**Course:** CS 2302

**Author:** Kimberly Morales

**Assignment:** Lab 3

**Instructor:** Olac Fuentes

**TA(s):** Anindita Nath, Maliheh Zargaran

**Date:**  3/11/2019

# **Introduction**

The program's purpose is to implement five more additional features to the binary search tree(bst): display, iterative search, sorted list to bst, bst to sorted list, and print at the depth. To display, we utilize the matplotlib library to create a figure that will annotate the value of the node with a circle drawn around it and connect to the next node if it exists. An iterative search is the loop version of recursive search that will look for an element to the left or right if it is less than or greater than the element respectively. Sorted list to bst will first sort the list and then create a tree by recursively calling from the middle element and adding this middle element to the tree. This will be done without the insert operation. BST to sorted list will be similar to inorder but instead, it will append the tree's current node to the list. Printing at depth will recursively call and subtract from d and go to the left and right and will print the element.

# **Proposed Solution Design and Implementation**

To begin my solution and design, I first tried to understand how to traverse through a bst through inorder, preorder, and postorder. Then I looked for methods that I could reuse for several parts of the main lab code so that it can cohesively run and not program similar methods.

**Utility**

sortList

Initially, I used the default Python sort method which utilizes timsort which I am not too familiar with but only used it for the initial stages of building the main lab methods. After building all my lab methods, I got rid of the sort method and made my own with a modified bubble sort algorithm that only takes in a list. This version of bubble sort has a Boolean flag that will mark it true if an item was switched. If at the end of the inner loop it ends up being false, then that means that the list has already been sorted since all elements are ascending order so it stops executing the loop. This makes it an optimized big o of n(linear) since it will not needlessly continue the loop. Nothing is returned since it is just modifying the items and references. This method is utilized for the building of a balanced bst.

list\_to\_bst

This method is not used for any of the lab methods but as a comparison to building a balanced bst with a sorted list. This method just adds a node to the left or right subtree with the already implemented insert operation. This method would often just make a linked list like structure.

height

Returns the max height of the tree from the traversals of the right and left subtrees. It returns the max height from the left and right plus 1. If the tree is only empty, then return -1. This method is an important part of the display method since it will draw the line height.

**Part 1. Display bst as a figure**

circle

Method used from the first lab and draws a circle based on an x and y coordinate. It used to draw a circle over a node value in the draw trees method.

draw\_trees

This method draws the bst which includes lines, circles, and node values. My general approach to this method was to see how I would draw it initially and then formulate a recursive algorithm. The first base case is that if the tree is empty then don’t draw anything which means an empty canvas.

Then I create xm and ym which are two coordinates in charge of connecting two nodes centers to each other. They utilize the smallest canvas x size which is zero so that it follows a coordinate system and the largest canvas x size which is the line length so that it does not zoom out too much from the tree. The ym coordinate is based on the minimum y which is the line length so that the canvas is not drawn too big and this value is subtracted from the height of the tree. Having the height makes it consistent to draw the tree since only half the value of y will make it smaller in the line length.

With these coordinates, they are used as the center point for the circle and create the xc and yc coordinates. These coordinates are in charge of adding the node value and drawing the circles. These circles are then filled in with white color so that it hides the line that hits the circles center point.

Next, the tree recursively calls to the left and will plot elements in the negative side of the coordinate system which means using the xmin value with xm so that the elements shift consistently and half the value so that is moving slightly within the root node. The y coordinate is kept consistently since the line length should not be smaller each time a new line is plotted. This is similarly done with the right subtree and will plot elements more near the positive side of a coordinate system which means using the xmax value with xm. The y coordinate is still consistent for similar reasons.

Once the recursive functions are done then the node value is annotated from the previous xm and ymin values with the attributes to center the value. The circle is drawn around this value.

deploy\_trees

This is the program that handles the inputs for the draw\_trees method since having it in the main method would make the menu look cluttered and to debug it effectively since it only deals with the draw\_trees program directly. This method also makes sure that if the height is 0 then draw\_trees will only draw node and to avoid a zero error. There are increments to slight shift the tree and these only exist to make it look more consistent.

**Part 2. Iterative Search**

search

I already had code written down from a previous exercise but still rewrote some parts since I forgot to return None if nothing is found. I create a temp t and within a while loop that will loop if T is not none. If k is equal to the current item in T then return the value of T.item. if k greater than the current node then go to the right. If k is less than the current node then goes to the left. If the loop ends, then that means that k is not in the tree so return None.

**Part 3. Build bst from sorted list**

slist\_to\_bst

Unlike list\_to\_bst, this method already has a sorted list to build a balanced bst. To build a balanced tree means that the left and right subtree heights differ by 1 and both the right and left subtrees are balanced. If the tree is empty, then do nothing and return None. Else, then get the median value of the sorted list by dividing the length of the list by 2 and make that the root node since it can potentially have half elements in the left and right subtrees. Then recursively add elements to the left subtree by adding elements from the beginning of the list to the median. Same for the right subtree, except add elements from the element after the median until the end of the list. Then return the new tree. This should be big o of n since it does not rely on the insert operation and only goes through the list linearly.

**Part 4. Build bst from sorted list**

bst\_to\_slist

This is the opposite of building a bst from a sorted list and instead extracts elements from the tree to make a sorted list. My initial thought was heap sort but seemed too intricate of a solution, so I thought of just reversing the algorithm from the previous method. Another algorithm that I looked at was inorder traversal since it makes a bst sorted but instead of printing the current node, it is appended into the list. First, I make sure that the list is not empty and if it is then it cannot enter the base case and returns an empty list. Else, then the tree goes all the way to the left since it will be the smallest element in the list which is then appended to the list. Then it recursively calls to the right, like inorder traversal. Once the tree has finished the traversal, just return the new sorted list. This should be big o(n) since traversal through a list means visiting a node at least once in the tree.

**Part 5. Print by depth**

print\_at\_depth

Prints item based on the depth. If the tree is empty, then return a string that the list is empty. If the depth is 0 which means only one node, then just print the value of the root. Anything else then recursively calls to the left and subtract one from the depth. Do the same for the right subtree. These recursive calls are set up this way because printing by depth means the distance away from the node which means we still must visit both tree trees but from a certain distance which is d.

**Main/User I/0**

I created an overall menu with two submenus: building from either a list or bst, then choosing one of the main lab methods. Either way, the user is always asked for the size of the list. This size is used to generate a random list from a sample. I created a random list each time for testing purposes since I wanted to make sure it will always build a balanced tree. No individual method is made for generating a random list since I am working with the native python list and making a separate method for small lines of code would have been not efficient. I still used the list\_to\_bst to show how to draw the example figure from the lab instructions and only inserts items to list without sorting the list. If slist\_to\_bst is chosen, then the list is sorted and printed which then leads to the second menu. The second menu contains the search, bst to sorted list, and print by depth methods. Choosing search will prompt for the element to be found. bst to sorted list will show how the list is built and returns the sorted list. Print by depth will prompt the depth number and prints the items based on the depth. All three methods lead to the figure being built. This figure is also saved within the program folder as a png. Since the program runs on a while loop, it will ask to continue. If the user picks y/yes or odd input, then it restarts the program and asks for a new list. if n/no then it stops the program.

**Experimental Results**

I picked the basic base cases of an empty list/tree, a tree with one node, and tree sizes over 1. I tested erroneous input from negative numbers and letters. The input column shows how to navigate through the program. It may not accurately display all of the programs since I truncated some repeated parts so to shorten the columns. The time is shown in the middle column since it records all operations of the program. The output column shows results of the functions, the original list, sorted list, and the figure is drawn.

|  |  |  |
| --- | --- | --- |
| Input | Time | Output |
| Binary Search Tree Program  Enter the size of the list  0  1. List to BST  2. Sorted List to BST  choice = 2  1. Search  2. BST to Sorted List  3. Print at Depth  choice = 1  Input Element = 0  Continue?(y/n)  y  Enter the size of the list  0  1. List to BST  2. Sorted List to BST  choice = 2  1. Search  2. BST to Sorted List  3. Print at Depth  choice = 2  Continue?(y/n)  y  Binary Search Tree Program  Enter the size of the list  0  1. List to BST  2. Sorted List to BST  choice = 2  1. Search  2. BST to Sorted List  3. Print at Depth  choice = 3  Input Depth = 0  Continue?(y/n)  n | 27.91981873508801 | Original List: []  Sorted L: []  Search: None  BST to List: []  Key at depth 0: NONE  Figure: EMPTY |
| Binary Search Tree Program  Enter the size of the list  1  1. List to BST  2. Sorted List to BST  2  1. Search  2. BST to Sorted List  3. Print at Depth  1  Input Element: 84  Continue?(y/n)  y  Enter the size of the list  1  1. List to BST  2. Sorted List to BST  choice = 2  1. Search  2. BST to Sorted List  3. Print at Depth  choice = 2  Continue?(y/n)  y  Binary Search Tree Program  Enter the size of the list  1  1. List to BST  2. Sorted List to BST  choice = 2  1. Search  2. BST to Sorted List  3. Print at Depth  3  Input Depth: -1  Continue?(y/n)  n | 41.70803822247405 | Original List: [84]  Sorted L: [84]  Search: 84  Original List: [68]  Sorted L: [68]  [68]  BST to List[68]  Original List: [57]  Sorted L: [57]  Key at depth -1: |
| Binary Search Tree Program  Enter the size of the list  -5  Binary Search Tree Program  Enter the size of the list  a  Binary Search Tree Program  Enter the size of the list  0  Original List: []  1. List to BST  2. Sorted List to BST  8 | 14.596015278066963 | ERROR: INVALID INPUT  ERROR: INVALID INPUT  ERROR: INVALID INPUT |
| Binary Search Tree Program  Enter the size of the list  10  1. List to BST  2. Sorted List to BST  choice = 2  1. Search  2. BST to Sorted List  3. Print at Depth  choice = 1  Input Element: 76  Continue?(y/n)  y  Binary Search Tree Program  Enter the size of the list  10  1. List to BST  2. Sorted List to BST  choice = 2  1. Search  2. BST to Sorted List  3. Print at Depth  choice = 2  Continue?(y/n)  y  Binary Search Tree Program  Enter the size of the list  10  1. List to BST  2. Sorted List to BST  choice = 2  1. Search  2. BST to Sorted List  3. Print at Depth  3  Input Depth : 1  Continue?(y/n)  n | 41.69136794336602 | Original List: [63, 42, 76, 89, 57, 21, 82, 87, 22, 99]  Sorted L: [21, 22, 42, 57, 63, 76, 82, 87, 89, 99]  Search: 76  Original List: [50, 27, 24, 3, 20, 49, 23, 43, 6, 85]  Sorted L: [3, 6, 20, 23, 24, 27, 43, 49, 50, 85]  [3]  [3, 6]  [3, 6, 20]  [3, 6, 20, 23]  [3, 6, 20, 23, 24]  [3, 6, 20, 23, 24, 27]  [3, 6, 20, 23, 24, 27, 43]  [3, 6, 20, 23, 24, 27, 43, 49]  [3, 6, 20, 23, 24, 27, 43, 49, 50]  [3, 6, 20, 23, 24, 27, 43, 49, 50, 85]  BST to List[3, 6, 20, 23, 24, 27, 43, 49, 50, 85]  Original List: [78, 14, 92, 31, 17, 35, 90, 27, 36, 53]  Sorted L: [14, 17, 27, 31, 35, 36, 53, 78, 90, 92]  Key at depth 1:  27  90  Figures: |
| Binary Search Tree Program  Enter the size of the list  5  1. List to BST  2. Sorted List to BST  choice = 2  1. Search  2. BST to Sorted List  3. Print at Depth  choice = 1  Input Element: -5  Continue?(y/n)  y  Binary Search Tree Program  Enter the size of the list  5  1. List to BST  2. Sorted List to BST  choice = 2  1. Search  2. BST to Sorted List  3. Print at Depth  choice = 2  Continue?(y/n)  y  Binary Search Tree Program  Enter the size of the list  5  1. List to BST  2. Sorted List to BST  choice = 2  1. Search  2. BST to Sorted List  3. Print at Depth  3  Input Depth: -8  Continue?(y/n)  n | 32.91414663326154 | Original List: [21, 90, 15, 68, 36]  Sorted L: [15, 21, 36, 68, 90]  Search: None  Original List: [45, 71, 50, 15, 19]  Sorted L: [15, 19, 45, 50, 71]  [15]  [15, 19]  [15, 19, 45]  [15, 19, 45, 50]  [15, 19, 45, 50, 71]  BST to List[15, 19, 45, 50, 71]  Original List: [5, 32, 97, 43, 27]  Sorted L: [5, 27, 32, 43, 97]  Key at depth -8:  Figures: |

# **Conclusions**

I learned new methods and ways to traverse a tree from already previous knowledge of tree traversals and balanced bsts. I also learned that a bst algorithm does not always have to be implemented recursively as seen with the search operation and building the list from the bst. Building a balance bst can also be done with big o of n without the insert operation just be sorting the list beforehand. When thinking about the processes of the tree, it also reminded me of a stack popping and this expanded my knowledge of recursion even more. I admit to having difficulty with drawing the bst tree since it required me reusing/recode parts of my lab 1 code and seeing how the height is consistent when drawing the figure. When adding text, the annotate feature of matplotlib is useful for marking data at points and could even be used to be centered at the coordinate. Matplotlib also has a fill function and ordering of graph elements which made it easier to sneakily hide lines that hit the items in the circles. Overall, my previous difficulties with bsts were expanded greatly.

# **Appendix**

"""

Course: CS 2302 [MW 1:30-2:50]

Author: Kimberly Morales

Assignment: Lab 3

Instructor: Olac Fuentes

TA(s): Anindita Nath , Maliheh Zargaran

Date: 3/8/2019

Date of last modification: 3/8/2019

Purpose of program:

The programs purpose is to implement five more additonal features to the binary search tree(bst): display, iterative search, sorted list to bst, bst to sorted list, and print at the depth.

To display, we utilize the matplotlib library to create a figure that will annotate the value of the node with a circle draw arround it and connect to the next node if it exists.

Iterative search is the loop version of recurisve search that will look for an element to the left or right if it is less than or greater than the element respectively.

Sorted list to bst will first sort the list and then create a tree by recurisvely calling from the middle element and adding this middle element to the tree. This will be done without the insert operation.

BST to sortedlist will be similar to inorder but instead it will append the tree's current node to the list

Printing at depth will recursively call and subtract from d and go to the left and right and will print the element.

"""

import numpy as np

import matplotlib.pyplot as plt

import math

import random

#################################################################################################

#BINARY SEARCH TREE CLASS

#################################################################################################

class BST(object):

# Constructor

def \_\_init\_\_(self, item, left=None, right=None):

self.item = item

self.left = left

self.right = right

def Insert(T,newItem):

if T == None:

T = BST(newItem)

elif T.item > newItem:

T.left = Insert(T.left,newItem)

else:

T.right = Insert(T.right,newItem)

return T

#################################################################################################

#UTILITY METHODS

#################################################################################################

#Sorts list with modfiied bubble sort algorithm that has a boolean flag to indicate if a swap was done

def sortList(L):

for i in range(len(L)):

swapped = False

for j in range(len(L)-1):

if L[j] > L[j+1]:

L[j], L[j+1] = L[j+1], L[j]

swapped = True

if swapped == False:

break

#Traditionally inserts node to the list

def list\_to\_bst(L):

T = None

for i in range(len(L)):

T = Insert(T,L[i])

return T

#Counts the longest height from the node to the deepest node

#If there is no tree then it is -1

#If there is one node then it is h =0

def height(T):

if T is None:

return -1

return max(height(T.left) , height(T.right)) + 1

#################################################################################################

#LAB METHODS

#################################################################################################

#1) Display the binary search tree as a figure

########################################DISPLAY###################################################

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

#Draws tree based on the canvas size and height of the tree

#xmin: min size which is 0 so to not work with negatives

#xmax: max size which is the line length so to not to make the canvas too big or smallest

#ymin is same size as xmax but goes slightly to the right so to move the tree nodes

#h is ymin divided by the tree height so to adjust the canvas to how big the tree is

def draw\_trees(ax,T,xmin, xmax,ymin,h):

#If the tree is empty then return the tree and draw nothing

if T is None:

return T

#xm: x coordinate that draws the line from the node center to next node center, add max and min element to avoid overdrawing and divides by 2 to minimize distance

xm = (xmin + xmax) / 2

#ym: y coordinate that draws the line from the node center to next node center, subtracts height to avoid overdrawing

ym = (ymin- h)

#xc and yc are coordinates for the center of the circle to be draw around number

xc,yc = circle([xm,ymin],3)

ax.fill(xc,yc, zorder=3,color='white') #Fills circle to hide line that hits the number, uses zorder to put it as a front layer

#Plot left elements on the left side of the axis

if T.left is not None:

ax.plot([xm,((xm+xmin)/2)],[ymin,ym],color='k',zorder=1) #Uses min since it is messing with negatives

draw\_trees(ax,T.left,xmin,xm,ym,h)

#Plots right elements on the right side of the axis

if T.right is not None:

ax.plot([xm,(xm+xmax)/2],[ymin,ym],color='k') #Uses max since it is messing with positives

draw\_trees(ax,T.right,xm,xmax,ym,h)

#Matplotlib function (annotate) that adds text to a coordinate and centers it

ax.annotate(str(T.item), xy=(xm,ymin), xycoords = 'data',ha='center',va='center')

ax.plot(xc ,yc,color='k') #Plots with circle element to keep it center

#Used for top down design and to seperate the main function of drawing the tree

def deploy\_trees(T):

#line\_length: Same line length from the origin to the end points

line\_length = 100.0

#To avoid zero error if there is only a single node in the tree

if height(T) == 0:

h = 0

else:

#This will change line length approripriately with the height of the tree

h = (line\_length-10.0)/(height(T))

fig, ax = plt.subplots()

draw\_trees(ax,T,0,line\_length,line\_length-5,h)

ax.set\_aspect('1')

ax.axis('off')

plt.show()

fig.savefig('bst' + str(height(T)) + '.png')

##############################################################################################

#2) Iterative version of the search operation

def search(T,k):

t = T #Temp tree

while t is not None:

if t.item == k: #If k is the same value of the current node then k is in the node

return t.item

elif t.item < k: #If k is greater than the node value then go to the right

t = t.right

elif t.item > k: #If k is less than the node value then go to the left

t = t.left

return None

##############################################################################################

#3) Building a balanced binary search tree given a sorted list as input. Note: this should not use the insert operation,

#the tree must be built directly from the list in O(n) time

def slist\_to\_bst(L):

if not L:

return None

#If there is only 1 element then the bst is already balanced

if len(L) == 1:

T = BST(L[0])

return T

mid = (len(L)) // 2 #mid: Middle element/median of sorted list

T = BST(L[mid]) #Make mid the root

#From start until middle of the list, add the middle element for the left subtree

T.left = slist\_to\_bst(L[:mid])

#From mid plus until end of the list, add the middle element for the righ subtree

T.right = slist\_to\_bst(L[(mid+1):])

return T

##############################################################################################

#4) Extracting the elements in a binary search tree into a sorted list. As above, this should be done in O(n) time.

#Uses modified version of inorder to append the current element instead of printing it

def bst\_to\_slist(T,s\_list):

if T is not None:

bst\_to\_slist(T.left,s\_list) #Go all the way to the left to get smallest element in the list

s\_list.append(T.item) #Add current node to the list

print(s\_list)

bst\_to\_slist(T.right,s\_list)

return s\_list

##############################################################################################

#5) Printing the elements in a binary tree ordered by depth. The root has depth 0, the root’s children have depth one,

#and so on. For example, for the tree in the figure, your program should output:

def print\_at\_depth(T,d):

if T is None:

return "List is Empty"

if d == 0:

print (T.item)

else:

print\_at\_depth(T.left,d-1)

print\_at\_depth(T.right,d-1)

#################################################################################################

#MAIN

#################################################################################################

#A menu is created to allow for multiple options along with error exceptions

cont = True #cont: Will continue the loop as long it is inputted true by the user

while cont:

try:

#Create randomed generated list from size n

print("Binary Search Tree Program")

n = int(input("Enter the size of the list \n"))

L = random.sample(range(100), n)

#L = [10,4,15,2,8,12,18,1,3,5,9,7]

print("Original List: " + str(L))

#First Menu: Can traditionally create BST with unsorted list or use sorted list

print("\n1. List to BST" + "\n2. Sorted List to BST ")

choice = int(input())

if choice == 1:

T = list\_to\_bst(L)

elif choice == 2:

sortList(L)

print("Sorted L: " + str(L))

T = slist\_to\_bst(L)

else:

print("ERROR: INVALID INPUT")

break

#Second Menu: Can use search, create sorted list from BST from the first menu or print at depth

print("\n1. Search" + "\n2. BST to Sorted List" + "\n3. Print at Depth")

choice = int(input())

if choice == 1:

print("Input Element \n")

k = int(input())

print("Search: " + str(search(T,k)))

elif choice == 2:

print("BST to List" + str(bst\_to\_slist(T,[])))

elif choice == 3:

print("Input Depth \n")

d = int(input())

print("Key at depth " + str(d) + ": ")

print\_at\_depth(T,d)

else:

print("ERROR: INVALID INPUT")

break

#Displays figure to matplotlib window and saves it as a figure

deploy\_trees(T)

#Ask to continue loop and keep usng the program

print("Continue?(y/n) ")

cont = input()

if cont.lower() == 'y' or cont.lower() == 'yes':

cont = True

elif cont.lower() == 'n' or cont.lower() == 'no':

cont = False

else:

print("ERROR: INVALID INPUT")

cont = True

except ValueError: #Catches invalid inputs such as negative numbers, letters, and special characters

print("ERROR: INVALID INPUT")

# **Academic Honesty**

“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.”

Name: Kimberly Morales

Signature: