CS2302 - Data Structures

Spring 2019

Lab No.3 Report

Introduction

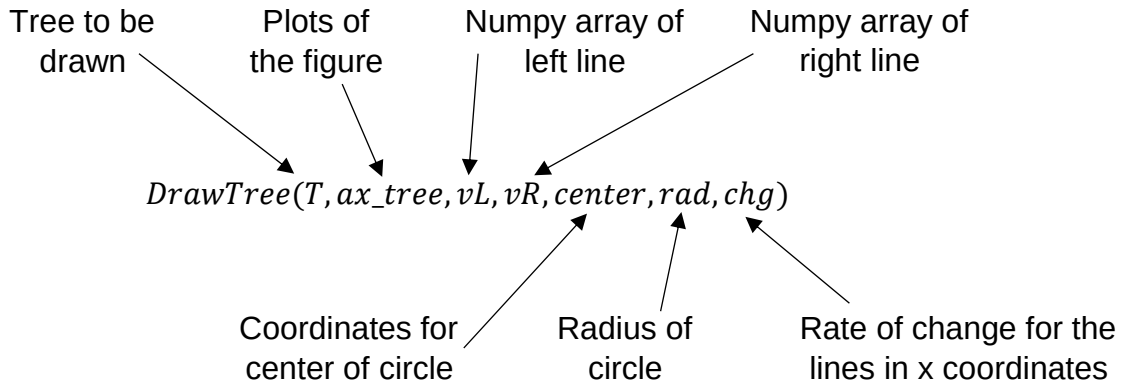
This lab makes use of Binary Search Trees. A type of data structure that stores items in memory, allowing fast lookup, addition and removal of items. In contrast to Binary Trees, these trees have a condition that the items on the left will be smaller than the current one and items in the right will be greater.

In this project we will be performing different types of methods in Binary Search Trees such as displaying the data structure using matplotlib, converting a list into a Binary Tree and vice versa.

Design & Implementation

Problem #1: Displaying Binary Tree

For this problem, the use of the matpotlib is necessary. The method receives various different parameters, which are the following:



First the method's base case will check if the current tree node is None, meaning we have reached a leaf, and returning if it is.

The method will then assign the new centers of our future circle nodes. The center will be the point at which the left and right lines end.

$$newCenterL = vL[1]$$

Then there will be a call to the method that plots the circle in x and y coordinates, this method was previously used in Lab#1. Receiving for parameters the center coordinates in x, y and the desired radius. Returning the x and y coordinates for the circle figure.

$$circle(center, rad)$$

Plotting the previous x and y coordinates for the circle and then filling up this figure in a black color since we don't want the lines for the future nodes to be seen. Finally inserting the item of the node in the center of the circle in a white font.

Then perform a check, if there exist a left node draw a line in the left and do the same for a right node.

$$if\ T.left\ != None$$

After drawing the corresponding previous lines the method will then modify the numpy arrays of these, obtaining the future coordinates for the left and right lines of the next nodes. Here the rate of change parameter will be used, the nodes will always move the same length in the y axis but will continuously reduce in the x coordinate each time to prevent nodes being on top of other.

Finally, the method will do 2 recursive calls, performing all the previous commands in the left and right nodes.

Problem #2: Iterative Search

This method will search if there exists a given item in our Binary Search Tree in an iterative manner.

The method starts with receiving the Binary Search Tree and the item to be searched "k", as parameters. Then performing a check to see if the passed tree is empty, if it is return nothing.

Then we will start the search assigning a temp variable with the root of the given tree. Starting a while loop that exits it if temp is None, meaning the item was not found.

The while loop will proceed to perform 3 checks. If the item in temp is equal to "k" then exit the loop. If the item in temp is bigger than "k" then move to the left subtree, else move to the right subtree.

After exiting the while loop check if temp is None, if it is then the item was not found. Print “The item was not found” and return nothing. Else, if it isn’t None then the item was found print “Item was found” and return the node where the item resides.

Problem #3: Sorted list to BST

The method will receive a list for parameter and return the root of the BST.

First, there will be a check to see if the list is empty, if it is then return nothing.

Since the list is already sorted, transforming it into a BST is simple. We will obtain the middle element of the list and place it as the root node.

Then 2 recursive calls will be performed. Since the left node is smaller than the root the left side of the list will be passed as parameter. Same for the right node, the right side of the list will be passed.

Finally the root is returned, the recursive call will assign the left and right nodes to the current one in this way.

Problem #4: BST to List

Even simpler since we will perform the same as the previous method but backwards.

Having for base case checking if the current node is empty, if it is return an empty list.

Then return a recursive call, calling the left and right nodes with the current node item in the middle.

$$return\ createList(T.left)\ +\ [T.item]\ +\ createList(T.right)$$

In this way, the list will be created automatically.

Problem #5: Print by Depth

For this method a queue is used to perform a breadth first traversal since we want to print them by depth.

First initializing a queue and inserting the root node in it. A while loop will then be used to print the queue.

If the queue is empty then exit the loop and since we want to print by level we will have another loop that checks the depth. For this a counter with the current queue size is used, if the counter is less than 0 then this means another depth has been reached.

Print the first queue item, eliminating the node from the queue in the process and storing it in a temp variable. Then check if this temp node has a left and right node, if there are, add them to the queue.

This way the first loop will print our current level and the second one will print the items in it.

Design

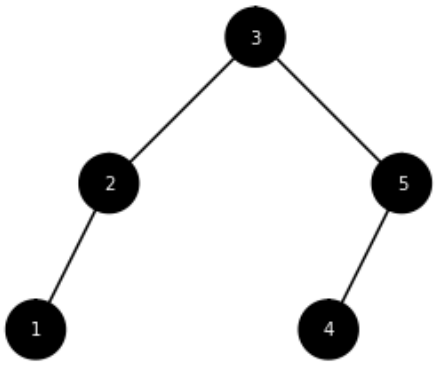
The code provides a randomly generated tree each time it is run. Additionally there is a user interface that asks the user for input when calling the search and list to tree methods. Then the provided trees are output.

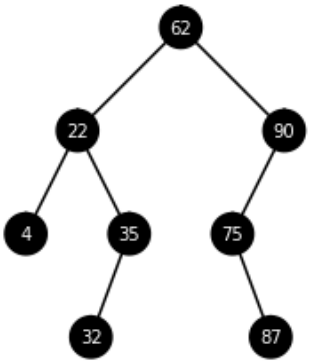
Conclusion

This lab helped in the aspect of understanding how a tree works and how efficient the search for an item is when using this data structure. The assigned problems helped in having a more firm grasp in comprehending how the Tree data structure works and how different types of traversals can be used in it. Furthermore, the use of previous subjects helped understanding them more.

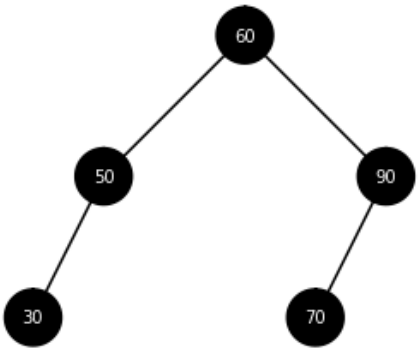
Outputs

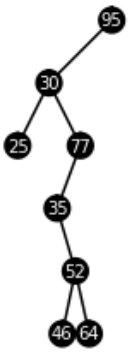
```
----Randomly generated list:
 [62, 22, 35, 90, 75, 87, 32, 4]

----Choose value to find iteratively: 2
Item not found in list.

----Each depth keys:
Keys at depth  0 : 62
Keys at depth  1 : 22 90
Keys at depth  2 : 4 35 75
Keys at depth  3 : 32 87

----Returned sorted list from randomly created tree:
 [4, 22, 32, 35, 62, 75, 87, 90]

----Enter sorted list, separating elements by a space: 1 2 3 4 5
```





```
----Randomly generated list:
 [95, 30, 25, 77, 35, 52, 64, 46]

----Choose value to find iteratively: 25
Item found in list.

----Each depth keys:
Keys at depth  0 : 95
Keys at depth  1 : 30
Keys at depth  2 : 25 77
Keys at depth  3 : 35
Keys at depth  4 : 52
Keys at depth  5 : 46 64

----Returned sorted list from randomly created tree:
 [25, 30, 35, 46, 52, 64, 77, 95]

----Enter sorted list, separating elements by a space: 30 50 60 70 90
```





Appendix

```
"""
Draw BST ----------------------------------------------------------------------
The method will first draw a circle if the current node isn't null, then put
the item of the node in the middle of the circle and check if the node has
a left and right child. A left line will be ploted if there is a left child
and the same repeats for a right child.
"""
def circle(center,rad):
    n = int(4*rad*math.pi)
    t = np.linspace(0,6.3,n)
    x = center[0]+rad*np.sin(t)
    y = center[1]+rad*np.cos(t)
    return x,y

def DrawTree(T,ax_tree,vL,vR,center,rad,chg):
    if T == None:
        return

    newCenterL = vL[1] #New center for left child
    newCenterR = vR[1] #New center for right child

    x,y = circle(center,rad)
    ax_tree.plot(x,y,'k') #plot a circle in our current center
    ax_tree.fill(x,y,'k') #fill the circle in a black color



    #Output the current node item at the center of the circle in a white font
    ax_tree.text(center[0],center[1],T.item,color='w',
                 horizontalalignment='center',verticalalignment='center',
```

```
            fontsize=10)

    if T.left != None: #check for left child and draw a line if it exists
        ax_tree.plot(vL[:,0],vL[:,1],color='k')
    if T.right != None: #check for right child and draw a line if it exists
        ax_tree.plot(vR[:,0],vR[:,1],color='k')

    #New vertices arrrays for left and right lines
    leftVL = np.copy(vL)*0
    leftVL[0] = vL[1]
    leftVL[1][0] = vL[1][0]-chg[1] #rate of change for x
    leftVL[1][1] = vL[1][1]-chg[0] #rate of change for y

    leftVR = np.copy(vL)*0
    leftVR[0] = vL[1]
    leftVR[1][0] = vL[1][0]+chg[1]
    leftVR[1][1] = vL[1][1]-chg[0]

    rightVL = np.copy(vR)*0
    rightVL[0] = vR[1]
    rightVL[1][0] = vR[1][0]-chg[1]
    rightVL[1][1] = vR[1][1]-chg[0]

    rightVR = np.copy(vR)*0
    rightVR[0] = vR[1]
    rightVR[1][0] = vR[1][0]+chg[1]
    rightVR[1][1] = vR[1][1]-chg[0]

    DrawTree(T.left,ax_tree,leftVL,leftVR,newCenterL,rad,[chg[0],chg[1]*.75])
    DrawTree(T.right,ax_tree,rightVL,rightVR,newCenterR,rad,[chg[0],chg[1]*.75])

"""
```

```
Iterative Search ----------------------------------------------------------
If the item is bigger than the root then check the left child else check the
right one. If at the end of the while loop the current node is null then the
item isn't in the list.
"""
def Search(T,k):
    if T is None:
        return None
    temp = T
    while temp != None:
        if temp.item == k:
            break
        if temp.item > k:
            temp = temp.left
        else:
            temp = temp.right
    if temp == None:
        print("Item not found in list.")
        return
    print("Item found in list.")
    return temp


"""
Sorted List to BST ----------------------------------------------------------
Divide the list by the middle, the middle point becomes the root of the tree.
Continue dividing each newly partitioned list in the same way.
"""
def listToBST(L):
    if not L:
        return None
    mid = int((len(L))/2)
    root = BST(L[mid])
```

```
    root.left = listToBST(L[:mid])
    root.right = listToBST(L[mid+1:])
    return root

"""
BST to List -------------------------------------------------------------
Obtain the root node and add it to the list, repeat the same for the left and
right nodes
"""
def createList(T):
    if T == None:
        return []
    return createList(T.left) + [T.item] + createList(T.right)

"""
Print by Depth ----------------------------------------------------------
Used a queue to perform a Breadth-first traversal, while the queue isn't empty
continue obtaining the different nodes for the Tree.
"""
def PrintD(T):
    q = queue.Queue()
    q.put(T)
    depth = 0
    while not q.empty():
        counter = q.qsize()
        print("\nKeys at depth ",depth,": ",end='')
        while counter > 0: #when the counter reaches 0 means a new depth
            n = q.get()
            print(n.item,end=' ')
            if n.left:
                q.put(n.left) #if there is a left node add it to the queue
            if n.right:
```

```
                q.put(n.right) #if there is a right node at it to the queue
            counter -= 1
        depth += 1
    print("\n")

"""
Test ----------------------------------------------------------------------
"""
center = [0,0]
fig_tree, ax_tree = plt.subplots()
fig_tree2, ax_tree2 = plt.subplots()
vL = np.array([[0,0],[-100,-100]])
vR = np.array([[0,0],[100,-100]])

#Generate a Tree from a randomly generated List
T = None
A = random.sample(range(1, 101), 8)
for i in A:
    T = Insert(T,i)
print("----Randomly generated list: \n",A)

#Iterative search apllied
s = int(input("----Choose value to find iteratively: "))
Search(T,s)

print("\n----Each depth keys: ",end='')
PrintD(T)

#Convert Binary Search Tree to List
print("----Returned sorted list from randomly created tree: \n", createList(T))

#Convert list to Binary Search Tree
```

```
input_string = input("----Enter sorted list, separating elements by a space: ")
L = input_string.split(" ")
L = list(map(int,L))
T2 = listToBST(L)

#Tree drawn from previous list, parameters are: Binary Tree,
#axes for tree number 1, vertices for the left and right line, coordinates for
#center point, radius and the rate of change for the x and y axis for each
#new center
DrawTree(T2,ax_tree,vL,vR,center,20,[100,100/2])

#Tree drawn from randomly generated list
DrawTree(T,ax_tree2,vL,vR,center,20,[100,100/2])

ax_tree.set_aspect(1.0)
ax_tree2.set_aspect(1.0)
ax_tree.axis('off')
ax_tree2.axis('off')
plt.show()
```

I certify that this essay is original work prepared by me, Jesus A Hernandez.