Course: CS 2302

Author: Kimberly Morales

Assignment: Lab 4

Instructor: Olac Fuentes

TA(s): Anindita Nath, Maliheh Zargaran

Date: 3/25/2019

# Introduction

Lab 4 is all about trying to solve various problems of B trees by breaking down the components based on three types. By using these types, it becomes easier to see how traversing through a b tree is common and makes use of reusing code.

# Proposed Solution Design and Implementation

I remember in class that there are 3 types of problems that encompass b trees. First, following a path, independent of the tree’s content. Second, traverse through the tree, independent of the trees content. Third, follow a single path, determined by the tree’s content. With these guidelines, I separated each method to a type and based my solution around it.

Type 1:

Height

To find the height is to find the max depth of a B-tree or to hit a depth where all nodes are leaves. First, the method checks if the current tree is a leaf and if it is then it has no height therefore it returns 0. If not, then the method is called recursively throught the first child with a 1 added until it hits the leaf of the b tree.

Min\_ele

I know that to find the minimum element, all I must do is traverse by the first child of each node. The only index that I need is 0. If the depth is 0 then it just returns the first element in the tree since there is no other branches to traverse. If the tree is already a leaf, then it returns the first element. Otherwise, a recursive call is used to go through each branch and traverse the first children while subtracting one from depth since that would mean that the tree has reached the end.

Max\_ele

I know that to find the maximum element, all I must do is traverse by the last child of each node. Python can easily access the last element by using -1 since it indicates to start from the end of the list. Similarly, to the min\_ele method, this time we traverse thought the last element each time and will the item if the tree is currently a leaf or the depth has reached 0.

Type 2

Bt\_to\_slist

I reused the PrintD code and had it as a structure since it prints all elements in order, except this method appends it to a new sorted list. First, the method checks if the current tree is a leaf and if it is then it just appends all the elements in the leaf and returns the list. Otherwise, the tree is traversed based on the current node’s length and traverses through the children of the current node. Once the recursive call is finished then the item is appended into the sorted list. Then another recursive call is used to go to the next item in the current node. Once it has reached a leaf, it will return a sorted list.

Num\_nodes

To count the number nodes within a given depth means to use a recursive call that has d to handle how far it is from the root node. I know that if the tree is currently a leaf then all that there is to do is count the length of the current node. If the depth is 0 then just return the length of the root node. Else, then a counter is used to count the number of nodes in each recursive call while going through each child.

Print\_at\_d

This solution was like num\_nodes so I reused part of the code except recursively call within the loop and print the items of the tree once d is zero.

Full\_nodes and Full\_leaves

I’m combining my explanations for both methods since they are almost exactly the same except for a minor difference. While they can both output the same number, there are cases where there will be different numbers. Most of the cases, however, are almost similar since the leaves are often the part of the tree that gains more members due to insert operations. Both methods have a counter that adds one for each traversal through a tree’s child within a loop. However, full\_nodes base case is that if the number of items of the current node is equal to the max items then it returns 1. Full\_leaves base case check if the current node is a leaf and if it is also having the max number of items then it returns 1. If not, then it returns 0 since the leaf is not full.

Type 3

K\_at\_d

This problem is a mix of searching and using parts of the type 2 problems since we are printing where key’s depth is. We must navigate based on whether the current node is greater or less than the key. If current node is greater than return the current index. Else, then keep traversing. Since the class code of the b tree already has a find child method, I used it to find an index of k and add one to the depth if it is not found.

Main and User I/O

I have the usual try and catch errors if a user inputs a letter, negative, or empty spaces. The program will not run any methods if the size of the list is 0 or negative since there is nothing to do if a tree is entirely empty. To display the btree, I reused code to insert each item from a randomly generated list that is based on the size inputted by the user and shows the structure. Results of the methods re shown in the main method and have the option to make the depth and key user inputted.

# Experimental Results

My results are based on increments of 25 due to my laptop’s limits since 1000 almost fried it. No erroneous input is included on the table since the main goal is to see the runtime for each method and the overall runtime when adding all values. I used a timer for each method and only added all runtimes in this report since Excel has a good sum function. The output of the screen is shown of the command prompt with slight indentation errors due to differences between Spydar and the command line.

|  |  |  |
| --- | --- | --- |
| Input | Time | Output |
| N = 25 | Height = 0.000478719  Min = 0.000932  Max = 0.00104405  Bt to slist = 0.00124  Num of nodes in d = 0.0013401588563977758  Print items in d = 0.010542071004099408  Num of full nodes = 0.0011059190562824062  Num of full leaves = 0.0010687990879581118  Key k at d = 0.0006024528192402623 | 124  122  121  120  119  110  109  103  100  90  78  74  71  70  64  59  52  49  34  16  14  13  8  7  4  Height: 1  Min Element in depth 0: 16  Max Element in depth 0: 120  B-Tree to Sorted List: [4, 7, 8, 13, 14, 16, 34, 49, 52, 59, 64, 70, 71, 74, 78, 90, 100, 103, 109, 110, 119, 120, 121, 122, 124]  Num of nodes in depth 0: 5  Print items in depth 0:  [16, 52, 71, 109, 120]  Num of full Nodes: 1  Num of full Leaves: 2  Key 11 at depth: Not Found |
| N = 50 | Height = 0.00047871959149261526  Min = 0.0009326925374357015  Max = 0.0010440524424085826  Bt to slist = 0.001244158938317706  Num of nodes in d = 0.0013401588563977758  Print items in d = 0.010542071004099408  Num of full nodes = 0.0011059190562824062  Num of full leaves = 0.0010687990879581118  Key k at d = 0.0006024528192402623 | 148  147  141  135  134  133  128  126  125  121  119  117  115  112  111  110  109  108  103  100  92  90  89  87  86  80  79  72  68  64  60  55  51  50  44  43  38  34  31  28  27  26  19  14  12  9  8  6  5  0  Height: 1  Min Element in depth 0: 16  Max Element in depth 0: 120  B-Tree to Sorted List: [4, 7, 8, 13, 14, 16, 34, 49, 52, 59, 64, 70, 71, 74, 78, 90, 100, 103, 109, 110, 119, 120, 121, 122, 124]  Num of nodes in depth 0: 5  Print items in depth 0:  [16, 52, 71, 109, 120]  Num of full Nodes: 1  Num of full Leaves: 2  Key 11 at depth: Not Found |
| N = 75 | Height = 0.0005751461758752633  Min = 0.0007129593916079856  Max = 0.0009100792233990617  Bt to slist = 0.0009241592113841384  Num of nodes in d = 0.0004761595936771477  Print items in d = 0.0021358915107059107  Num of full nodes = 0.005970768238277772  Num of full leaves = 0.001274878912103327  Key k at d = 0.0004735995958616776 | 171  170  169  165  163  162  158  156  154  153  152  150  145  144  138  136  134  133  132  131  129  123  121  119  117  114  113  112  111  110  107  105  102  97  93  92  86  84  83  81  79  77  76  74  73  72  71  69  68  64  54  52  51  50  49  47  45  41  40  36  33  28  25  23  16  14  13  12  11  9  8  7  3  1  0  Height: 2  Min Element in depth 0: 47  Max Element in depth 0: 123  B-Tree to Sorted List: [0, 1, 3, 7, 8, 9, 11, 12, 13, 14, 16, 23, 25, 28, 33, 36, 40, 41, 45, 47, 49, 50, 51, 52, 54, 64, 68, 69, 71, 72, 73, 74, 76, 77, 79, 81, 83, 84, 86, 92, 93, 97, 102, 105, 107, 110, 111, 112, 113, 114, 117, 119, 121, 123, 129, 131, 132, 133, 134, 136, 138, 144, 145, 150, 152, 153, 154, 156, 158, 162, 163, 165, 169, 170, 171]  Num of nodes in depth 0: 3  Print items in depth 0:  [47, 92, 123]  Num of full Nodes: 4  Num of full Leaves: 4  Key 11 at depth: 2 |
| N = 100 | Height = 0.000477439592584881  Min = 0.0005269328836839393  Max = 0.0010884257378767038  Bt to slist = 0.0013947721431277716  Num of nodes in d = 0.0010606924282091281  Print items in d = 0.001530452027347605  Num of full nodes = 0.0028808508750072515  Num of full leaves = 0.0011630923408278704  Key k at d = 0.0009339725363434366 | Height: 3  Min Element in depth 0: 97  Max Element in depth 0: 97  B-Tree to Sorted List: [0, 2, 4, 9, 10, 11, 12, 13, 14, 15, 16, 18, 20, 22, 23, 24, 25, 26, 27, 36, 37, 40, 41, 49, 51, 56, 57, 58, 59, 64, 65, 66, 67, 68, 70, 71, 73, 74, 75, 78, 80, 82, 83, 84, 88, 92, 94, 95, 96, 97, 99, 100, 101, 104, 105, 107, 109, 110, 111, 112, 117, 118, 121, 123, 124, 125, 127, 128, 131, 134, 135, 136, 137, 138, 141, 142, 143, 144, 148, 149, 150, 153, 154, 158, 159, 160, 161, 162, 167, 169, 172, 173, 175, 176, 178, 184, 185, 187, 189, 191]  Num of nodes in depth 0: 1  Print items in depth 0:  [97]  Num of full Nodes: 4  Num of full Leaves: 3  Key 11 at depth: 2 |
| N = 125 | Height = 0.004018343237680438  Min = 0.0005132795620014403  Max = 0.0023965846215811236  Bt to slist = 0.0011345056985551365  Num of nodes in d = 0.0010969590639282652  Print items in d = 0.0010261324577003038  Num of full nodes =  0.004450982868494621  Num of full leaves = 0.0012369056111738788  Key k at d = 0.0004326396308141807 | Height: 3  Min Element in depth 0: 139  Max Element in depth 0: 139  B-Tree to Sorted List: [0, 1, 2, 3, 4, 7, 9, 10, 12, 13, 15, 16, 17, 20, 22, 23, 24, 27, 28, 29, 30, 31, 33, 35, 36, 37, 39, 40, 41, 45, 46, 49, 52, 54, 55, 57, 62, 63, 64, 65, 66, 67, 68, 69, 70, 72, 73, 74, 75, 76, 77, 78, 79, 84, 85, 86, 87, 91, 92, 93, 97, 98, 100, 102, 105, 107, 110, 112, 118, 119, 123, 125, 130, 132, 133, 135, 136, 139, 142, 143, 144, 146, 147, 148, 149, 150, 152, 153, 154, 155, 159, 161, 162, 163, 164, 165, 166, 168, 170, 171, 173, 175, 176, 177, 181, 182, 184, 185, 187, 189, 191, 192, 193, 195, 197, 198, 208, 209, 210, 212, 214, 218, 219, 220, 224]  Num of nodes in depth 0: 1  Print items in depth 0:  [139]  Num of full Nodes: 3  Num of full Leaves: 3  Key 11 at depth: Not Found |
| N = 150 | Height = 7.679993446405592e-05  Min = 0.00022741313927412117  Max = 0.0001501865385074871  Bt to slist = 0.00044586628619410244  Num of nodes in d = 0.0001365332168249884  Print items in d = 0.00030421307373817703  Num of full nodes = 0.00017791984817506308  Num of full leaves = 0.00022015981213029372  Key k at d = 0.0001318398874966295 | Height: 3  Min Element in depth 0: 130  Max Element in depth 0: 130  B-Tree to Sorted List: [0, 2, 3, 5, 8, 9, 10, 12, 14, 15, 17, 19, 20, 22, 23, 24, 25, 27, 28, 32, 34, 35, 38, 39, 40, 41, 46, 47, 48, 49, 50, 51, 53, 54, 55, 57, 59, 60, 62, 64, 65, 67, 68, 69, 74, 76, 78, 79, 80, 82, 85, 86, 90, 92, 93, 94, 96, 97, 98, 101, 105, 106, 107, 111, 112, 113, 114, 115, 116, 117, 118, 120, 121, 123, 124, 125, 126, 127, 128, 130, 132, 134, 137, 138, 139, 140, 141, 144, 146, 147, 148, 149, 150, 151, 152, 153, 154, 156, 158, 159, 161, 165, 168, 169, 171, 172, 174, 176, 179, 180, 181, 182, 185, 187, 188, 189, 190, 191, 194, 195, 196, 199, 200, 202, 203, 208, 209, 212, 213, 214, 216, 217, 218, 219, 223, 225, 226, 228, 229, 230, 231, 234, 235, 237, 239, 241, 242, 246, 247, 248]  Num of nodes in depth 0: 1  Print items in depth 0:  [130]  Num of full Nodes: 7  Num of full Leaves: 7  Key 11 at depth: Not Found |

# Conclusions

Overall, I started off with a rough start since b trees didn’t make complete sense to me until I traced and realized that many of the problems are alike. Writing code for b trees also heavily needs comprehension of recursive calls and how local variables can be used within methods to count. The lab also showed the differences between a b tree and binary search tree since b trees can easily balance themselves.

# Appendix

# """

# Course: CS 2302 [MW 1:30-2:50]

# Author: Kimberly Morales

# Assignment: Lab 4

# Instructor: Olac Fuentes

# TA(s): Anindita Nath , Maliheh Zargaran

# Date: 3/15/2019

# Date of last modification: 3/24/2019

# Purpose of program:

# To create a more substantial b tree program with specific b tree operations that follow certain problem solving approaches.

# """

# #################################################################################################

# #B TREE CLASS

# #################################################################################################

# import random

# import timeit

# 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)

# 

# #################################################################################################

# #LAB METHODS

# #################################################################################################

# #1. Compute the height of the tree

# def height(T):

# if T.isLeaf:

# return 0

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

# #2. Extract the items in the B-tree into a sorted list.

# def bt\_to\_slist(T,s\_list):

# #Just append all items to the new list

# if T.isLeaf:

# for t in T.item:

# s\_list.append(t)

# return s\_list

# #Go through each child and add items into the new list

# else:

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

# bt\_to\_slist(T.child[i],s\_list)

# s\_list.append(T.item[i])

# 

# #Go to next child in the current node

# return bt\_to\_slist(T.child[len(T.item)],s\_list)

# #3. Return the minimum element in the tree at a given depth d.

# def min\_ele(T,d):

# if d > height(T) or d < 0:

# return "ERROR: Out of depth"

# 

# if d == 0:

# return T.item[0]

# 

# if T.isLeaf:

# return T.item[0]

# 

# return min\_ele(T.child[0],d-1)

# #4. Return the maximum element in the tree at a given depth d.

# def max\_ele(T,d):

# if d > height(T) or d < 0:

# return "ERROR: Out of depth"

# if d == 0:

# return T.item[-1]

# 

# if T.isLeaf:

# return T.item[-1]

# return max\_ele(T.child[-1],d-1)

# #5. Return the number of nodes in the tree at a given depth d.

# def num\_nodes(T,d):

# c = 0

# if d > height(T) or d < 0:

# return "ERROR: Out of depth"

# if T.isLeaf:

# c += len(T.item)

# if d == 0:

# return len(T.item)

# 

# #Traverse through list and add one to the count until d is 0

# else:

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

# c += num\_nodes(T.child[i],d-1)

# return c

# #6. Print all the items in the tree at a given depth d.

# def print\_at\_d(T,d):

# if d > height(T) or d < 0:

# return ("ERROR: Out of depth")

# #Print current item in tree and do not add new line

# if d == 0:

# print(T.item, end=' ')

# if T.isLeaf:

# return

# 

# #Traverse through the current node and print each node

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

# print\_at\_d(T.child[i],d-1)

# #7. Return the number of nodes in the tree that are full.

# def full\_nodes(T):

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

# return 1

# else:

# c = 0

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

# c+= full\_nodes(T.child[i])

# return c

# #8. Return the number of leaves in the tree that are full.

# def full\_leaves(T):

# #Check if t is currently a leaf and if exceeds max items then return 1

# if T.isLeaf:

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

# return 1

# else:

# return 0

# 

# #Traverse through each node and add one to the count

# else:

# c = 0

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

# c += full\_leaves(T.child[i])

# return c

# #9. Given a key k, return the depth at which it is found in the tree, of -1 if k is not in the tree.

# def k\_at\_d(T,d,k):

# if k in T.item: #If k in current depth or root is k then return the depth

# return d

# if T.isLeaf: #Since the previous check was false then return not found

# return "Not Found"

# else: #Use FindChild to check subtree if k is in it and if not found then add 1 to d

# return k\_at\_d(T.child[FindChild(T,k)], d+1, k)

# #################################################################################################

# #MAIN

# #################################################################################################

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

# #L = [6,3,16,11,7,17,14,8,5,19,15,1,2,4,18,13,9,20,10,12,21]

# #Try and catch errors if input is not an integer

# try:

# n = int(input("Enter the size of the list \n"))

# L = random.sample(range(n+100), n) #Creates random list

# T = BTree()

# #Build b tree from the generated list and insert with b tree insert and print the order

# for i in L:

# print('Inserting',i)

# Insert(T,i)

# PrintD(T,'')

# #Print(T)

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

# #d = int(input("Enter the depth \n"))

# #k = int(input("Enter the key \n"))

# d = 0

# k = 11

# 

# start = timeit.default\_timer()

# print("Height: " + str(height(T)))

# stop = timeit.default\_timer()

# print('Time: ', stop - start)

# 

# start = timeit.default\_timer()

# print("Min Element in depth " + str(d) + ": " + str(min\_ele(T,d)))

# stop = timeit.default\_timer()

# print('Time: ', stop - start)

# start = timeit.default\_timer()

# print("Max Element in depth " + str(d) + ": " + str(max\_ele(T,d)) + "\n")

# stop = timeit.default\_timer()

# print('Time: ', stop - start)

# 

# start = timeit.default\_timer()

# print("B-Tree to Sorted List: " + str(bt\_to\_slist(T,[]))+ "\n")

# stop = timeit.default\_timer()

# print('Time: ', stop - start)

# 

# start = timeit.default\_timer()

# print("Num of nodes in depth " + str(d) + ": " + str(num\_nodes(T,d))+ "\n")

# stop = timeit.default\_timer()

# print('Time: ', stop - start)

# start = timeit.default\_timer()

# print("Print items in depth "+ str(d) + ": ")

# print\_at\_d(T,d)

# stop = timeit.default\_timer()

# print('Time: ', stop - start)

# print("\n")

# 

# start = timeit.default\_timer()

# print("Num of full Nodes: " + str(full\_nodes(T))+ "\n")

# stop = timeit.default\_timer()

# print('Time: ', stop - start)

# 

# start = timeit.default\_timer()

# print("Num of full Leaves: " + str(full\_leaves(T))+ "\n")

# stop = timeit.default\_timer()

# print('Time: ', stop - start)

# 

# start = timeit.default\_timer()

# print("Key " + str(k) + " at depth: " + str(k\_at\_d(T,0,k)))

# stop = timeit.default\_timer()

# print('Time: ', stop - start)

# except IndexError:

# print("Invalid input")

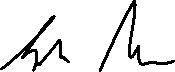
# except ValueError:

# print("Invalid input")

# Academic Honesty

“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.”

Name: Kimberly Morales



Signature: