**Question 1**

**Objective**

The Objective of this question is to be able to find the height of a b-tree assuming its always going to be balanced.

**Proposed solution**

For this question I know that I have to iterate through the tree and count each level if the tree keeps going down. The only way I can stop counting is when I reach a leaf node in my tree; when I reach the leaf node I have to return 0 since it’s the end of the tree. The way I plan to traverse this is by using recursion.

**Implementation**

To begin the implementation, I created a method called height which the parameters are the b-tree which I will be using. Inside the method, I need to make a base case which is going to be if my tree node is a leaf and if the node that is being looked at is a leaf then I have to return 0. If my tree doesn’t meet the base case then I need to traverse the tree until I reach a leaf node, but the only way to do that is by calling the method recursively but passing the left side of the tree to traverse (works with either side). Also, every time I call the method again I want to return the line of code and add 1 to it in order to keep count of how many levels the tree has.

**Tracing**

For this tracing, I will be using the tree provided from the example program of b-trees in class. The tree contains the values of 30, 50, 10, 20, 60, 70, 100, 40, 90, 80, 110, 120, 1, 11 , 3, 4, 5,105, 115, 200, 2, 45, 6. When passing the tree into the height method, I initially know that 60 is the number that will be at the top/root of the tree and now inside the height method the base case checks if the value 60 is a leaf node and since 60 is the root of my tree, I can see that it has children. Since this node has children the method proceeds to return 1 and call the method recursively; when calling the method with recursion the parameter passed is the Tree while accessing the child node to the left direction. Now the point that is being looked at is the value 3 since it’s the value that is in the left side, but it goes down in a depth of 1. Since the value 3 is not a leaf node in our tree the method will return the value 1 and call the method while passing the tree and traversing the tree to the left. The new value that is being looked at is 1 and since 1 is actually a leaf node the method will return 0 this time since a leaf node is where the tree ends. Now going back in the method there were two recursive calls that added 1 and since 1+1 = 2, 2 is the height of the tree is 2.

A close up of a logo

Description automatically generated

A picture containing clipart

Description automatically generated  
O(1)

**Question 2**

**Objective**

The objective of this method is to create a code that will allow me to transfer the values from the b-tree to a list.

**Proposed solution**

For this question, I wanted to approach it like how I did it in lab #3 when I transferred the values from a BST to an empty list. I know that I need an empty list where I will be storing the values from the b-tree into the list and the way to do that is to traverse the b-tree and store the values that are in a node of a tree. However, to traverse I need to go left and right of the tree by accessing the child nodes of the tree, but if I want to access them I need to use a for loop to access each node and element that the node may have. Also, when obtaining the values from the node I want to append the elements into my empty list.

**Implementation**

To implement my idea, I created a method which takes the parameters of the tree and an empty list. Inside the method, I made a base case that if the node that I’m currently looking at is a leaf then it would use a for loop which goes from the range of the items in the node. In the for loop is where the values from the tree will be appended into the list. The reason for the base case is to be able to take note of when we reach the end of the tree by checking if we reach a leaf node.

However, if the node that I’m looking at is not a leaf node then I would use an else statement that would also require the use of a for loop with the same parameters as before. However, inside the for loop I would immediately make a recursive call that will pass the child node at I which basically would allow me to move to the left side of my tree, also the recursive call needs to keep passing the list that I have. After that recursive call I append the values of the tree into my list as it was done in Lab #3; after appending the items from the left side of the tree I now need to traverse to the right side of the tree and the way I did that was to make a recursive call that passes the tree and goes to the child node on the right direction and the other thing that has to be passed is my list in order to keep appending the items from the tree into there. At the very end of the method I return the list, but when I return the list, it has to be outside the if-else statement.

**Tracing**

To trace this method, I will be using a smaller tree just to be able to explain the process better. The tree that I will be using is a smaller version of the original tree which is

L = [ 50, 10, 20, 60, 70, 100, 40, 90, 80] and I will be using an empty list called E which is going to be used to append the values from the tree to the list. Once these values get passed to the method BT\_ToList, the method begins to check if the node that its currently looking at is it’s a leaf. Since the node that we are looking at is 50, we can see that 50 is the root of the tree and since it’s the root then it has children. Since 50 is not a leaf node then it goes to the else and what happens is that it goes through a for loop that iterates through the range of the items in the node and since this node only has one item then it would iterate once. Inside the method there is a recursive call that passes the child node of the tree on the left side and as we can see the left side contains the values 10,20,40 in the left node. Since the recursive call occurred, the node that we are looking at is the left side node that has the values 10,20,40. The method begins checking if the node on the left side is a leaf and as we can see it is; since it’s a leaf node then it would be passed into a for loop that goes in the range of the amount of items of the node and since we have 3 items, then the for loop iterates 3 times. Inside the for loop, the method will append the values into the list, so what the list looks like now is 10,20,40 and since that node was a leaf node that means that there’s nothing else on the left side, which then we have to go back to when we made the recursive call with 50. Since the left side is done with then the next thing that happens is that the value 50 is appended to the list which now the list looks like 10,20,40,50 and since that’s the only value that we can append from the root, the method proceeds to make a recursive call that passes the child node to the right and passes the list. Once the right side is passed we know that the values that we are using are 70,80,90,100 and as we can see this side is a leaf, which means that it will pass the base case that is set up. Since our node has 4 items that means that the for loop will iterate 4 times and will append the remaining values, which now makes the finished list as 10,20,40,50,70,80,90,100.A close up of a piece of paper

Description automatically generated



O(1)

**Question 3**

**Objective**

The objective of this method is to create a code that will be able to find the smallest element in the b-tree at a specific depth.

**Proposed solution**

For this question, I want to approach it as any basic find the smallest value, which the basic part always involves traversing. In the method, I know that I have to traverse the b-tree in order to find the smallest value, but since this is a B-TREE I know that all my smallest values will always be on the left side, so the smallest value is in the left most side of the tree. But since I need to find the smallest value at a certain depth I need to keep track of what level I am in my tree.

**Implementation**

To begin implementing my idea, I began to write the signature of my method and I passed the parameters of the b-tree and the depth the will be looked at. To begin I made a value called h that will store the height of my current tree and the purpose of h is to compare the depth that we will be looking at in the base case. I made the base case be if the depth is greater than the height of my tree then my method returns -1, this is because if my depth is greater than my tree then I can’t have a value to return. Next I created an if statement that if the depth is equal to 0 then return the item of the tree on the left side. After making that if my depth doesn’t apply to any of those cases then I need to call the method recursively and pass the child of the tree in the left side and subtract 1 from the depth in order to be able to get the values from the depth we’re looking for.

**Tracing**

For this method, I will be tracing it by using the tree that contains the values [ 50, 10, 20, 70, 100, 40, 90, 80] and the depth that I will be using is 1. Once I pass those values into the method, the first thing my method does is that it gets the height of my tree which is 1 because it has a root and leaf nodes after. Now that I know the height, the method compares the depth to the height and since my depth is less than the height, then that means that I do have a depth of 1 and it doesn’t pass the base case. Since the base case is not applicable, the method continues and checks if the depth is equal to 0 and since our depth is 1 then its not equal. Since the depth didn’t pass any of the cases then a recursive call occurs, and it returns the method by passing the child of the tree at the left side and subtracts 1 from the depth. When the recursive call is made the tree moves to the left side where the item that the leaf node contains are 10,20,40 and our depth now became 0. When checking the base case, we see that the depth is still less than the height which causes the base case to not be applicable to this. Now the next thing that happens is that the method checks if the depth is equal to 0 and as we can see this is true. Since this case is applicable that means that the method will return the leftmost item in the b-tree at the depth of 1 and the item that is returned is 10 which is the smallest element in the b-tree.

A close up of a piece of paper

Description automatically generated



O(1)

**Question 4**

**Objective**

The objective of this method is to create a code that will be able to find the largest element in the b-tree at a specific depth.

**Proposed solution**

For this question, I want to approach it as before when I looked for the smallest value, but in this instance I’m looking for the largest value. In the method, I know that I have to traverse the b-tree in order to find the largest value, but since this is a B-TREE I know that all my largest values will always be on the right side, so the largest value is in the right most side of the tree. But since I need to find the largest value at a certain depth I need to keep track of what level I am in my tree.

**Implementation**

To begin implementing my idea, I began to write the signature of my method and I passed the parameters of the b-tree and the depth the will be looked at. To begin I made a value called h that will store the height of my current tree and the purpose of h is to compare the depth that we will be looking at in the base case. I made the base case be if the depth is greater than the height of my tree then my method returns -1, this is because if my depth is greater than my tree then I can’t have a value to return. Next I created an if statement that if the depth is equal to 0 then return the item of the tree on the right side. After making that if my depth doesn’t apply to any of those cases then I need to call the method recursively and pass the child of the tree in the right side and subtract 1 from the depth in order to be able to get the values from the depth we’re looking for.

**Tracing**

For this method, I will be tracing it by using the tree that contains the values [ 50, 10, 20, 70, 100, 40, 90, 80] and the depth that I will be using is 1. Once I pass those values into the method, the first thing my method does is that it gets the height of my tree which is 1 because it has a root and leaf nodes after. Now that I know the height, the method compares the depth to the height and since my depth is less than the height, then that means that I do have a depth of 1 and it doesn’t pass the base case. Since the base case is not applicable, the method continues and checks if the depth is equal to 0 and since our depth is 1 then it’s not equal. Since the depth didn’t pass any of the cases then a recursive call occurs, and it returns the method by passing the child of the tree at the right side and subtracts 1 from the depth. When the recursive call is made the tree moves to the right side where the item that the leaf node contains are 10,20,40 and our depth now became 0. When checking the base case, we see that the depth is still less than the height which causes the base case to not be applicable to this. Now the next thing that happens is that the method checks if the depth is equal to 0 and as we can see this is true. Since this case is applicable that means that the method will return the rightmost item in the b-tree at the depth of 1 and the item that is returned is 100 which is the smallest element in the b-tree.

A close up of a piece of paper

Description automatically generated



O(1)

**Question 5**

**Objective**

For this method my objective is to make a code that will find the total amount of nodes at a certain depth only.

**Proposed Solution**

For this question I know that I need to use a counter in order to keep count of the number of nodes. Also, I want to keep the base case from the min and max value methods which is where I compare the depth I’m using and if the depth is greater than my tree height then I return -1. Once I find the depth I’m looking for, I want to count all the values in that depth . if I don’t find the depth then I want to move on to the child nodes until I get to the depth.

**Implementation**

To begin implementing my idea I made the method signature and passed the b-tree and the depth I’ll be looking at in my tree. In my method, I declare the variable count to 0 and h to the height of my tree. Now with those declared I need to make a base case that checks if my depth is greater than the height of my tree then it returns -1. I then made a second if statement that if my depth is equals to 0 then I return the length of the items I have in that specific node. However, if I don’t find the depth I needed to make an else statement that will go through a for loop by the range of the items in the node. Inside the for loop I recursively call the method and pass the child nodes of the tree and decreases the depth by 1, also when I call the method I add the count value from the resulting counts of the method. Outside the for loop, I return the count and the recursive call of the method by passing the child node and the length of the items, but still decreasing the depth by 1.

**Tracing**

To trace this method, I will pass the tree of [ 50, 10, 20, 70, 100, 40, 90, 80] and a depth of 1. In my method I first have to check if the depth is greater than the height and as we can see, the height for the tree is 1 and are depth is 1, this base case then doesn’t apply to this depth since its not greater than 1. Next we check if d is equal to 0 and since our depth is 1 still that case doesn’t apply to us. The only option left is to use the else statement which makes us go in a for loop in a range of the items in the node and since our node is the root where we only have 1 value then the for loop only iterates once. In the for loop we add to our count once we do a recursive call so when we call the method we pass the child to the left side of the tree and decrease the depth by 1. Now the values that we look at is 10,20,40 and we have a depth of 0 now, which we know as the user that we will find the number of nodes. However, the program checks if our depth is greater than 1 , but since our depth is 0 then we don’t pass into that base case. Now in the second if statement we actually go inside the if since our depth is 0. What the if does, it returns the amount of numbers we have in the node which is 3 and going back to the recursive call the count now holds the value of 3. After that call, another recursive call occurs which add the count value to the result that is returned from the recursive call that passes the right side of the items of the child node and decreases the depth by 1; also, this recursive call is being returned. Since now we have the values 70,80,90,100 as the items we have in the node we continue to check if the depth is greater than 1 and since our depth is 0 again then we don’t go in the base case. However, we do go in the second if statement since our depth is 0 and we return the length of the node which is 4. Going back to the call we add the count with the length of the node we got, when we add the value that we return is 7 for the number of nodes we have.

A close up of a piece of paper

Description automatically generated

A picture containing clipart

Description automatically generated

**O(n)**

**Question 6**

**Objective**

The objective of this question is to create a method that prints all the elements of the b-tree at a certain depth.

**Proposed solution**

For this method I know that I just have to print the values at the specific depth, and I don’t need to return anything. However, I need to traverse the tree in order to get to the depth I’m looking for and if I need to print the left and right side of the tree at the certain depth. Also, If I don’t find the depth then I print none.

**Implementation**

To implement this idea, I made a method that passes the tree and the depth that we need, next I made h to be equal to the height of the tree. I made a base case that checks if the depth is greater than the height and if it is then I print none since we don’t have values at that depth. Next I made an if statement that if the depth is equal to 0 then I go in a for loop that iterates through the range of items that we may have in the node. Inside the for loop we print all the items that may be at the node. If we never pass the second if statement then I made an else statement were if my node is not a leaf then I go into a for loop which goes in the range of the number of items I have. Inside the for loop I make two recursive calls which one passes the tree to traverse to the left and another to traverse to the right side of the tree while decreasing the depth by 1.

**Tracing**

To trace this method I will use the tree [ 50, 10, 20, 70, 100, 40, 90, 80] and pass the depth of 1. When the tree and depth is passed, the variable h is made to store the height of the tree which is 1. We then go in the base case and check if the depth is greater than the height but since the depth is 1 and the height is 1 then we don’t go in the base case since they’re equal. Next the method checks if the depth is 0, but since this is false we go into the else statement. In the else if our tree is not a leaf then we iterate once in the for loop since the root is a size of 1. In the for loop we recursively call the method and pass the child node to the left side while we decrease the depth by 1. Now that the left node which is 10,20,40 is the one that we are using and the depth is 0 we can see that we will go into the if statement where the depth is equal to 0. Since we go into that if statement we iterate the for loop 3 times cause of the size of our node and we print all the values in that node which we print 10,20,40. Now we go back to the recursive calls and enter the second call where we pass the right side child node and decrease the depth by 1. Since we now have the right side which is 70,80,90,100 and the depth is 0, we can see that we will enter the second if statement where the depth is equal to 0. And since we enter that if statement we print all the values thanks to the for loop that iterates through the node. In the end we print 10 20 40 70 80 90 100.

A close up of a piece of paper

Description automatically generated



O(n)

**Question 7**

**Objective**

The objective of this question is to count the values of full nodes only, meaning that we ignore the leaf nodes.

**Proposed solutions**

The idea for this is to count the amount of full nodes and the way to know if a node is full is to use T.max\_items which is 5. We need to use this and compare it to the node to check if its full, if its not full then I need to traverse the tree to see if there are more full nodes.

**Implementation**

To implement the idea I made the method signature to pass the tree only and inside the method I declare the count variable to 0. Then I begin with a base case that if the length of my node is equal to the max items if that is true the count increases by 1. Then I made an else statement that uses a for loop to iterate through the length of the node, in the for loop I add count to a recursive call to the child nodes of the left side and it will add only if there are full nodes. Then outside the loop I add the count to the second recursive call which passes though the length of the items in the child node which would be the right. In the end of the method I return the count value.

**Tracing**

To trace I will use a tree of [30, 50, 10, 20, 60, 70, 100, 40, 90, 80, 110]. In the method I first check if the root is the max size and since there are only 2 values then doesn’t go in the if. We know that this isn’t a leaf node so we ignore the second method, we go in the else and iterate through the size of the node which is two and go into the recursive call that passes the left side of the child node. Since the next node is a leaf node we return 0 and go back to the original call, but we recursively call the method again and pass the right side of the tree. Since the right side is a leaf node we return 0. The only way there would be an increase in the count of full nodes is if I manage to fill the root with 5 values.



A screenshot of a cell phone

Description automatically generated

O(n)

**Question 8**

**Objective**

The objective of this question is to count the values of full leaves only, meaning that we ignore the nodes.

**Proposed solutions**

The idea for this is to count the amount of full leaves and the way to know if a leaf is full is to use T.max\_items which is 5. We need to use this and compare it to the node to check if its full, if its not full then I need to traverse the tree to see if there are more full leaves.

**Implementation**

To implement the idea I made the method signature to pass the tree only and inside the method I declare the count variable to 0. Then I begin with a base case that if the node is a leaf then I need a nested if that compares the size of the leaf to the max items and if they’re equal then count increases by 1. Then I made an else that iterates with a for loop with the size of the node and in the loop it has recursive calls where it traverses to the left side of the tree and adds the count. Outside the loop is another call that adds the count and traverses the right side of the tree. In the end it returns the count.

**Tracing**

To trace I will use a tree of [30, 50, 10, 20, 60, 70, 100, 40, 90, 80, 110,10,10,10,10,10,10]. In the method I first check if the root Is a leaf and since we know that its not we go into the else and traverse through the loop . we enter the recursive call and go to the left side of the tree and we can see the left is a leaf but is not equal to the max so it doesn’t add a count. We go back to the call and continue to the next recursive call and pass the rest of the leave nodes, where we see that we have only 1 full leaf node since the size of it is 5 which is the max.

A close up of a logo

Description automatically generatedA screenshot of a cell phone

Description automatically generated

O(n)

Academic dishonesty includes but is not limited to cheating, plagiarism and collusion. Cheating may involve copying from or providing information to another student, possessing unauthorized materials during a test, or falsifying data (for example program outputs) in laboratory reports. Plagiarism occurs when someone represents the work or ideas of another person as his/her own. Collusion involves collaborating with another person to commit an academically dishonest act. Professors are required to - and will - report academic dishonesty and any other violation of the Standards of Conduct to the Dean of Students.

|  |
| --- |
| # -\*- coding: utf-8 -\*- |
|  | """ |
|  | Created on Mon Mar 11 12:58:39 2019 |
|  | @author: Alexis Navarro |
|  | CS 2302 |
|  | Lab #4 |
|  | Purpose:The purpose of this lab is to be able to work with b-tree and understand how they work and to be able to |
|  | manipulate them |
|  | """ |
|  | #Given Code |
|  | 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 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+' ') |
|  |  |
|  |  |
|  | #My written code |
|  | #------------------------------------------------------------------------------ |
|  | #Question 1 |
|  | #Find the height of a Tree by using recursion |
|  | def height(T): |
|  | if T.isLeaf: |
|  | return 0 |
|  | return 1 + height(T.child[0]) |
|  | #------------------------------------------------------------------------------ |
|  |  |
|  | #Question 2 |
|  | #Transfer the values from the b-tree to a list. This method is similar to transfering the values of a BST to a list from lab #3 |
|  |  |
|  | def BT\_ToList(T,L): |
|  | if T.isLeaf: |
|  | for i in range(len(T.item)):#This is done by using for loops to traverse the elements that are in that depth |
|  | L.append(T.item[i])#appends the elements of the tree to the list |
|  | else: |
|  | for i in range(len(T.item)): |
|  | BT\_ToList(T.child[i], L)#Recursive call to traverse the left side of the b-tree |
|  | L.append(T.item[i]) |
|  | BT\_ToList(T.child[-1],L)#Recursive call to traverse the right side of the b-tree |
|  | return L |
|  |  |
|  |  |
|  | #------------------------------------------------------------------------------ |
|  |  |
|  | #Question 3 |
|  | #Finds the minimum element in the b-tree |
|  | def minElem(T,d): |
|  | h=height(T) |
|  | if d>h: |
|  | return -1 |
|  | if d==0: |
|  | return T.item[0] |
|  | return minElem(T.child[0],d-1)#Recursive call to traverse the left side of the b-tree since we want the minimum element |
|  |  |
|  |  |
|  | #------------------------------------------------------------------------------ |
|  |  |
|  | #Question 4 |
|  | #Finds the maximum element in the b-tree |
|  | def maxElem(T,d): |
|  | h=height(T) |
|  | if d>h: |
|  | return -1 |
|  | if d==0: |
|  | return T.item[-1] |
|  | return maxElem(T.child[-1],d-1)#Recursive call to traverse the right side of the b-tree since we want the maximum element |
|  |  |
|  |  |
|  | #------------------------------------------------------------------------------ |
|  | #Question 5 |
|  | #Finds the amount of nodes in the b-tree |
|  | def numOfNodes(T,d): |
|  | count=0 |
|  | h=height(T) |
|  |  |
|  | if d>h: |
|  | return -1 |
|  | if d == 0: |
|  | return len(T.item) |
|  | else: |
|  | for i in range(len(T.item)): |
|  | count+=numOfNodes(T.child[i],d-1)#Recustive call where the method traverses the tree,but still adds the count value of nodes |
|  | return count+ numOfNodes(T.child[len(T.item)],d-1)#Recursive call where it returns the amount of nodes and traverses the tree, by passing the length of the elements in that depth of the tree |
|  |  |
|  |  |
|  | #------------------------------------------------------------------------------ |
|  | #Question 6 |
|  | #prints all the elements at a certain depth in the b-tree |
|  | def print\_AtDepth(T,d): |
|  | h=height(T) |
|  | if d>h: |
|  | print(None) |
|  | if d == 0: |
|  | for i in range(len(T.item)): |
|  | print(T.item[i],end=' ') #prints all the elements in that certain depth |
|  | else: |
|  | if T.isLeaf is not True: |
|  | for i in range(len(T.item)): |
|  | print\_AtDepth(T.child[i],d-1)#Recursive call that taverses the left side of the b-tree |
|  | print\_AtDepth(T.child[i-1],d-1)#Recursive call that traverses the right side of the b-tree |
|  |  |
|  |  |
|  | #------------------------------------------------------------------------------ |
|  | #Question 7 |
|  | #Counts the amount of full nodes inside the b-tree |
|  | def numOf\_FullNodes(T): |
|  | count=0 |
|  | if len(T.item)==T.max\_items:# The max items that we can have is 5 |
|  | count+=1 |
|  | if T.isLeaf: #We return 0 if we reach a leaf node since we only want NODES |
|  | return 0 |
|  | else: |
|  | for i in range(len(T.item)): |
|  | count+=numOf\_FullNodes(T.child[i]) |
|  | count+numOf\_FullNodes(T.child[len(T.item)]) |
|  | return count |
|  |  |
|  | #------------------------------------------------------------------------------ |
|  | #Question 8 |
|  | #Counts the amount of full leaves in the b-tree |
|  | def numOf\_FullLeaves(T): |
|  | count=0 |
|  | if T.isLeaf: |
|  | if len(T.item)==T.max\_items:# The max items that we can have is 5 |
|  | count+=1 |
|  | else: # this else is if we have a node that is not a leaf |
|  | for i in range(len(T.item)): |
|  | count+=numOf\_FullLeaves(T.child[i]) |
|  | count+numOf\_FullLeaves(T.child[len(T.item)]) |
|  | return count |
|  |  |
|  |  |
|  | ''' |
|  | def depthOfNode(T,k): |
|  | if k in T.item: |
|  | return 0 |
|  | else: |
|  | ''' |
|  |  |
|  |  |
|  |  |
|  |  |
|  | #------------------------------------------------------------------------------ |
|  |  |
|  |  |
|  | L = [30, 50, 10, 20, 60, 70, 100, 40, 90, 80, 110,10,10,10,10,10,10] |
|  | T = BTree() |
|  | for i in L: |
|  | Insert(T,i) |
|  | PrintD(T,'') |
|  | print('\n####################################') |
|  | #T2 is for testing purposes, which is empty |
|  | L2=[] |
|  | T2= BTree() |
|  |  |
|  | for k in L2: |
|  | Insert(T2,k) |
|  |  |
|  | #E is an empty list |
|  | E=[] |
|  |  |
|  | print('The height is ',height(T))#1 |
|  | print('Tree to List', BT\_ToList(T,E))#2 |
|  | print('the minimum element is ',minElem(T,1))#3 |
|  | print('the maximum element is ',maxElem(T,1))#4 |
|  | print('The number of Nodes are:', numOfNodes(T,1))#5 |
|  | print('Elements at depth are: ') |
|  | print\_AtDepth(T,1)#6 |
|  | print() |
|  | print('The number of full Nodes are: ', numOf\_FullNodes(T))#7 |
|  | print('The number of full Leaves are: ', numOf\_FullLeaves(T))#8 |