**Lab #4**

CS 2302

Ana Luisa Mata Sánchez

# Introduction

This lab computes the following B-Tree operations:

1. Compute the height of the tree
2. Extract the items in the B-tree into a sorted list.
3. Return the minimum element in the tree at a given depth d.
4. Return the maximum element in the tree at a given depth d.
5. Return the number of nodes in the tree at a given depth d.
6. Print all the items in the tree at a given depth d.
7. Return the number of nodes in the tree that are full.
8. Return the number of leaves in the tree that are full.
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.

# Proposed solution design and implementation

**Module 1 – Height**

Using the height method provided by Dr. Fuentes, the method was called and returned the height. It uses recursion to go through the first child of each node and adds one to the height. When it reaches a leaf it reaches the end and counts it as 0.

**Module 2 – B-tree to sorted list**

The method bTreeToList receives a B-Tree and a list. If the tree is a leaf just add all of the items to the list. If it is not a leaf you have to go to that nodes leaf, add that leaf, add the parent and after you do this for all items you have to add the last child node that is located in index length of the items.

**Module 3** – **The minimum element in the tree at a given depth d**

The method MinAtDepth receives a B-Tree and a depth. First it checks if the given depth is valid by checking that it is not more than the height and that it is not less than zero, if it is invalid it return infinity. After that it iterates through the leftmost child (that is the minimum element at a certain depth), subtracting one from the depth until it equals to 0. When the depth equals to zero it returns the leftmost item in that node.

**Module 4** – **The maximum element in the tree at a given depth d**

The method MaxAtDepth receives a B-Tree and a depth. First it checks if the given depth is valid by checking that it is not more than the height and that it is not less than zero, if it is invalid it return infinity. After that it iterates through the rightmost child (that is the maximum element at a certain depth), subtracting one from the depth until it equals to 0. When the depth equals to zero it returns the rightmost item in that node.

**Module 5** – **Number of nodes in the tree at a given depth d**

The method NumNodesAtDepth receives a B-Tree and a depth. First it checks if the given depth is valid by checking that it is not more than the height and that it is not less than zero, if it is invalid it return infinity. The method iterates through each of the children nodes and adds them using the count variable. Each time it goes to a child it subtracts one from the depth. When the depth reaches zero or the desired depth it returns one to be added to count. If it reaches a leaf and the depth is not 0, it will count not count it.

**Module 6** – **Print all the items in the tree at a given depth d.**

The method PrintAtDepthD receives a B-Tree and a depth. It iterates through the tree by going to each of the children, when it goes to a child it subtracts one from the depth. If the depth is 0 it prints all of the items in the node.

**Module 7** – **Return the number of nodes in the tree that are full**

The method FullNodes receives a B-Tree. It first checks if the current node is full, if it is it returns 1. If it reaches a leaf that was not full, it returns 0. It iterates by going through each of the children of a node and adds whatever they return to the count variable.

**Module 8** – **Return the number of leaves in the tree that are full.**

The method FullLeaves receives a B-Tree. It first checks if the current node is full and if it is a leaf, if it is it returns 1. If it reaches a leaf that was not full, it returns 0. It iterates by going through each of the children of a node and adds whatever they return to the count variable.

**Module 9** – **Given a key k, return the depth at which it is found in the tree**

The method FindDepth receives a B-Tree and a key to find. If it finds k in the array of items, it returns 0 (to start counting from that node up if it is not the root). If it did not find the item and it has reached a leaf it returns -1. It iterates through the tree using Dr. Fuente’s FindChild method that, as he describes it, determines value of c, such that k must be in subtree T.child[c], if k is in the BTree. The value returned by calling the method with the child that may contain k is stored in the variable d. If at any point d is -1 the method immediately returns -1. If this does not happen it will keep adding one to the depth until it finds the item.

# Experimental results

List representation of T, the B-Tree used in the following methods:

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

|  |  |  |
| --- | --- | --- |
| **Method call** | **Output** | **Analytical running time** |
| print("########### Height ##########")  print(**height**(T))  print("") | ####### Height ##########  2 | O(n) |
| print("####### Tree to sorted list ########")  A = list()  **btreeToList**(T,A)  print(A)  print("") | ####### Tree to sorted list #######  [1, 2, 3, 4, 5, 6, 10, 11, 20, 30, 40, 45, 50, 60, 70, 80, 90, 100, 105, 110, 115, 120, 200] | O(n) |
| print("###### Minimum element at depth d #######")  print(**MinAtDepth**(T,**2**))  print("") | ##### Minimum element at depth d #######  1 | O(n) |
| print("###### Minimum element at depth d #######")  print(**MinAtDepth**(T,**3**))  print("") | ##### Minimum element at depth d #####  inf | O(n) |
| print("###### Minimum element at depth d #######")  print(**MinAtDepth**(T,**1**))  print("") | ##### Minimum element at depth d #####  3 | O(n) |
| print("##### Maximum element at depth d #####")  print(**MaxAtDepth**(T,**2**))  print("") | ##### Maximum element at depth d #####  200 | O(n) |
| print("##### Maximum element at depth d #####")  print(**MaxAtDepth**(T,**1**))  print("") | ##### Maximum element at depth d #####  110 | O(n) |
| print("##### Maximum element at depth d #####")  print(**MaxAtDepth**(T,**3**))  print("") | ##### Maximum element at depth d #####  inf | O(n) |
| print("##### Number of nodes at depth d #####")  print(**NumNodesAtDepth**(T,**2**))  print("") | ##### Number of nodes at depth d #####  7 | O(n) |
| print("##### Number of nodes at depth d #####")  print(**NumNodesAtDepth**(T,**3**))  print("") | ##### Number of nodes at depth d #####  inf | O(n) |
| print("##### Number of nodes at depth d #####")  print(**NumNodesAtDepth**(T,**1**))  print("") | ##### Number of nodes at depth d #####  2 | O(n) |
| print("##### Print all elements at depth d #####")  **PrintAtDepthD**(T,**2**)  print(" ") | ##### Print all elements at depth d #####  1 2 4 5 6 11 20 40 45 50 70 80 100 105 115 120 200 | O(n) |
| print("##### Print all elements at depth d #####")  **PrintAtDepthD**(T,**3**)  print(" ") | ##### Print all elements at depth d ##### | O(n) |
| print("##### Print all elements at depth d #####")  **PrintAtDepthD**(T,**1**)  print(" ") | ##### Print all elements at depth d #####  3 10 30 90 110 | O(n) |
| print("##### Full node count #####)  print(**FullNodes**(T))  print("") | ##### Full node count #####  0 | O(n) |
| print("##### Full leaves count #####")  print(**FullLeaves**(T))  print("") | ##### Full leaves count #####  0 | O(n) |
| print("##### Find depth of an item #####")  print(**FindDepth**(T,**0**)) | ##### Find depth of an item #####  -1 | O(n) |
| print("##### Find depth of an item #####")  print(**FindDepth**(T,**60**)) | ##### Find depth of an item #####  0 | O(n) |
| print("##### Find depth of an item #####")  print(**FindDepth**(T,**90**)) | ##### Find depth of an item #####  1 | O(n) |
| print("##### Find depth of an item #####")  print(**FindDepth**(T,**11**)) | ##### Find depth of an item #####  2 | O(n) |

# Conclusion

This lab was helpful in understanding how to traverse and use a B-Tree.

**The code contained in this report and the report itself were created and written by me, unless stated otherwise.**

**Digital signature: Ana Luisa Mata Sánchez**

# Appendix

|  |  |
| --- | --- |
|  | # Author: Ana Luisa Mata Sánchez |
|  | # Course: CS2302 |
|  | # Assignment: Lab #4 |
|  | # Instructor: Olac Fuentes |
|  | # Description: B tree operations |
|  | # T.A.: Anindita Nath |
|  | # Last modified: 03/15/2019 |
|  | # Purpose: Compute the height of the tree, extract the B-tree into a sorted list, |
|  | # minimum and maximum element in the tree at depth, number of nodes in the tree at depth d, |
|  | # print all the items at depth d,number of nodes nad leaves that are full, and return the depth of a given item. |
|  |  |
|  | # Code to implement a B-tree |
|  | # Programmed by Olac Fuentes |
|  | # Last modified February 28, 2019 |
|  |  |
|  | 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 |
|  |  |
|  | 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 0 |
|  | 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) |
|  |  |
|  | ###################################### START OF MY CODE ###################################### |
|  |  |
|  | #Extract the items in the B-tree into a sorted list. |
|  | def btreeToList(T,A): |
|  | if T.isLeaf: |
|  | for i in range(len(T.item)): |
|  | A.append(T.item[i]) |
|  | else: |
|  | for i in range(len(T.item)): |
|  | #append the left part |
|  | btreeToList(T.child[i],A) |
|  | #append the parent |
|  | A.append(T.item[i]) |
|  | #append the right |
|  | btreeToList(T.child[len(T.item)],A) |
|  | return A |
|  |  |
|  | #Return the minimum element in the tree at a given depth d |
|  | def MinAtDepth(T,d): |
|  | #check that the depth is valid |
|  | if d>height(T) or d<0: |
|  | return math.inf |
|  | #if you reach the end, the leftmost element is the minimum |
|  | if d ==0 : |
|  | return T.item[0] |
|  |  |
|  | #iterate |
|  | return MinAtDepth(T.child[0],d-1) |
|  |  |
|  | #Return the maximum element in the tree at a given depth d |
|  | def MaxAtDepth(T,d): |
|  | #check that the depth is valid |
|  | if d>height(T) or d<0: |
|  | return math.inf |
|  | #if you reach the end, the rightmost element is the maxuimum |
|  | if d ==0 : |
|  | return T.item[-1] |
|  |  |
|  | #iterate |
|  | return MaxAtDepth(T.child[len(T.child)-1],d-1) |
|  |  |
|  | #Return the number of nodes in the tree at a given depth d |
|  | def NumNodesAtDepth(T,d): |
|  | #check that the depth is valid |
|  | if d>height(T) or d<0: |
|  | return math.inf |
|  | #if you reach intended depth, add 1 |
|  | if d == 0 : |
|  | return 1 |
|  | #if not and you reached the end of the tree, add 0 |
|  | if T.isLeaf: |
|  | return 0 |
|  | count = 0 |
|  | for i in range (len(T.child)): |
|  | #iterates |
|  | count += NumNodesAtDepth(T.child[i],d-1) |
|  | return count |
|  |  |
|  | #Print all the items in the tree at a given depth d |
|  | def PrintAtDepthD(T,d): |
|  | #if you reach intended depth, print all items in the node |
|  | if d == 0 : |
|  | for j in range (len(T.item)): |
|  | print(T.item[j], end=" ") |
|  | #iterates |
|  | for i in range (len(T.child)): |
|  | PrintAtDepthD(T.child[i],d-1) |
|  |  |
|  | #Return the number of nodes in the tree that are full. |
|  | def FullNodes(T): |
|  | #if it is full, add 1 |
|  | if IsFull(T): |
|  | return 1 |
|  | #if you reach the end, add 0 |
|  | if T.isLeaf: |
|  | return 0 |
|  | count = 0 |
|  | for i in range (len(T.child)): |
|  | #iterate |
|  | count += FullNodes(T.child[i]) |
|  | return count |
|  |  |
|  | #Return the number of leaves in the tree that are full |
|  | def FullLeaves(T): |
|  | #if it is full and it is a leaf, add 1 |
|  | if IsFull(T) and T.isLeaf: |
|  | return 1 |
|  | #if you reach the end, add 0 |
|  | elif T.isLeaf: |
|  | return 0 |
|  | count = 0 |
|  | for i in range (len(T.child)): |
|  | #add all full nodes |
|  | count += FullNodes(T.child[i]) |
|  | return count |
|  |  |
|  | #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 FindDepth(T,k): |
|  | #if you find the item, that depth is 0 |
|  | if k in T.item: |
|  | return 0 |
|  | #if you reach the end without finding, return -1 |
|  | if T.isLeaf: |
|  | return -1 |
|  |  |
|  | #find the child you should call and add |
|  | d = FindDepth(T.child[FindChild(T,k)],k) |
|  |  |
|  | #if it did not find it, return -1 |
|  | if d == -1: |
|  | return -1 |
|  | #if it is still going count the depth |
|  | return d + 1 |
|  |  |
|  |  |
|  |  |
|  | 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) |
|  |  |
|  | #1 |
|  | print("################# Height #################") |
|  | print(height(T)) |
|  | print("") |
|  | #2 |
|  | print("################# Tree to sorted list #################") |
|  | A = list() |
|  | btreeToList(T,A) |
|  | print(A) |
|  | print("") |
|  | #3 |
|  | print("################# Minimum element at depth d #################") |
|  | print(MinAtDepth(T,2)) |
|  | print("") |
|  | #4 |
|  | print("################# Maximum element at depth d #################") |
|  | print(MaxAtDepth(T,2)) |
|  | print("") |
|  | #5 |
|  | print("################# Number of nodes at depth d #################") |
|  | print(NumNodesAtDepth(T,2)) |
|  | print("") |
|  | #6 |
|  | print("################# Print all elements at depth d #################") |
|  | PrintAtDepthD(T,2) |
|  | print(" ") |
|  | #7 |
|  | print("################# Full node count #################") |
|  | print(FullNodes(T)) |
|  | print("") |
|  | #8 |
|  | print("################# Full leaves count #################") |
|  | print(FullLeaves(T)) |
|  | print("") |
|  | #9 |
|  | print("################# Find depth of an item #################") |
|  | print(FindDepth(T,10)) |