Author: Carlos Fernando Castaneda

Class : CS 2302

Date Modified: March 25, 2019

Instructor: Olac Fuentes

Assignment: Lab 4 B-Tree

TA: Anindita Nath & Maliheh Zaragan

**Introduction:**

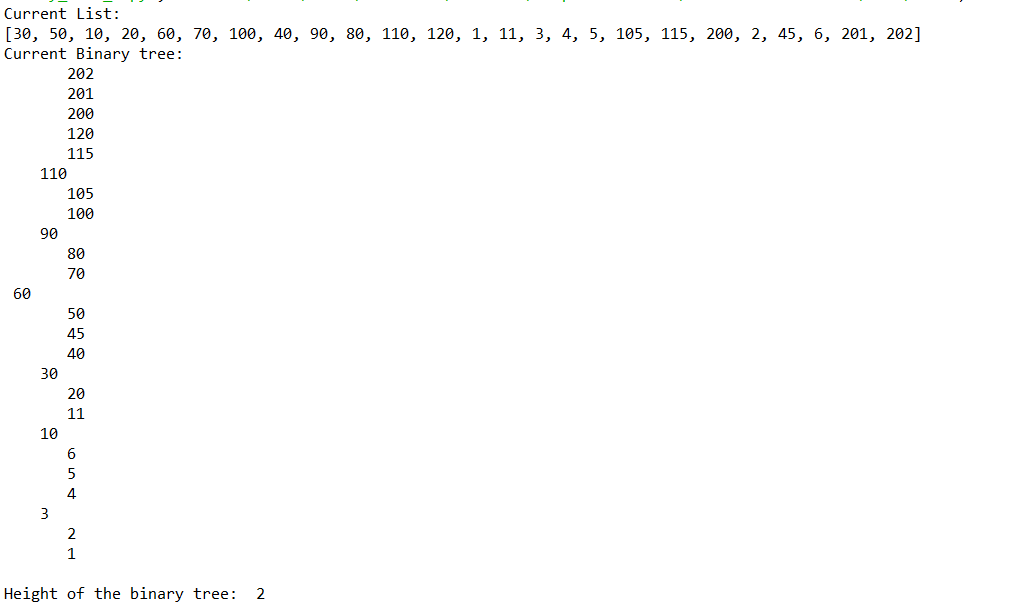
In python, you can use Binary Trees to organize items in a way that can be found using a structure that resembles a tree. Binary trees can be very specific when finding items, and on this lab, I am going to play with some of these functions to solve 9 tasks. The first task is to compute the height of the tree. The second task is to extract the items in the B-tree into a sorted list. The third task is to return the minimum element in the tree at a given depth d. The fourth task is to return the maximum element in the tree at a given depth d. The fifth task is to return the number of nodes in the tree at a given depth d. The sixth task is to print all the items in the tree at a given depth d. The seventh task is to return the number of nodes in the tree that are full. The eight task is to return the number of leaves in the tree that are full. The ninth and final task is when given a key k, return the depth at which it is found in the tree, of -1 if k is not in the tree.

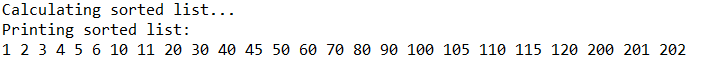
**Proposed solution design and implementation:**

1. Height: For the first task, it is already given to me by the code mad by the Professor. All I need to make sure is that the binary tree is correct so that the height is correct as well.
2. Sorted List: This task has to use the child function to determine where to start implementing the items of a tree into the list. This new list must also be defined before we begin this method otherwise, we would be implemented items into nothing. The trickiest part must be when deciding whenever an item is greater than or less than the current item it is reviewing over. Obviously, this method will not alter the tree at all so we can use it on the other methods.
3. Minimum: This task is checking for the minimum element of the tree, so the solution must be very similar to that I have done on the last lab. The only issue is that unlike the least lab, I cannot use T.left recursively to get my minimum, so I must probably use either T.item or T.child to get my desired value.
4. Maximum: Like Minimum, this task is is checking for the maximum element of the tree, so the solution must be very similar to that I have done on the last lab. The only issue is that unlike the least lab, I cannot use T.right recursively to get my maximum, so I must probably use either T.item or T.child to get my desired value.
5. Node Depth: This method has to read the depth of each level of the binary tree from top to bottom. This solution must be really simple if I recursively find the depth as long as there is a child for that particular item.
6. Print Element Depth: This task must print everything that is contained on the depth d,sort of like the method PrintD given with the lab.
7. Full Node: For this task, I must find a way to check the value of max\_items to that of the length of an item in the tree to return later on to the terminal. I would also need to make sure that this applies to any item, not just the leaves.
8. Full Leaves: For this task, I must find a way to check the value of max\_items to that of the length of an item in the tree to return later on to the terminal. I would also need to make sure that this applies just to the leaves.
9. Search Depth: This task must be able to find the depth of a item in the tree, if it is not found, it must print some sort of indicator to the user. I must also find a way to correctly traverse to different branches of the tree so that it can find the item correctly.

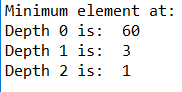
**Experimental results**:

NOTE: For all of my methods, I used the Tree A = [30, 50, 10, 20, 60, 70, 100, 40, 90, 80, 110, 120, 1, 11 , 3, 4, 5,105, 115, 200, 2, 45, 6, 201, 202].

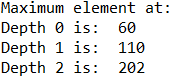
1. Height: As mentioned before, the height method is already given from the sample code, however, I decided to print both the list and the binary tree to check if the method does what it its supposed to do. My results are shown below:
2. Sorted List For this method, I had to find the smallest elements in the tree, and arrange them from smallest to largest as requested. To do that, I found all the leaves of the tree, and added them to my new empty list L. Otherwise, It would call itself again recursively until it formed the tree in order. Afterwards, I printed the list just to be sure. My results are shown below:



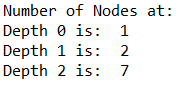
1. Minimum: This method gets the minimum number of a tree but used in terms for the B-tree. For starters, if the depth is 0, it will return the first element in the item, otherwise it will call itself recursively various times until it finds its desired depth. When the depth is found, it will print the smallest number of that depth. My results are shown below:



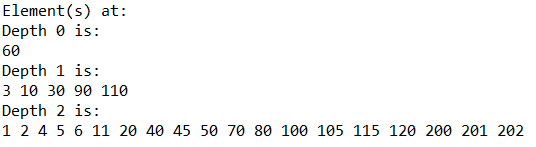
1. Maximum: This method gets the maximum number of a tree but used in terms for the B-tree. For starters, if the depth is 0, it will return the last element in the item, otherwise it will call itself recursively various times until it finds its desired depth. When the depth is found, it will print the largest number of that depth. My results are shown below:



1. Node Depth: On my method, I made a count variable and if the depth is 0, it will count recursively how many items are present in that particular depth. In the root, it will always return 1. My results are shown below:

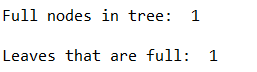


1. Print Element Depth: On my method, I check the depth d, and print all the corresponding items to that specific depth. If the depth is 0, it will print all the elements of an item using a for loop. Otherwise, it will recursively call itself again until the desired depth is found. My results are shown below:

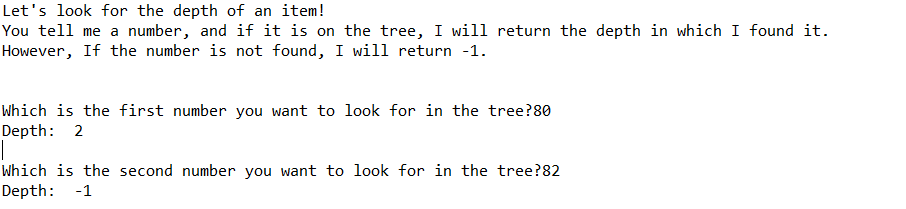


1. Full Node: On my method, I created a count value of 0 and checks if the item is not a leaf, it later checks the value of the length of the item to max\_items. If equal, it will add 1 to count for each node that is full. Once it checks every item in the tree, it will return the count value. My results from my example is shown after Full Leaves on the first line:

1. Full Leaves: On my method, I created a count value of 0 and checks if the item is a leaf, it later checks the value of the length of the item to max\_items. If equal, it will add 1 to count for each node that is full. Once it checks every item in the tree, it will return the count value. My results from my example is shown below on the second line:



1. Search Depth: On my method, I asked the user which number they wanted to search in the method. I called the user twice to display below when it finds and doesn’t find the desired number. As for my actual method, if the number was at depth o, it will return 0, otherwise it will check if it is greater than or less than the root to reach the desired location. If your number is on the tree, then it will print the depth in which they found it, otherwise, it will print -1. An example of the interaction is shown below:

****

**Method Comparison Table**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Task | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| O(n) | O(n) | O(log(n)) | O(n) | O(n) | O(n) | O(n) | O(n) | O(n) | O(log(n)) |

**Conclusions**:

With this lab, I was able to learn to code better using the Python language, including using algorithms to find create and modify binary trees to my advantage. I was also able to learn to solve different problems by using binary trees throughout my lab.

**Appendix :**

**binary\_tree\_2.py**

"""

Author: Carlos Fernando Castaneda

Class : CS 2302

Date Modified: March 16, 2019

Instructor: Olac Fuentes

Assingment: Lab 4 Binary Trees

TA: Anindita Nath & Maliheh Zaragan

Purpose: to implement various algorithms and methods to fully understand the

process of how a Binary Tree works.

"""

#Imports various tools to help us plot the binary tree to be used in this lab

import math

# This class in the program is used to create objects of BTrees, or binary trees

class BTree(object):

#Creates the Constructor

def \_\_init\_\_(self, item=[], child=[], isLeaf=True, max\_items=5):

self.item = item

self.child = child

self.isLeaf = isLeaf

#If the max item will equal 3 if it is less than 3

if max\_items < 3:

max\_items = 3

#If the max items is not odd or greater than 3, it will chang it so that it must be odd and greater or equal to 3

if max\_items % 2 == 0:

max\_items += 1

self.max\_items = max\_items

# Method that is used to find the correct index position of child

def FindChild(T, k):

#For i in range of the length of the item

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

#If the key is less than the child, it will return the index

if k < T.item[i]:

return i

#Returns the length of the item

return len(T.item)

#Method that is used to insert items into the binary tree into non-leaf nodes

def InsertInternal(T, i):

#If T is a leaf it will insert the item as a leaf node

if T.isLeaf:

InsertLeaf(T, i)

else:

#Will return the correct position of the child

k = FindChild(T, i)

# The method checks if node is full, and if so, will find a new location to place it in the tree

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)

# Method is used to split full nodes to be used throughout the tree

def Split(T):

#Checks and gets the middle position of the node

mid = T.max\_items // 2

# If it is a leaf node, it creates a left and right child

if T.isLeaf:

# Creates left child with array elements from 1st to index before mid

leftChild = BTree(T.item[:mid])

# Creates right child with array elements from index after mid to last

rightChild = BTree(T.item[mid + 1:])

else:

# Creates left child with array elements from 1st to index before mid and points the splitted node to its left child

leftChild = BTree(T.item[:mid], T.child[:mid + 1], T.isLeaf)

# Creates left child with array elements from 1st to index before mid and points the splitted node to its left child

rightChild = BTree(T.item[mid + 1:], T.child[mid + 1:], T.isLeaf)

return T.item[mid], leftChild, rightChild

# Method is used to insert leaf nodes into the tree

def InsertLeaf(T, i):

#Adds the leaf to the tree, and then sorts the item afterwards

T.item.append(i)

T.item.sort()

# Method is used to check if node is full using the variable max\_items from the constructor

def IsFull(T):

return len(T.item) >= T.max\_items

# Method is used to insert items into the nodes of the tree

def Insert(T, i):

# Checking if node is full, and if so inserts it into non-leaf nodes

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)

# Method that is used to find height of tree

def height(T):

if T.isLeaf:

return 0

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

# Method is used to search item in a B Tree

def Search(T, k):

if k in T.item:

return T

if T.isLeaf:

return None

return Search(T.child[FindChild(T, k)], k)

# Method that is used to get number of leaves that are full in the tree

def fullLeaves(T):

count =0

#Checks that the length is equal to the maximum number required

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

return 1

else:

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

count += fullLeaves(T.child[i])

return count

# Method that is used to get the number of full nodes in the tree

def fullNodes(T):

count = 0

if not T.isLeaf:

for c in T.child:

count += fullNodes(c)

#Checks that the length is equal to the maximum number required

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

count += 1

return count

# Method that prints the items in the tree in ascending order

def Print(T):

# Prints the leaf node items

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)])

# Method that prints the items in the structure of B-tree

def PrintD(T, space):

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 + ' ')

# Method that extracts the items in the B-tree into a sorted list

def sortedBTree(T, L):

#If T is a leaf, it will append the leaf node items into list

if T.isLeaf:

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

L.append(T.item[i])

else:

# Recursively calls on the trees children to append their items

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

sortedBTree(T.child[i], L)

L.append(T.item[i])

sortedBTree(T.child[-1],L)

return L

# Method that is used to get the number of nodes at a give depth d

def NumberOfNodesAtD(T,d):

# Counter to keep track of the nodes

count = 0

#If d is 0, it will return 1

if d == 0:

return 1

#Else, it will keep counting the number of nodes at given depth

else:

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

count+=NumberOfNodesAtD(T.child[i],d-1)

return count

# Method that is used to print all items in tree at a given depth d

def printItemsAtD(T,d):

#If d is 0, it will print the items at index o

if d == 0:

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

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

#Calls the method again in the range of the item T until it prints the rest of the items

else:

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

printItemsAtD(T.child[i],d-1)

printItemsAtD(T.child[-1],d-1)

# Method that is used get the depth at which a given key, k is found in the tree

def findDepthK(T,k):

#If k is in T, return 0

if k in T.item:

return 0

#If T is a leaf, return -1

if T.isLeaf:

return -1

#If k is greater than T.item[-1], it will call itself on the last child of that particular node

if k > T.item[-1]:

s = findDepthK(T.child[-1], k)

else:

# Checks for the correct index of the node's child where k can be found

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

if k < T.item[i]:

s = findDepthK(T.child[i],k)

# Break out of loop if k is found

break

#If s is -1, it will return -1

if s == -1:

return -1

#Else retruns d + 1

return s + 1

# Method that is used to find the maximum element in the tree at a given depth d

def maximumDepthD(T,m):

# If the maximum depth is 0, it will return the item at index -1

if m == 0:

return T.item[-1]

# If T is a leaf, it will return infinity

if T.isLeaf:

return -math.inf

else:

return maximumDepthD(T.child[-1],m-1)

# Method that is used to find the minimum element in the tree at a given depth d

def minimumDepthD(T,m):

# If the minimum depth is 0, it will return the item at index 0

if m == 0:

return T.item[0]

# If T is a leaf, it will return infinity

if T.isLeaf:

return -math.inf

else:

return minimumDepthD(T.child[0],m-1)

#Sets a new list called A.

A = [30, 50, 10, 20, 60, 70, 100, 40, 90, 80, 110, 120, 1, 11 , 3, 4, 5,105, 115, 200, 2, 45, 6, 201, 202]

#Prints the initial List that will be used in the program

print('Current List: ')

print(A)

#Sets a new Empty tree caled T

T = BTree()

#Inserts the content of the List into the tree T

for i in A:

Insert(T, i) # Inserting list items in list

#Prints the binary tree

print('Current Binary tree: ')

PrintD(T,'')

#Task 1

#Prints the height of the tree

print()

depth = height(T)

print('Height of the binary tree: ', depth)

print()

#Task 2

#Prints the extracted items from the binary tree into a sorted list

L = []

print('Calculating sorted list...')

sortedList = sortedBTree(T,L)

print('Printing sorted list: ')

for i in L:

print(i,end = ' ')

print()

print()

#Task 3

#Prints the minimum element of a tree at a given depth d from 0 to 2

print('Minimum element at: ')

for i in range(depth+1):

print('Depth' ,i, 'is: ',minimumDepthD(T,i))

print()

#Task 4

#Prints the maximum element of a tree at a given depth d from 0 to 2

print('Maximum element at: ')

for i in range(depth+1):

print('Depth' ,i, 'is: ',maximumDepthD(T,i))

print()

#Task 5

#Prints the number of nodes of a tree at a given depth d from 0 to 2

print('Number of Nodes at: ')

for i in range(depth+1):

print('Depth' ,i, 'is: ',NumberOfNodesAtD(T,i))

print()

#Task 6

#Prints the items of a tree at a given depth d from 0 to 2

print('Element(s) at: ')

for i in range(depth+1):

print('Depth' ,i, 'is: ')

printItemsAtD(T,i)

print()

print()

#Task 7

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

print('Full nodes in tree: ', fullNodes(T))

print()

#Task 8

#Prints the number of leaves of the tree that are full

print('Leaves that are full: ',fullLeaves(T))

print()

#Task 9

#Asks user to return the depth of th tree given a particular k 'k'. If it is not found, it will return -1

print("Let's look for the depth of an item!")

print('You tell me a number, and if it is on the tree, I will return the depth in which I found it.')

print('However, If the number is not found, I will return -1.')

print()

k1 = int(input('Which is the first number you want to look for in the tree?'))

print('Depth: ',findDepthK(T,k1))

k2 = int(input('Which is the second number you want to look for in the tree?'))

print('Depth: ',findDepthK(T,k2))

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 provide inappropriate assistance to any student in the class.

