Laila Romero

CS 2302 Data Structures

March 26, 2019

Lab 4

* Introduction

For this lab we were trying to manipulate B-Trees. We must compute the height of the tree, extract the items in to a sorted list, return the minimum and maximum elements at a given depth, return the number of nodes at a given depth, print all the items at a given depth, return the number of nodes and leaves that are full, and given a key return the depth of the key.

• Proposed Solution Design and Implementation

For computing the height, you first check if the node is a leaf and if it is you return zero. If not, then you make a recursive call for to check if there is a child to the right and then return that height plus one for the root.

For finding the maximum and minimum elements in the B-Tree you check if the given depth is zero and if it is you return the item in the depth. If not, you return the left child for the maximum and then return the right child for the minimum elements in the depth.

For returning the number of nodes at a given depth, you first check the given depth and if it is zero then you return the length of the item. Then you check if the node is not a leaf then you return the length of the nodes of the left and right children.

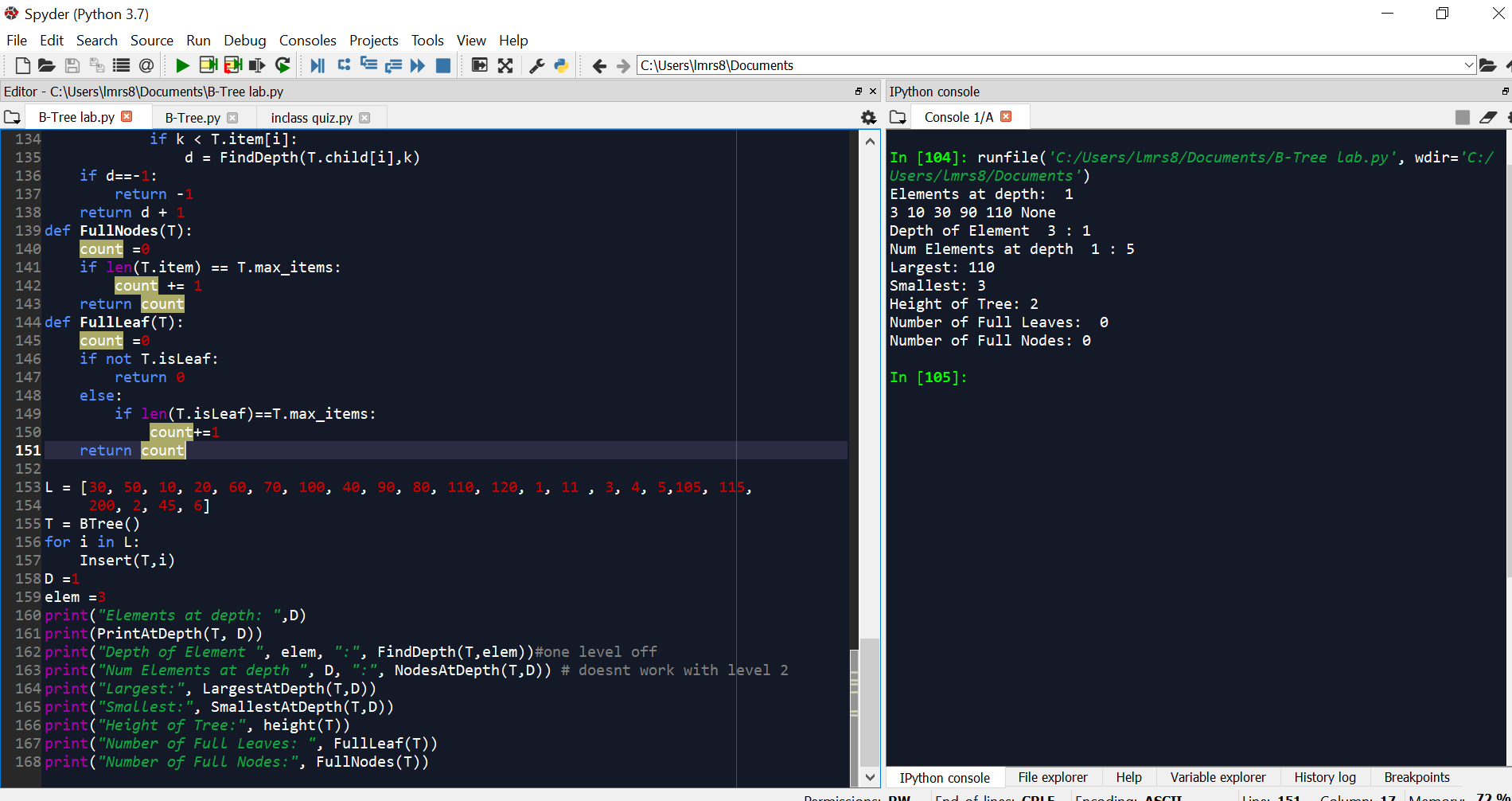
For printing all the elements at a given depth, you first check if the depth is zero and if it is then you use a for loop to traverse the elements in the depth then print each item. If the depth is not zero, then you take use a for loop to traverse the items in the depth by calling the right and left children and printing the items.

For returning full nodes you first check if the length of the item is equal to the max number of items for the B-tree, if it is equal then you increase the counter by one and return the counter. If it is not equal, then you return 0. For the full leaves you first check if the item is a leaf if not then you check if the left and right children are leaves, if none of them are leaves then you return 0. If they are leaves, then you check if the length of the leaves equal to the max items for the B-tree. If they are equal then you increase the counter by one and return the counter at the end.

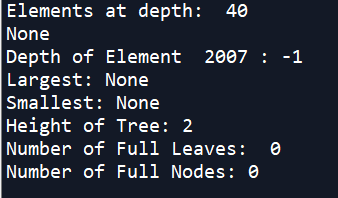
For returning a depth for a given key you first check is the item is in T.item and if it is then you return 0. If not then you return -1. If the element is greater than the last item in the node then the depth is equal to the depth of the children of the item. If the item is not in the tree then return -1.

• Experimental results

For the computing the height, the code worked well for different height trees that I input. It also worked for computing the max and min elements in the given depth of the tree if the depth was within the height of the tree. All of the methods worked correctly when tested with existing depths and existing elements, the result can be seen below.



I also checked instances where the element and depth did not exist in the tree to see if the maximum and minimum methods would work and they returned none. The method to print the elements of the tree also returned none as the depth did not exist. The results can be seen below.



• Conclusion

In conclusion, I learned how to traverse through and manipulate a B-tree through recursive and iterative methods. I had never heard about B-trees before so being able to learn all the properties they hold and all there is to them was very interesting. Although there was one method I could not get to work I was still able to understand the logic behind all the methods and how to properly check each node and leaf to traverse through the entire B-tree.

• Appendix – Source codes

|  |
| --- |
| class BTree(object): |
|  | 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): |
|  | for i in range(len(T.item)): |
|  | if k < T.item[i]: |
|  | return i |
|  | return len(T.item) |
|  |  |
|  | def InsertInternal(T,i): |
|  | 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): |
|  | #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): |
|  | if k in T.item: |
|  | return T |
|  | if T.isLeaf: |
|  | return None |
|  | return Search(T.child[FindChild(T,k)],k) |
|  |  |
|  | def Print(T): |
|  | 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): |
|  | 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) |
|  | def LargestAtDepth(T,d): |
|  | if d==0: |
|  | return T.item[-1] |
|  | if T.isLeaf: |
|  | return None |
|  | return LargestAtDepth(T.child[-1],d-1) |
|  | def SmallestAtDepth(T,d): |
|  | if d==0: |
|  | return T.item[0] |
|  | if T.isLeaf: |
|  | return None |
|  | return SmallestAtDepth(T.child[0],d-1) |
|  |  |
|  | def PrintAtDepth(T,d): |
|  | if d==0: |
|  | for t in T.item: |
|  | print(t,end=' ') |
|  | if not T.isLeaf: |
|  | for i in range(len(T.child)): |
|  | PrintAtDepth(T.child[i],d-1) |
|  | def NodesAtDepth(T,d): |
|  | if d==0: |
|  | return len(T.item) |
|  | if not T.isLeaf: |
|  | return NodesAtDepth(T.child[0],d-1)+NodesAtDepth(T.child[-1],d-1) |
|  |  |
|  | def FindDepth(T, k): # Returns the depth of item k in b-tree with root T, or -1 if |
|  | # k is not in the tree |
|  | if k in T.item: |
|  | return 0 |
|  | if T.isLeaf: |
|  | return -1 |
|  | if k>T.item[-1]: |
|  | d = FindDepth(T.child[1],k) |
|  | else: |
|  | for i in range(len(T.item)): |
|  | if k < T.item[i]: |
|  | d = FindDepth(T.child[i],k) |
|  | if d==-1: |
|  | return -1 |
|  | return d + 1 |
|  | def FullNodes(T): |
|  | count =0 |
|  | if len(T.item) == T.max\_items: |
|  | count += 1 |
|  | return count |
|  |  |
|  | def FullLeaf(T): |
|  | count =0 |
|  | if not T.isLeaf: |
|  | return 0 |
|  | else: |
|  | if len(T.isLeaf)==T.max\_items: |
|  | count+=1 |
|  | return count |
|  |  |
|  | 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: |
|  | Insert(T,i) |
|  |  |
|  | D =1 |
|  | elem =3 |
|  | print("Elements at depth: ",D) |
|  | print(PrintAtDepth(T, D)) |
|  | print("Depth of Element ", elem, ":", FindDepth(T,elem)) |
|  | print("Num Elements at depth ", D, ":", NodesAtDepth(T,D)) |
|  | print("Largest:", LargestAtDepth(T,D)) |
|  | print("Smallest:", SmallestAtDepth(T,D)) |
|  | print("Height of Tree:", height(T)) |
|  | print("Number of Full Leaves: ", FullLeaf(T)) |
|  | print("Number of Full Nodes:", FullNodes(T)) |