Diego Quinones
80582918

CS2302 - Data Structures Spring 2019
Lab 4 B-trees

Report

This lab consisted of working with binary trees, which are a little bit different of the previous topic which were binary search trees. We were assigned 9 problems which on average were really easy thanks to the previous knowledge.

1- To compute the height I only had to count every level until I got to a leaf.

2- Used the print method but instead of printing it appends into a list

3- Recursive calls d times, moves to the left subtrees and returns the first value on the T.item

4- As the previous problem, but moving to the right and returning the last value on T.item

5- Recursive calls d times and when it gets to that point it counts every node

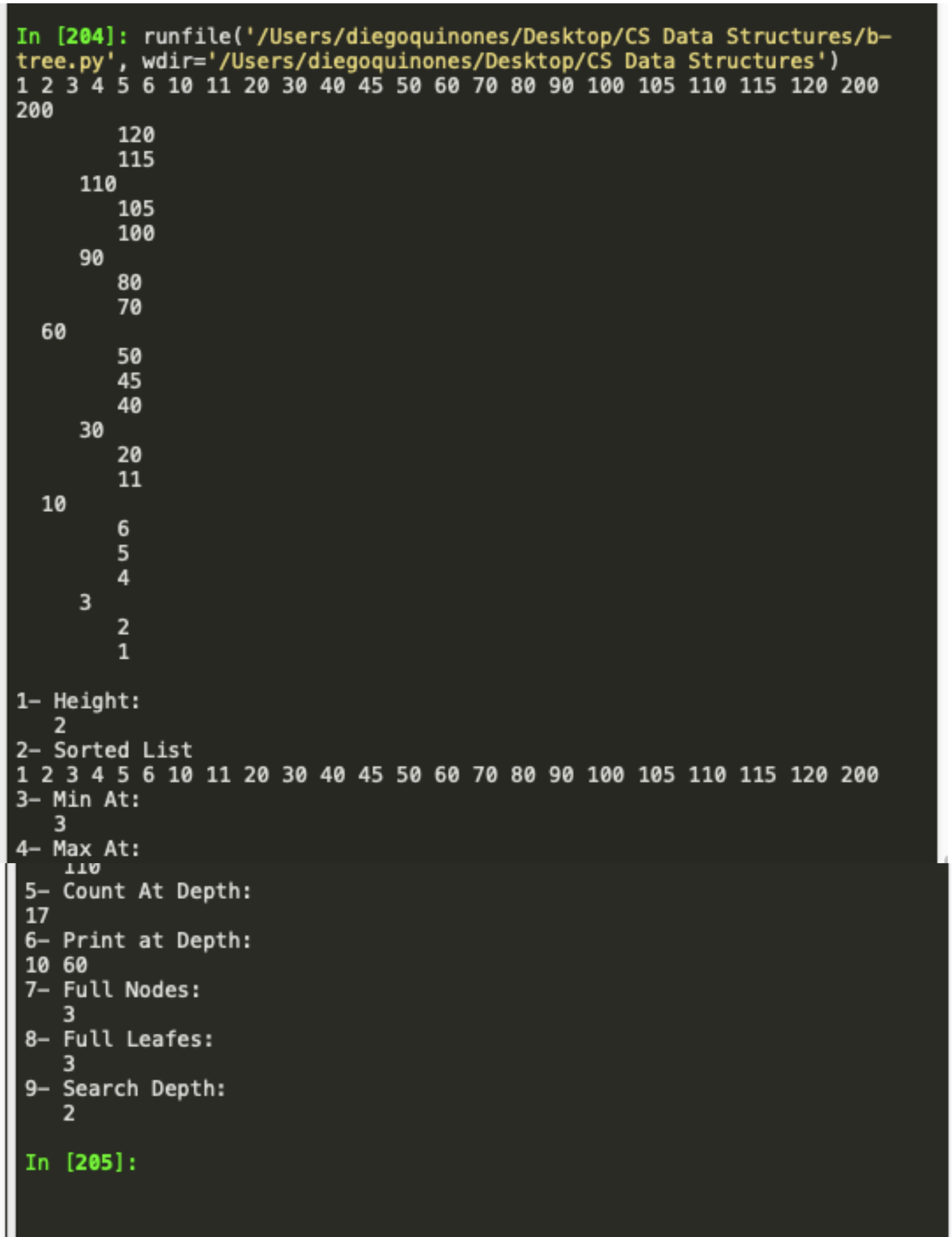
6- Used a for loop to print every value in T.item, when d is equal to 0, every recursive call diminishes d by 1.

7- I used the print method as a base because it is the one that traverses, but instead of printing I compared using the length of T.item with max_items and also if not T.isLeaf that way it only counts the ones that aren't leaves.

8- The same as past problem but instead it only counts if T.isLeaf

9- Used comparisons to make it move, if the value is larger than the last value on T.item, move to the right or, if it is smaller than the smallest one, move to the left, etc. If none of those conditions are fulfilled then it would use a for loop to traverse the T.item.

Screenshots



```
In [204]: runfile('/Users/diegoquinones/Desktop/CS Data Structures/b-
tree.py', wdir='/Users/diegoquinones/Desktop/CS Data Structures')
1 2 3 4 5 6 10 11 20 30 40 45 50 60 70 80 90 100 105 110 115 120 200
200
        120
        115
     110
        105
        100
     90
        80
        70
  60
        50
        45
        40
     30
        20
        11
  10
        6
        5
        4
     3
        2
        1

1- Height:
   2
2- 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
3- Min At:
   3
4- Max At:
   110
5- Count At Depth:
17
6- Print at Depth:
10 60
7- Full Nodes:
   3
8- Full Leafes:
   3
9- Search Depth:
   2

In [205]:
```

Source Code

```
# Code to implement a B-tree
# Programmed by Diego Quinones
# Last modified March 18, 2019

class BTree(object):
    # Constructor
    def __init__(self,item=[],child=[],isLeaf=True,max_items=3):
        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):
    #Compute the height of the tree
    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)

def SortedList(T,L):
    #Extract the items in the B-tree into a sorted list.
    #checks if its at the bottom
    if T.isLeaf:
        for t in T.item:
            #adds value to the list
            L.append(t)
    else:
        for i in range(len(T.item)):
            #recursive call for when the values is not a leaf
            SortedList(T.child[i],L)
            L.append(T.item[i])
        SortedList(T.child[len(T.item)],L)

def MinAt(T,k):
    #Return the minimum element in the tree at a given depth d
    if k is 0:
        # it got to the desired depth
        return T.item[0]
    if T.isLeaf:
        #if it got to the bottom without being at depth k
        return -1
    #it is called again but with depth being reduced as it gets closer
    return MinAt(T.child[0],k-1)

def MaxAt(T,k):
    #Return the maximum element in the tree at a given depth d.
    if k is 0:
        # it got to the desired depth
        return T.item[-1]
    if T.isLeaf:
        #if it got to the bottom without being at depth k
        return -1
    #it is called again but with depth being reduced as it gets closer
    return MaxAt(T.child[-1],k-1)

def PrintAtDepth(T,k):
    # Prints items in tree in ascending order
    if k==0:
        for i in range(len(T.item)):
            print(T.item[i],end=' ')
    if T.isLeaf:
        return
    else:
        for i in range(len(T.item)):
            PrintAtDepth(T.child[i],k-1)
        PrintAtDepth(T.child[len(T.item)],k-1)

def CountAtDepth(T,k):
    # Prints items in tree in ascending order
    a=0
    if k==0:
        #counts full leafes
        for i in range(len(T.item)):
            a=a+1
        return a
    else:
        for i in range(len(T.child)):
          #adds up every full leaf
           a=a+CountAtDepth(T.child[i],k-1)
    return a
```

```
def SearchDepth(T,k):
    # Given a key k, return the depth at which it is found in the tree, of -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]:
        return 1+SearchDepth(T.child[-1],k)
    else:
        return 1+SearchDepth(T.child[0],k)

def PrintLeafesFull(T):
    #Return the number of leaves in the tree that are full
    # counting value
    a=0
    if T.isLeaf:
        #counts full leafes
        if len(T.item) is T.max_items:
            return 1
    else:
        for i in range(len(T.child)):
          #adds up every full leaf
           a=a+PrintLeafesFull(T.child[i])
    return a

def PrintNodesFull(T):
    # Return the number of nodes in the tree that are full.
    #counting value
    a=0
    if T is None:
        return
    if not T.isLeaf:
        for i in range(len(T.child)):
            a=a+PrintNodesFull(T.child[i])
    if len(T.item)==T.max_items:
        #counts full nodes
        a=a+1
    return a




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)
Print(T)
PrintD(T,' ')
print()
print('1- Height:')
print('  ',height(T))
L1=[]
print('2- Sorted List',end=' ')
print()
SortedList(T,L1)
for i in L1:
    print(i,end=' ')

print()
print('3- Min At:')
print('  ',MinAt(T,1))
print('4- Max At:')
print('  ',MaxAt(T,1))
print('5- Count At Depth:')
print(CountAtDepth(T,2))
print('6- Print at Depth:')
PrintAtDepth(T,0)
print()
print('7- Full Nodes:')
print('  ',PrintNodesFull(T))
print('8- Full Leafes:')
print('  ',PrintLeafesFull(T))
print('9- Search Depth:')
print('   ',SearchDepth(T,200))
```

Academic Honesty Certification

“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.”

-Diego Quinones