Yahir Rivas

Lab 4 Report

Instructor: Dr. Olac Fuentes

Teaching Assistant: Nath, Anindita

CS 2302

MW 1:30 - 2:50

**Lab 4 Report**

This lab consisted of submitting 9 different method implementations for a B-Tree. B-Trees are similar to binary trees, the difference being that a B-Tree can have up to 3 children, and these children can be in-between 2 nodes. The methods that are going to be explained below, consisted mostly of recursion and counting depending of depth.

**Height**

This method was provided by the professor, it recursively calls each child and counts 1 every time a recursive call is made. Then when the Node is a leaf, meaning it reached the last Node, it returns 0.

**Sorted List**

For the sorted list method, I made for loop inside a recursive call to traverse through every element of the current depth and added to the current list. The recursion goes on until the list is on the leaf nodes. After the loop I added the last child to have all elements of the B-Tree.

**Min at Depth**

This method consists of returning the minimum element at a given depth. So whenever the depth is 0 I return the leftmost element, otherwise make a recursive call that goes to the leftmost child and subtracts 1 to the given depth.

**Max at Depth**

This method returns the maximum element at a given depth. Similar to the previous method I returned the right most element whenever the depth is 0. Other wise recursively call the rightmost child and subtract 1 to the given depth.

**Nodes at Depth**

For this method I made a for loop that sums the length of T.item for every child there is. If not then I recursively call the next child and subtract 1 to the depth until I reach the desired depth.

**Print at Depth**

Similar to all the previous methods I made a for loop that prints every element in T.item which is the element at current depth, if the depth is 0. If not traverse all possible children by making another for loop that goes through every one of them and then make a recursive call for each of those children.

**Full Leaves**

For the last method I made the if statement to be if I was in a leaf node then it would check if the length of T.item is equal to the max number of items allowed, if so then return 1. Otherwise, recursively call every child and add the return value in a variable, then return the value of that variable.

**Index:**

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=' ')

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)

#Method1

def height(T):

if T.isLeaf: #T is at the last Node

return 0

return 1 + height(T.child[0]) #Returns 1 + height of the child

#Method2

def SortedList(T):

l = []

if T is not None:

if T.isLeaf:

return T.item #returns leaf elements to the list

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

l = l + SortedList(T.child[i]) + [T.item[i]] #adds current elements to the list and recursively calls childs

l = l + SortedList(T.child[-1]) #adds the last child on the tree

return l

#Method3

def MinAtDepth(T,d):

if d==0:

return T.item[0] #return first element

if T.isLeaf:

return -1

return MinAtDepth(T.child[0], d-1) #calls recursively for leftmost child

#Method4

def MaxAtDepth(T,d):

if d==0:

return T.item[-1] #returns last element

if T.isLeaf:

return -1

return MaxAtDepth(T.child[-1], d-1) #calls recursively for rightmost child.

#Method5

def NodesAtDepth(T,d):

if d==0:

n = 0

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

n = n + len(T.item)

return n

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

#Method 6

def PrintAtDepth(T,d):

if d==0:

for i in T.item:

print(i) #print every element on T.item

else:

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

PrintAtDepth(T.child[i], d-1) #recursively calls all possible childs

#Method 8

def FullLeaves(T):

if T.isLeaf:

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

return 1

d = 0

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

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

return d #return value

L = [30, 50, 10, 20, 60, 70, 100, 40, 90, 80, 110, 120, 1, 11 , 3, 4, 5,105, 115, 200, 2, 45, 6]

T = BTree()

for i in L:

print('Inserting',i)

Insert(T,i)

PrintD(T,'')

#Print(T)

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

SearchAndPrint(T,60)

SearchAndPrint(T,200)

SearchAndPrint(T,25)

SearchAndPrint(T,20)

print('The height of the tree is: ' , height(T))

l = SortedList(T)

print('The B-Tree as a sorted list: ' ,l)

mn = MinAtDepth(T,2)

mx = MaxAtDepth(T,2)

print('The minimum element of the list is:' ,mn)

print('The maximum element of the list is:' ,mx)

d = 1

n = NodesAtDepth(T,d)

print('The elements at depth' ,d, 'are:' ,n)

PrintAtDepth(T,1)

f = FullLeaves(T)

print('There are' ,f, 'full leaves')

**Conclusion**

The goal of this lab was to familiarize ourselves with B-Trees and the differences it has from the binary trees we already know. This lab was also a good practice on recursion and using loops inside them. It is very important to get to know data structures and what makes them unique and better for certain tasks and this lab helped in getting to know B-Trees better and when to use them.