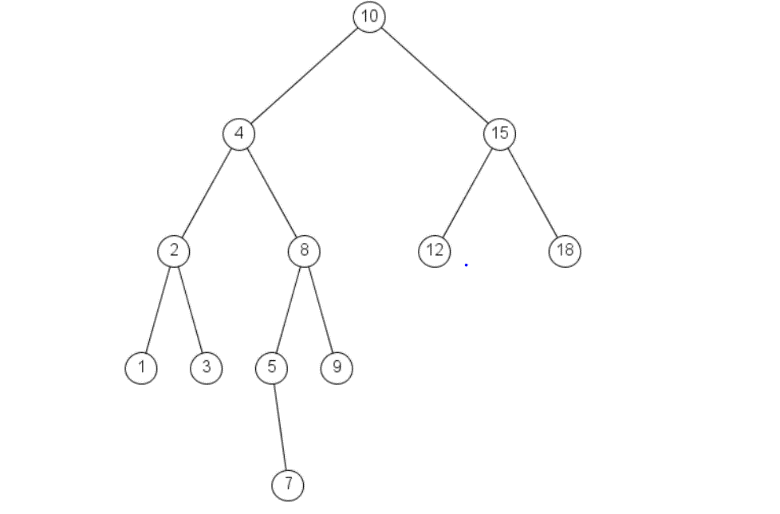
Rigoberto Quiroz

3/11/19

Lab3 Report

CS2302 1:30 PM – 2:50 PM

Description:

For this Lab I had to manipulate a Binary Search Tree (BST) to preform various functions such as displaying a BST as shown in the image below, preform a iterative search to find any key in the BST and if not found returning None. Build a new BST from a sorted list that would take O(N) time to complete. Extract elements from a BST into a list in ascending order, and given any depth of a tree, print all elements that reside there, and if given a depth that exceeds or tree’s depth print None.  


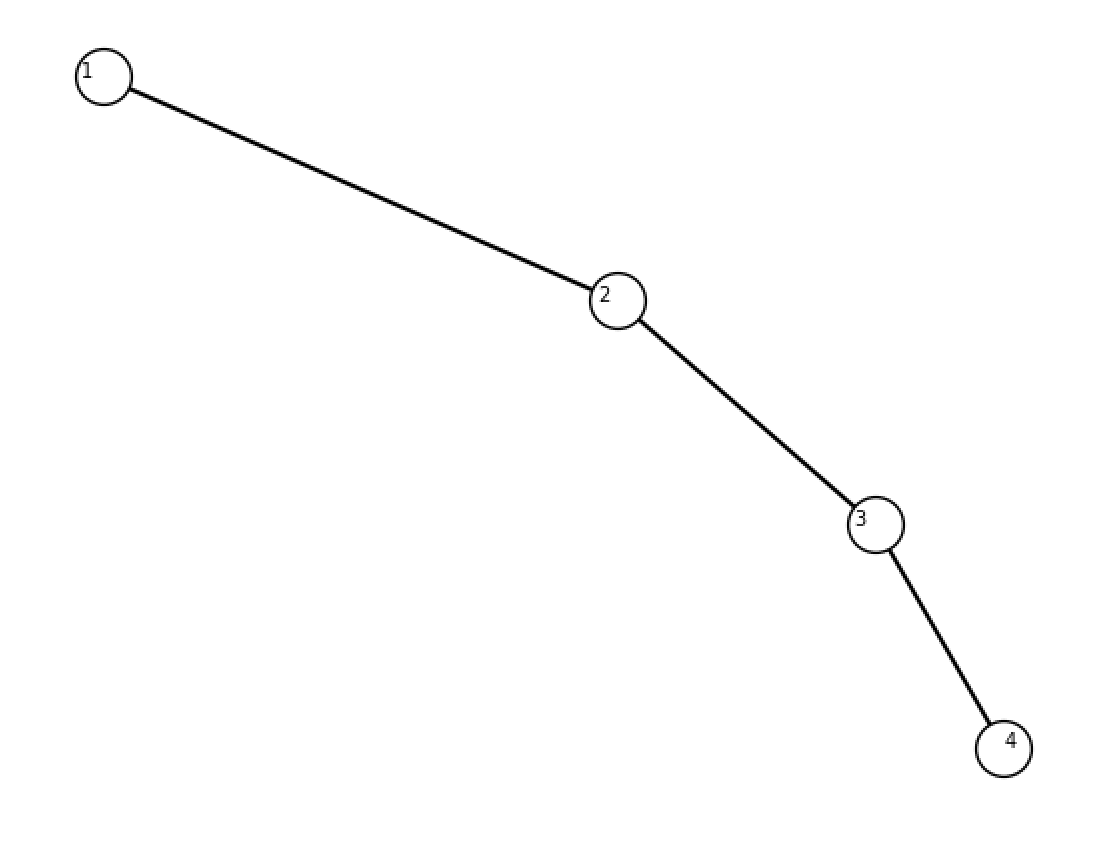
The way I was able to solve the problems in this lab was by (1) reusing some code from Lab 1 to draw a BST and reading some documentation on the library matplotlib to be able to draw circles and insert text inside each circle. (2) Using loops to find a given key, moving to the left or right depending where the might be located. (3) Given a sorted list (ascending order), we will take the midpoint of the list and make that our root, then split the remaining of the list and do the same procedure until we have an empty list. (4) We will move to the left most node and then middle node and then rightmost node to extract each item of the nodes in a sorted list (ascending order). Given a depth, we will print the items at that certain depth, we will move to the right and left branches until we reach our desired depth using recursion and print of nodes.

**3.1 Displaying BST:**

First, we will have to construct our BST using the insert method and our class BST. Then we will call another method, printBST, which will take as parameters the BST, ax(plotting points), coordinates, and changes to each coordinate. In this recursive method we will have 3 cases that we will have to check, if the BST is empty, then we will return None. If our BST node is a leaf, then we will input our text and draw the circle that contains in the info and then return. If our BST node only has a left or right child, then we will branch to the left or right, depending which branch is available to us. If none of these cases are entered then we will draw both the left and right branches and circle then, insert our root repeat this process until they are no more elements in the BST.

BST: 1,2,3,4

Output:

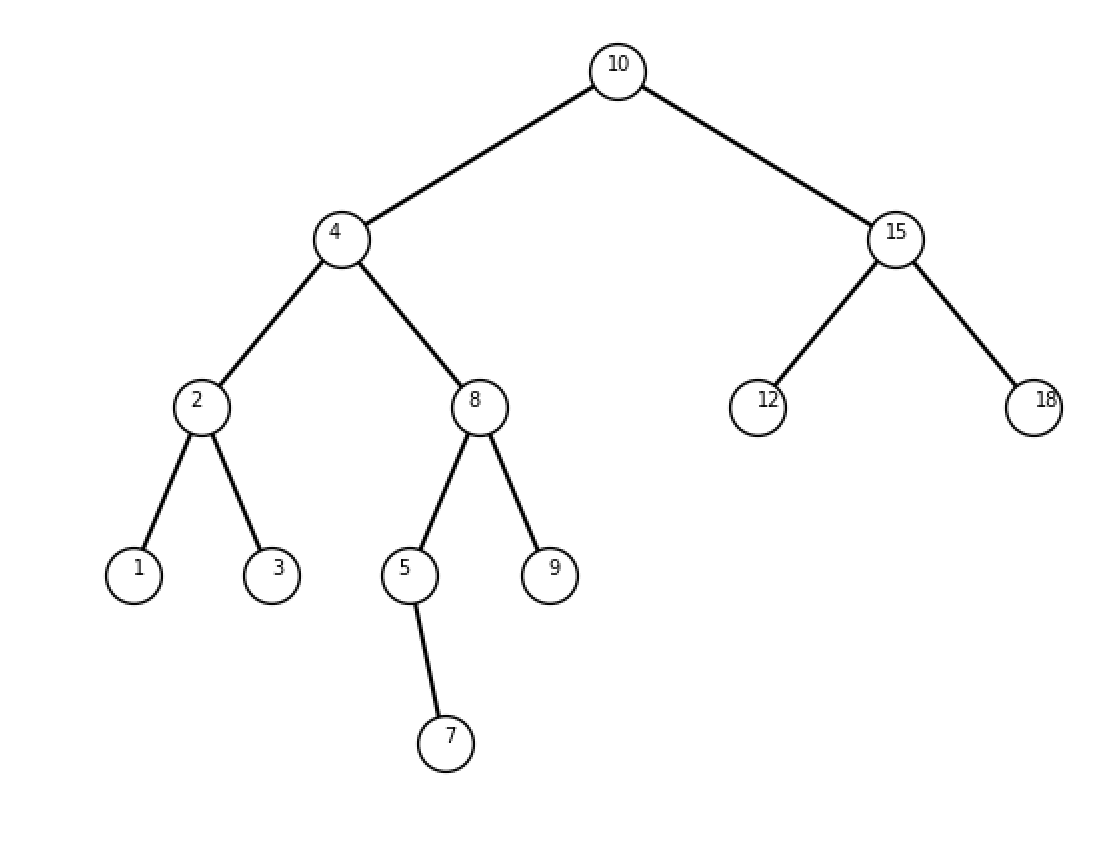


BST: Empty

Output:

BST: 10,4,15,2,8,12,18,1,3,5,9,7

Output:



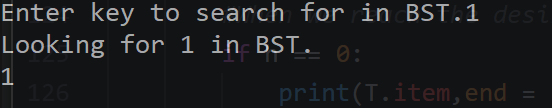
3.2 Iterative Search:

We will send out BST, and key we want to find to our method, IterativeSearch, we will then save our root in a temp value in have a while loop move through our Tree. If our key is greater than our root then we will move to the right because the right branches store bigger values than the root. Otherwise we will move left because the left stores values that are less than our root. We will stop until our key has been found or we no longer have elements in our tree. If that is the case then we will return None.

BST: 10,4,15,2,8,12,18,1,3,5,9,7

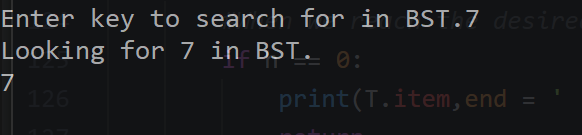
Key: 1

Output:



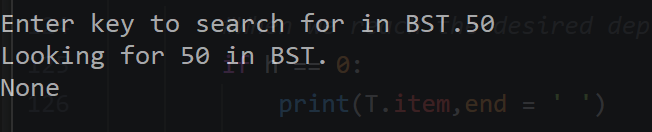
Key: 7

Output:



Key: 50

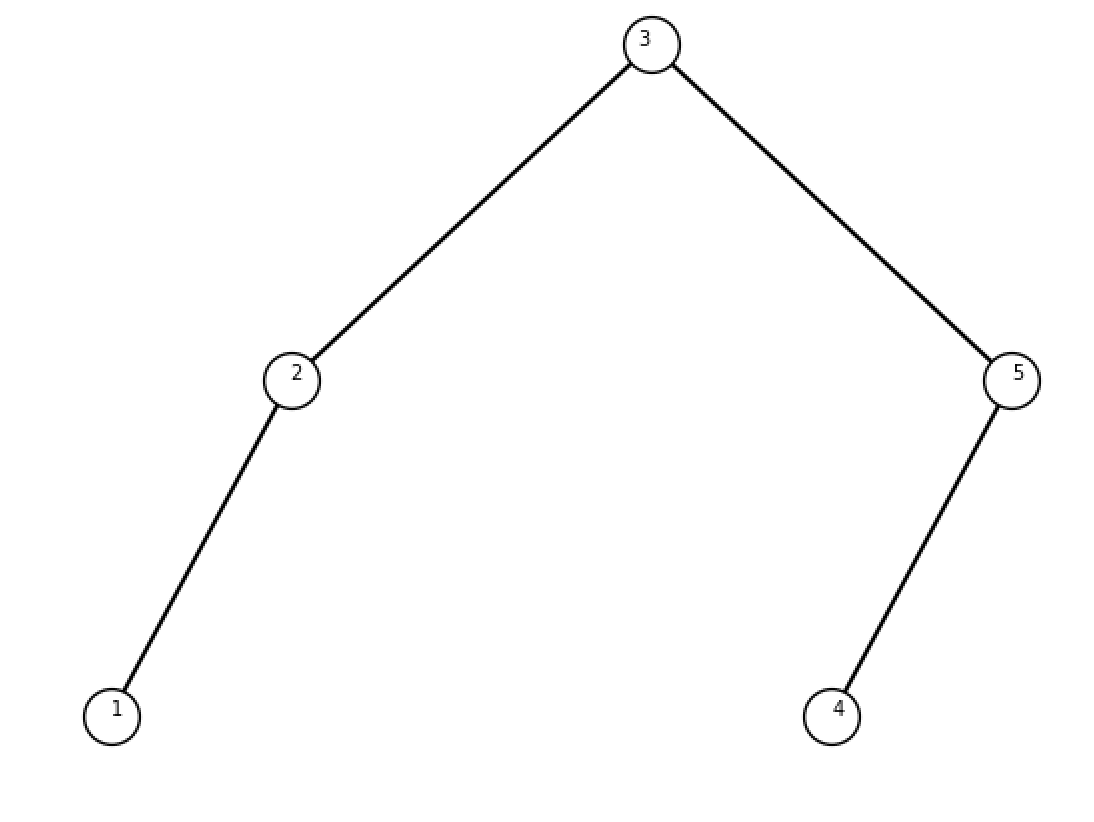
Output:



**3.2 Building B-Tree from sorted list:**

We will send a sorted list (ascending order) to our method, buildTree, and we will take the midpoint of our list and insert it into out tree, the we will create to separate list, one of elements that are before the midpoint, and the other with elements that are after the midpoint. We will repeat the same process until we do not have more elements.

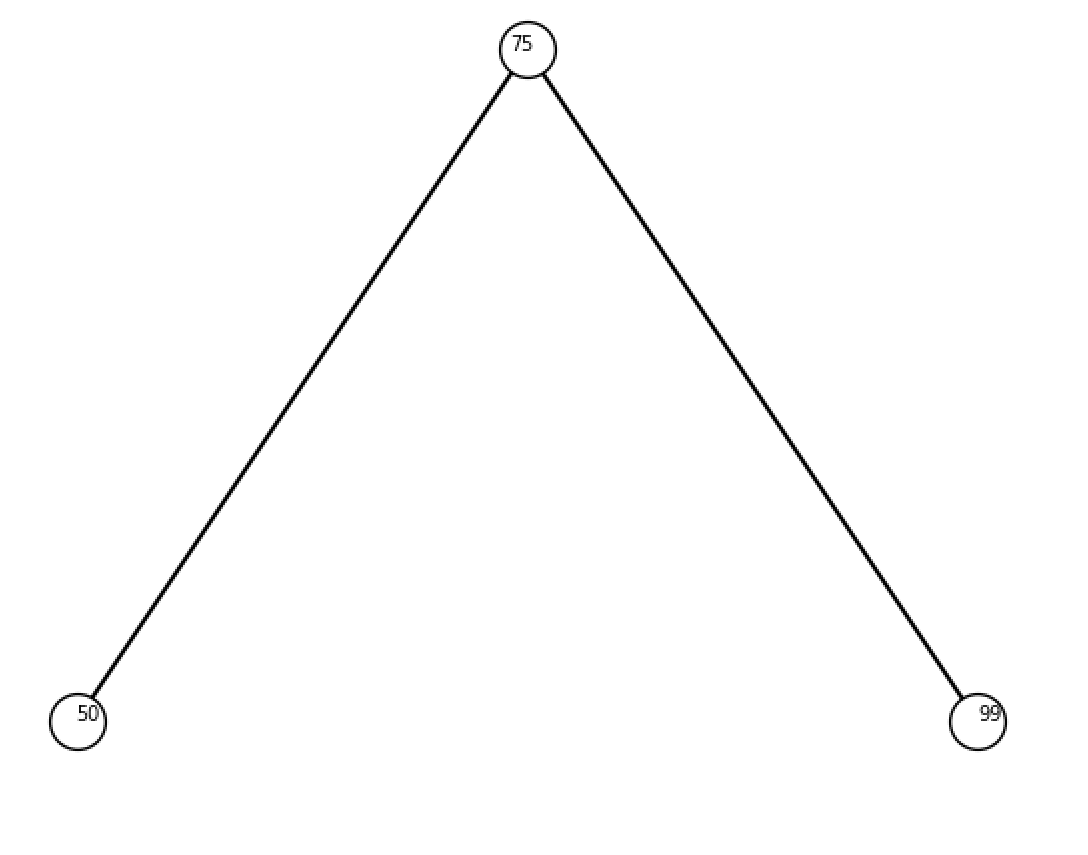
Sorted List: 1 2 3 4 5

Output:   
Sorted List:Empty

Output:

Sorted List: 50, 75, 99

Output:

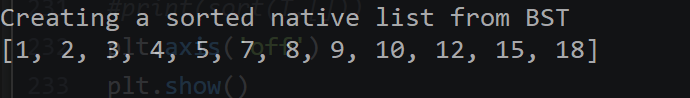


**Extracting Elements in ascending order:**

For this method we will go to the leftmost element of the tree, append that element into our list, then we will go to the parent of the child and append it, and finally move to its right child (if it has any) and append it to our list. We will do this until our BST is empty.

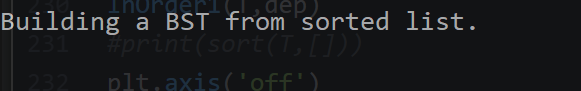
BST: [10,4,15,2,8,12,18,1,3,5,9,7]

Output:



BST: []

Output:



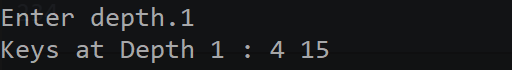
**Printing elements at depth:**

For this method we will given the depth we want to print out and the BST. While out BST has elements inside, we will move through the tree (each new level will subtract 1 from our given depth). Once our depth reaches 0 we will print out both the left and right branches element. If the depth exceeds our depth we will print nothing.

BST: [10,4,15,2,8,12,18,1,3,5,9,7]

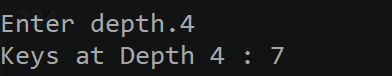
Depth: 1

Output:



Depth 4

Output:



Depth 10

Output:



What I learned in the lab was how to manipulate a BST to do various task. I also learned what happens when we feed a BST a sorted list, we must create a special set of instructions that will handle those types of problems. I also got a better understanding on how each level or depth works in a BST, and how to manipulate them to get special sets of information.

Appendix:

# Author: Rigoberto Quiroz

# Section: 1:30PM - 2:50 PM

# This program will create a BST(Binary Search Tree), and will(1) display the

# BST using matplotlib. Then(2) it will search for any key in a iterative way.

# (3) Then given a sorted native list in python it is going to create a BST.

# (4) Then it will extract the items of any BST into to list in ascending order.

# (5) Finally it will print items at a certain depth

import random

import matplotlib.pyplot as plt

import numpy as np

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

def Delete(T,del\_item):

if T is not None:

if del\_item < T.item:

T.left = Delete(T.left,del\_item)

elif del\_item > T.item:

T.right = Delete(T.right,del\_item)

else: # del\_item == T.item

if T.left is None and T.right is None: # T is a leaf, just remove it

T = None

elif T.left is None: # T has one child, replace it by existing child

T = T.right

elif T.right is None:

T = T.left

else: # T has two chldren. Replace T by its successor, delete successor

m = Smallest(T.right)

T.item = m.item

T.right = Delete(T.right,m.item)

return T

def InOrder(T):

# Prints items in BST in ascending order

if T is not None:

InOrder(T.left)

print(T.item,end = ' ')

InOrder(T.right)

def InOrderD(T,space):

# Prints items and structure of BST

if T is not None:

print(space,T.item)

InOrderD(T.right,space+' ')

InOrderD(T.left,space+' ')

def SmallestL(T):

# Returns smallest item in BST. Returns None if T is None

if T is None:

return None

while T.left is not None:

T = T.left

return T

def Smallest(T):

# Returns smallest item in BST. Error if T is None

if T.left is None:

return T

else:

return Smallest(T.left)

def Largest(T):

if T.right is None:

return T

else:

return Largest(T.right)

def Find(T,k):

# Returns the address of k in BST, or None if k is not in the tree

if T is None or T.item == k:

return T

if T.item<k:

return Find(T.right,k)

return Find(T.left,k)

def FindAndPrint(T,k):

f = Find(T,k)

if f is not None:

print(f.item,'found')

else:

print(k,'not found')

def IterativeSearch(T,key):

temp = T

# Looks for item in BST

while temp is not None:

# Key found

if temp.item == key:

return temp.item

# Move right is key is greater than root

elif key > temp.item:

temp = temp.right

else:

# move left if less

temp = temp.left

return None

def sort(T,sL):

if T is not None:

sort(T.left,sL)

#inserts BST items into list

sL.append(T.item)

sort(T.right,sL)

return sL

def InOrder1(T,h):

# Prints items in BST in ascending order

if T is not None:

#When we reach the desired depth we want to print

if h == 0:

print(T.item,end = ' ')

return

InOrder1(T.left,h-1)

InOrder1(T.right,h-1)

def depth(T):

if T is None:

return -1

# Calculates the depth of left and right branches

left = 1 + depth(T.left)

right = 1 + depth(T.right)

# returns which ever was bigger

if left > right:

return left

return right

def buildTree(L):

if L is None:

return

if len(L) <= 0:

return

# Gets midpoint index of list

midPoint = len(L)//2

# Makes the midpoint the roo

root = BST(L[midPoint])

n = 0

left = []

right = []

# Creates 2 list, midpoint is not included in neither of the list

while n < len(L)//2:

left.append(L[n])

n = n + 1

n = n + 1

while n < len(L):

right.append(L[n])

n = n + 1

# inserts the rest of the list items

root.left = buildTree(left)

root.right = buildTree(right)

return root

def printBST(T,ax,x,y,dx,dy):

if T is None:

return

# If the item is a leaf

if T.left is None and T.right is None:

ax.text(x,y,T.item,fontsize=7)

ax.plot(x,y,'o',markersize=20,markeredgecolor='black',markerfacecolor='white')

return

# if we only have 1 children, right or left

if T.left is None:

ax.plot((x,x+dx),(y,y-dy),color='k')

ax.text(x-1,y-1,T.item,fontsize=7)

ax.plot(x,y,'o',markersize=20,markeredgecolor='black',markerfacecolor='white')

printBST(T.right,ax,x+dx,y-dy,dx/2,dy)

return

if T.right is None:

ax.plot((x,x-dx),(y,y-dy),color='k')

ax.text(x,y,T.item,fontsize=7)

ax.plot(x,y,'o',markersize=20,markeredgecolor='black',markerfacecolor='white')

printBST(T.left,ax,x-dx,y-dy,dx/2,dy)

return

# if we have both children

ax.plot((x,x-dx),(y,y-dy),color='k')

ax.plot((x,x+dx),(y,y-dy),color='k')

ax.text(x-1,y-1,T.item,fontsize=7)

ax.plot(x,y,'o',markersize=20,markeredgecolor='black',markerfacecolor='white',label=T.item)

printBST(T.left,ax,x-dx,y-dy,dx/2,dy)

printBST(T.right,ax,x+dx,y-dy,dx/2,dy)

# Code to test the functions above

T = None

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

#A =[]

#A = [1,2,3,4]

for a in A:

T = Insert(T,a)

plt.close("all")

B = None

#L = [1,2,3,4,]

#B = buildTree(L)

x = 50

y = 100

dx = x / 2

dy = y \* len(A)

fig, ax = plt.subplots()

print('Displaying BST Tree')

printBST(T,ax,x,y,dx,dy)

print('\n')

key = int(input('Enter key to search for in BST.'))

print('Looking for',key, 'in BST.')

print(IterativeSearch(T,key))

print('\n')

print('Building a BST from sorted list.')

L = [50, 75, 99]

B = buildTree(L)

#printBST(B,ax,x,y,dx,dy)

print('\n')

print('Creating a sorted native list from BST')

print(sort(T,[]))

print('\n')

dep = int(input('Enter depth.'))

print("Keys at Depth", dep,':',end=' ')

InOrder1(T,dep)

#print(sort(T,[]))

plt.axis('off')

plt.show()