**Introduction**

The objective of this lab is to learn how to manage B-tree. On lab3, we learned how to use binary search tree, therefore, we are continuing b-tree which can accommodates child and items depends on adding element. There are 9 different functions that require to implement in this assignment. They will be introduced below.

**Part-1: find height**

This method is to find the height of the tree. In b-tree, every input requires to change or modify the tree whenever the max\_item reaches. Thus, right and left children will always have the same depth, it is a balanced tree. So, doing recursive call with child[0] or child[-1] will reach the leaf eventually. Add one every recursive call and return the value at the end.

**Part2: Extract tree into a sorted list**

This method is to extract tree into a sorted list. The idea of extract a tree is in infix order, left node and right. First, I need a base case which returns the item whenever T reaches leaf. Otherwise, it will do recursive calls with a for loop which will traverse each child [0:-2]. After the recursive call, add the item at index i. After closing the for loop, add child[-1] part to get the whole tree extracted in sorted array.

**Part3: Return the minimum element at a given depth d**

We know that b tree is created based on value of item. Left side is smaller than the right side. When d=0, return the item. Recursive call is hold with child[0] to get the smallest number and d-1. Needed condition check is to check if d is greater than depth. If t is at leaf but d is not 0, it will return math.inf.

**Part4: Return the maximum element in the tree at a given depth d**

This method is very identical to part 3, but it will do the opposite. Instead of child[0], it will be child[-1] to get the larger element.

**Part5: Return the number of nodes in the tree at a given depth d**

This method is similar to part 3 and 4 since it needs to reach until d=0. In this method will have exactly the same condition as part 3 and 4. 1st, if T is leaf but d int not 0, which occurs when d is greater than height of the tree. 2nd condition is when d = 0, which returns 1 since it shows that there is a node. The only difference is how recursive calls are held. With for loop with index 0 to len(T.child)), calling with T.child[i] and d-1. Return value from recursive call will be stored in a variable and return it at the end.

**Part6: Print all the items in the tree at a depth d**

This method is similar to part5. In part 5, when d=0, statement was to return 1 since there is a node. When d reaches 0, it will print the whole item with a for loop. If T is leaf and d is not 0, it means that d is greater than height so return nothing. Recursive call is same as part 5, but no return value since it is just printing.

**Part7: Return the number of nodes in the tree that are full**

This method is to count all the nodes that has same element as max\_items. The idea of this is similar to print all the element. With for loop, do recursive call with each child. If len(T.item) is same as max\_items, return 1+d, when d is the number of summations of full nodes. If T is leaf, check if len(T.item) is same as max\_items. If so, return 1, otherwise return 0.

**Part8: Return the number of leaves in the tree that are full**

This method is similar to part 7, but the only check point is when T is at leaf. Thus, 1st condition is to check if T is at leaf and len(T.item) is same as max\_items. If so, return 1. To check each leaf, it requires to have a for loop which do recursive calls with each child. Return value from recursive calls will be stored in a variable and return it at the end.

**Part9: Given a key k, return the depth at which it is found in the tree, of -1 if k is not in the tree**

In this method, there are three things that I had to code. The 1st thing is to find the k, 2nd is to get the depth when k is found, and the 3rd is to check if return value is -1. Since k needs to be found, so compare k with each item in T. Depends on the number of k, recursive call will be held. When T becomes at leaf, it will check if k is in T or not. Return -1 if k is not in T, otherwise return 0. The return value will be stored in a variable because we need to check if the value is -1. If -1, it means the k was not found so return -1 again no matter what. Else-case, return 1+d to get the appropriate depth.

**Output: L = [30, 50, 10, 20, 60, 70, 100, 40, 90, 80, 110, 120, 1, 11 , 3, 4, 5, 105, 115, 200]**

The height of the tree: 1

List extracted from the tree: [1, 3, 4, 5, 10, 11, 20, 30, 40, 50, 60, 70, 80, 90, 100, 105, 110, 115, 120, 200]

Min value at depth 2 is: inf

Max value at depth 2 is: -inf

Total nodes at depth 2 is: 0

There is/are 1 full nodes in the tree.

There is/are 0 full nodes at leaf in the tree.

The number 30 is at depth: 0.

**Output: L = [30, 50, 10, 20, 60, 70, 100, 40, 90, 80, 110, 120, 1, 11 , 3, 4, 5, 105, 115, 200, 2, 45, 6, 9, 81, 82, 83, 84, 85, 123, 134, 136, 156, 245, 346, 456, 56, 456, 343, 444, 555, 666, 777, 888, 999, 232, 544, 542, 657, 325, 958, 546, 657, 876, 674]**

The height of the tree: 2

List extracted from the tree: [1, 2, 3, 4, 5, 6, 9, 10, 11, 20, 30, 40, 45, 50, 56, 60, 70, 80, 81, 82, 83, 84, 85, 90, 100, 105, 110, 115, 120, 123, 134, 136, 156, 200, 232, 245, 325, 343, 346, 444, 456, 456, 542, 544, 546, 555, 657, 657, 666, 674, 777, 876, 888, 958, 999]

Min value at depth 2 is: 1

Max value at depth 2 is: 999

Total nodes at depth 2 is: 14

1 2 4 5 6 9 11 20 40 45 50 56 70 80 82 83 84 85 100 105 115 120 134 136 200 232 245 325 343 444 456 542 544 546 657 657 666 674 876 888 958 999 0

There is/are 1 full nodes in the tree.

There is/are 1 full nodes at leaf in the tree.

The number 30 is at depth: 1.

**Big – O notation for methods**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Part1 | Part2 | Part3 | Part4 | Part5 | Part6 | Part7 | Part8 | Part9 |
|  | O(log(n)) | O(n) | O(log(n)) | O(log(n)) | O(n) | O(n) | O(n) | O(n) | O(log(n)) |

"""""""""""""""""""""

Lab 4 - B-trees

03/15/2019 - Ken M. Amamori

CS2302 MW 10:30 - 11:50

Professor: Olac Fuentes

TA: Anindita Nath, Maliheh Zargaran

"""""""""""""""""""""

import math

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

#find the depth at which it is found in the tree

def findDepthOfk(T, k):

d = 0

if T.isLeaf:

if k in T.item: #T.item contians k

return 0

else:

return -1

if k <= T.item[-1]: #k is less than the last element

for i in range(len(T.item)): #compare with each element

if k==T.item[i]: #equal

return 0

if k<T.item[i]: #k is less

d = findDepthOfk(T.child[i], k) #call with appropriate T node

break #break the loop

else:

d = findDepthOfk(T.child[-1], k) #call with last child

if d == -1: #not found case

return -1

else:

return d + 1

#find the number of leaves in the tree that are full

def FullLeaves(T):

if T.isLeaf and len(T.item)==T.max\_items: #len(T.item)=max\_item

return 1

d = 0

for i in range(len(T.child)): #traverse through children

d += FullLeaves(T.child[i])

return d #return value

#find the number of nodes in the tree that are full

def countFullNode(T):

if T.isLeaf: #

if len(T.item) == T.max\_items: #T is full

return 1

else:

return 0

else:

d = 0

for i in range(len(T.child)): #traverse through each element

d += countFullNode(T.child[i])

if len(T.item) == T.max\_items: #checks if it is full

return 1 + d

else:

return d

#print all the items in the tree at a given depth d

def printAtDepth(T, d):

if T.isLeaf and d!=0: #d>height

return

if d==0: #depth wanted

for i in T.item: #traverse through each element

print(i, " ", end='')

else:

for i in range(0, len(T.child), 1):

printAtDepth(T.child[i], d-1) #call evert children

#find the number of nodes in the tree at a given depth d

def nodesAtDepth(T, d):

if T.isLeaf and d!=0:

return 0 #d>height

if d==0:

return 1 #exit one node

k = 0

for i in range(len(T.child)): #traverse each child

k += nodesAtDepth(T.child[i], d-1) #add number of elements to k

return k #return the value

#find the maximum element in the tree at given depth d

def maxAtDepth(T, d):

if T.isLeaf and d!=0: #is leaf and not d=0

return -math.inf

if d==0: #depth wanted

return T.item[-1] #return max element

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

#find the minimum element in the tree at a given depth d

def minAtDepth(T, d):

if T.isLeaf and d!=0: #is leaf but not at d=0

return math.inf

if d==0: #depth wanted

return T.item[0] #return min element

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

#extract the items in the B-tree into a sorted list

def extract\_tree(T):

if T is not None: #not null

if T.isLeaf:

return T.item

d = []

for i in range(len(T.item)): #traverse through child[0:-2]

d += extract\_tree(T.child[i]) + [T.item[i]]

d += extract\_tree(T.child[-1]) #add the last child

return d

#find the height of the tree

def height(T):

if T.isLeaf:

return 0

return 1 + height(T.child[0])

"""Uploaded by Prof. Fuentes"""

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 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+' ')

"""Uploaded by Prof. Fuentes"""

L = [30, 50, 10, 20, 60, 70, 100, 40, 90, 80, 110, 120, 1, 11 , 3, 4, 5, 105, 115, 200, 2, 45, 6, 9, 81, 82, 83, 84, 85, 123, 134, 136, 156, 245, 346, 456, 56, 456, 343, 444, 555, 666, 777, 888, 999, 232, 544, 542, 657, 325, 958, 546, 657, 876, 674]

T = BTree()

for i in L:

print('Inserting',i)

Insert(T,i)

PrintD(T,'')

#Print(T)

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

print("The height of the tree:", height(T)) #part1#

print("List extracted from the tree:", extract\_tree(T)) #part2#

d = 2 #depth

print("Min value at depth",d,"is:", minAtDepth(T, d)) #part3#

print("Max value at depth",d,"is:", maxAtDepth(T, d)) #part4#

print("Total nodes at depth", d,"is:", nodesAtDepth(T, d)) #part5#

printAtDepth(T, 2) #part6#

print(0)

print("There is/are", countFullNode(T), "full nodes in the tree.") #part7#

print("There is/are", FullLeaves(T), "full nodes at leaf in the tree.") #part8#

k = 30 #key number

print("The number",k, "is at depth:", findDepthOfk(T, k),end='.') #part9#

**Conclusion**:

Throughout this lab, I learned how to manage b-tree and found out the common process that occurs in binary search tree and b-tree since both are created depends on input item. In addition, I learned the effectiveness of using recursive method. Since trees has a pattern and it requires to manage all the children. I felt using recursive methods are a lot simpler than iterative method, especially in b-tree because the root can have multiple children. Most of the methods I wrote are recursively, and I don’t see how it will look if I am supposed to use iteration way.

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.