

PrintD(T, '')

print(len(L))

print(L)

print('\n---------------Depth/Height-----------------')

H = height(T)

print('\nHeight of Tree', height(T))

print('\n-------------Extract------------------------')

B = []

A = Extract(T,B)

print('\n', A)

print('\n---------------Min Item at Depth-------------------')

x = MinItemAtDepth(T,depth)

print('\nMin Item at depth', depth,':', x)

print('\n---------------Max Item at Depth-------------------')

ma = MaxItemAtDepth(T,depth)

print('\nMax Item at depth',depth, ':', ma)

print('\n---------------Number of Nodes at Depth------------')

y = NumNodesAtDepth(T,depth)

print('\n',y)

print('\n----------------Print At Depth-----------------------')

print()

PrintAtDepth(T,depth)

print()

print('\n----------------Full Nodes--------------------------')

x = FullNodes(T)

print('\n',x)

print('\n----------------Full Leaves--------------------------')

x = FullLeaf(T)

print('\n',x)

print('\n----------------Key At Depth-------------------------')

val = random.randrange(201)

print(val)

dep = KeyAtDepth(T, val)

print('\nKey at depth:',dep)

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.

\_\_\_\_\_\_\_\_\_\_\_\_Hugo Chavez\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_