Lab 3 : Binary Trees

Steven Robles [*]

University of Texas, El Paso, TX

March 12, 2019

This paper aims to demonstrate implementations of a Binary Tree algorithm. To illustrate the structure of the binary tree, the structure is plotted using the matplot library in python. The binary trees can be constructed in two ways: unbalanced or balanced. Both are demonstrated in this lab as well as constructing a sorted array by transversing a binary tree.

;

Contents

I Introduction **1**

II Design **2**

A Part 1: Plotting the Tree 2
B Part 2: Search by Iteration 2
C Part 3: Balanced Tree 2
D Part 4: Sorted Array 2
E Part 5: Depth Display 3

III Results **3**

IV Discussion **3**

V Source Code **4**

VI Academic Dishonesty **9**

I. Introduction

Binary trees are an important topic in computer science. They are used to sort and search data in an efficient manner, typically in $O(logN)$ for the best case scenarios and $O(N)$ for worst case scenarios. They are termed binary since the tree is compramised of a single node with a maximum of two children. In this particular set of BST (Binary Search Tree), the left child's item will be smaller than the parent's, and the right child's item will be greater or equal to the parent's.

To demonstrate the different implementations of a BST, two types are constructed in this lab. The first type is a regular BST. This BST content's are organized based upon the order of which they are inserted. The second type of BST demonstrated is a balanced BST. A balanced BST is organized so that the lowest depth of the BST is never heavy one one side. This type of BST makes the sorting the data in order possible.

[*]ID Number: 80678755

II. Design

This lab consisted of many parts, each with their own challenges. The main focus, although, revolved around the construction of a binary search tree. One of the major challenges involved in this lab was printing the binary tree as a figure utilizing the matplot library in python. Other revolved around building the functions, as they are implemented as recursive calls.

A. Part 1: Plotting the Tree

Plotting the Binary Search Tree (BTS) was a major challenge of this lab. It combined many elements used in a previous lab which also deal with drawing figures through recursion. The first step taken to draw the figure was designing the placement of the circles which contains a node's item value. The placement of these circles were crucial since they determined the overall structure of the tree as well as signify a present node. The mathematical formula used to draw a circle base on x and y coordinates are shown below.

$$C_{Left}(x, y) = (x - (width_{n-1} * 0.50), y - height) \tag{1}$$

$$C_{Right}(x, y) = (x + (width_{n-1} * 0.50), y - height) \tag{2}$$

In equations (1) and (2), with represents the change of the x value of the center point as well as the height in the y value. These pair of equations closely resembles to that of a previous lab since both recursive functions are designed to build a tree-like structure. To draw the connected lines between the nodes to create the *branches*, a separate none-recursive helper function is called. The following shows the mathematical model used to draw the lines.

$$P_{Bottom}(x, y) = (x - (radius * sin(angle)), y - (radius * cos(angle))) \tag{3}$$

$$P_{Top}(x, y) = (x + (radius * sin(angle)), y + (radius * cos(angle))) \tag{4}$$

The *angle* shown in the previous two equations are constants with values of 1.5 or 4.5 radians, depending if the line is to be drawn to the left or right of the original circle. These two functions allowed the program where to start and stop the line effectively preventing the line to be draw within the circle.

B. Part 2: Search by Iteration

The section of the lab had the least challenge. It just demonstrated how using a recursive method can reduce the complexity of a function. The complexity of a recursive search function of a BST is $O(logN)$ while a iteration version of the search function is $O(N)$. In this case, a while loop is used to keep searching a binary tree.

C. Part 3: Balanced Tree

A balanced tree is designed to have its nodes in order throughout the tree. The goal of this section was to construct a balanced tree in $O(N)$ time. In order to achieve this, a in order transversal approach was utilized. Since a sorted array is given, the recursive function only stored the middle element of the array. Repeating this process allows that the current node will have the same amount of children left and right.

D. Part 4: Sorted Array

The object of this section of the lab is to build a sorted array from a binary search tree. In order to accomplish this in $O(N)$ time, an in order transversal approach was taken again. The in order transversal returns the items of a binary tree in order as implied by the name. After realizing this, there was little to no challenge in this sections. Although, a problem arouse when needing to create an empty list with the length matching the number of items in the binary search tree. This was solved by counting the number of items within the tree beforehand.

E. Part 5: Depth Display

Following after Part 1 and 3, this came to be the third most challenging section of the lab. One of the biggest problems faced in this section is tracking which elements to be printed at depth *d*. In order to meet the requirements of this sections, a seperate helper function was built. The helper function is recursive which transvereses through the given tree and only prints the items at depth *d*.

III. Results

After the successful implementations of the previous parts of the lab, all of the trees created can be displayed via by plotting them using part 1. These results are based upon the given tree that came with the lab, but a user interface implemented in the program allows for many trees to be created and tested. Below shows the trees from the previous parts.

Figure 1: Original Tree

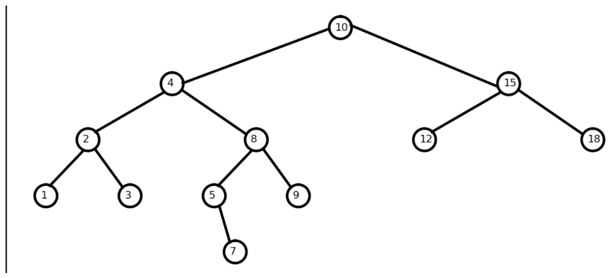


Figure 2: Balanced Tree

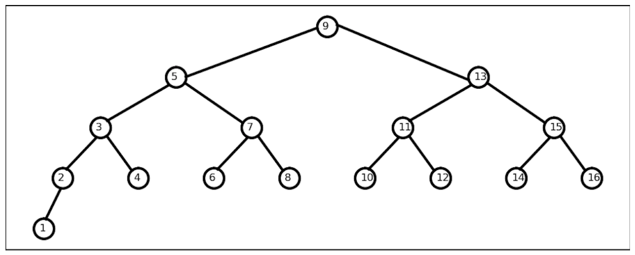
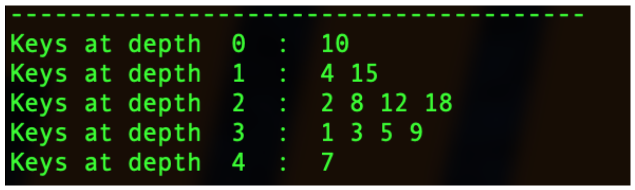


Figure 3: Depth Display



```
---------------------------------------
Keys at depth  0  :  10
Keys at depth  1  :  4 15
Keys at depth  2  :  2 8 12 18
Keys at depth  3  :  1 3 5 9
Keys at depth  4  :  7
```

Note that the first figure only displays the original given tree. This process can be repeated with any random set of items. A sample is also is given for tha last section of the lab. It displays which keys are present at depth level *d* which refers to the original tree.

IV. Discussion

Overall, implementing binary trees is an extensive processes, but much more cost effective as demonstrated in section two of the lab. Both versions of a binary search tree is successfully constructed and drawn.

V. Source Code

```python
"""
Author: Steven J. Robles
Class: CS 2302 Data Structures III
Instructor: Olac Fuentes
TA: Anindita Nath And Maliheh Zargaran
Last Modified: 03/08/2019
Discreption: Lab 3:
    The purpose of this lab is to construct balanced binary search trees and arrays, as well as plotting the trees
    using matplotlib. Build into the program is a timer to evluate the preforamnce of the functions.
"""

from TreeBuilder import BuildTree
from TreeBuilder import NewObject
from TreeBuilder import Search
from TreeBuilder import PrintAtDepth
from PrintTrees import PrintTree
from Balanced import BalancedArray
from Balanced import BalancedTree
import random

#the function gest the n number of items the user wants to insert
def GetUserLength():
  lengthIn = input("Input Number Of Items That Are Going To Be Inserted\n")
  return int(lengthIn)


#declaures a BST node for scope use
T = NewObject(None)
loop = True

#the following loop executes the commands the sure wishes to follow
while loop:
  #ask user input from options
  print("Hello! Do you want to proceed with?\n1. Construct Origional Tree\n2. Generate Own Tree")
  print("3. Generate Random Tree With N Items\n4. Build Balanced Tree With Sorted Array Numbers\n5. Print Constructed Tree")
  print("6. Search Tree for k\n7. Print According To Depth\n8. Build A Sorted Array From Constructed Tree")
  choice = input("9. Exit\n")
  try: #the try and except functions to convert input into an integer unless the input isnt a number
    choiceNum = int(choice)
  except ValueError:
    choiceNum = -1
  print("--------------------------------------------")

  if (choiceNum == 1): #generates the origional BST
    A = [10, 4, 15, 2, 8, 12, 18, 1, 3, 5, 9, 7]
    T = BuildTree(A)

  elif(choiceNum == 2): #geneates the user's BST
    length =  GetUserLength()
    print("Keep Entering Desired Item To Be Inserted.")
    i = 1
    A = [-1]* length
    while i <= length:
      print("Item ", i, end = "")
      itemIN = input(": ")
      item = int(itemIN)
      A[i-1] = item
      i+=1
    T = BuildTree(A)

  elif(choiceNum == 3): #Generates a random tree with N items
```

```
    length = GetUserLength()
    A = [-1]* length
    for i in range(length):
      A[i] = random.randint(1,101)
    T = BuildTree(A)

  elif(choiceNum == 4): #Builds a balanced tree with a sorted list of N items
    length = GetUserLength()
    A = [-1]* length
    for i in range(length):
      A[i] = i+1
    T = BalancedTree(A)

  elif(choiceNum == 5): #prints constructed tree
    if T.item is not None:
      PrintTree(T)
    else:
      print("Consturct A Tree First")

  elif(choiceNum == 6): #Searches a constructed tree for item K
    if T.item is not None:
      kIn = input("Enter item desired to be search in the BST : \n")
      k = int(kIn)
      G = Search(T, k)
      if G is None:
        print("Item Not Found in BST")
      else:
        print("Item ", G.item, " Found in BST")
    else:
      print("Consturct A Tree First")

  elif(choiceNum == 7): #Prints the items in the tree according to the depth level
    if T.item is not None:
      PrintAtDepth(T)
    else:
      print("Consturct A Tree First")

  elif(choiceNum == 8): #Builds a sorted array from a constructed tree
    if T.item is not None:
      BalancedArray(T)
    else:
      print("Consturct A Tree First")

  elif(choiceNum == 9): #exits the program
    print("Goodbye!")
    loop = False
  else:       #allows for re-entry incase of missed input
    print("Try Again")
  print("----------------------------------------")
```

```
"""
Author: Steven J. Robles
Class: CS 2302 Data Structures III
Instructor: Olac Fuentes
TA: Anindita Nath And Maliheh Zargaran
Last Modified: 03/08/20199
Discreption: Lab 3:
    The purpose of this program is to construct a balanced binary search tree and fill a sorted array
    with elements of a binary search tree. Each function is checked by pring its results.

"""

from TreeBuilder import CountItems
from TreeBuilder import BuildTree
from TreeBuilder import NewObject
from PrintTrees import PrintTree
from TreeBuilder import PrintAtDepth


```

```python
# **** Part 3 ****
#the following is the recursive function that consturcts a blanced tree.
def FillTree( ar):
  if not ar:  #checks if the array is empty ar[]
    return None
  mid =len(ar)//2 #sets the middle node as the furtre parent
  T = NewObject(ar[mid])
  T.left = FillTree(ar[:mid])
  T.right = FillTree(ar[mid+1:])
  return T

#the following function is called from the main profile
def BalancedTree(arr):
  T = FillTree(arr)
  return T

# **** Part 4 ****
#the following recrusive function is utlized to fill a sorted array from a binary tree
def FillArray(T,ar, i):
  if T is not None:
    i = FillArray(T.left , ar, i)
    ar[i] = T.item
    i = FillArray(T.right ,ar, i+1)
    return i
  return i

#the following function is called from the main file
def BalancedArray(T):
  length =  CountItems(T) #these two line create the array with the right length
  SortFill = [0] * length
  FillArray(T, SortFill , 0) #recrursive inital call

  for i in range (length):
    print(SortFill[i], end = ' ')
  print()
```

```python
"""
Author: Steven J. Robles
Class: CS 2302 Data Structures III
Instructor: Olac Fuentes
TA: Anindita Nath And Maliheh Zargaran
Last Modified: 03/08/2019
Discreption: Lab 3:
        The purpose of this program is to draw a given BST using matplot. The porgram graws cirlces and the
        appropiet branch that connects the tree with the corresponding item within.

"""
import numpy as np
import matplotlib.pyplot as plt
import math

#plots the cirlce using its radius
def circle(center,rad):
    n = int(10*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

#the following function draws the branch which connects a child to its parent on the tree
def DrawBranch(ax,T, p, center1, center2, angle):
    if T is not None:
        p[0, 0] = center1[0] - 20*np.sin(angle)
        p[0, 1] = center1[1] - 20*np.cos(angle)
        p[1, 0] = center2[0] + 20*np.sin(angle)
        p[1, 1] = center2[1] + 20*np.cos(angle)
        ax.plot(p[:,0],p[:,1],color='k')

#the followin recursive function is the main function in which circles are set and ploted
```

```python
def DrawTree(ax,T,center1 , p , w, h, wd, angle):
    if T is not None:

        #the following setes the position for the left child
        center2 = center1 * 1
        center2[0] -=w * wd
        center2[1] -=h
        x,y = circle(center1, 20)
        ax.plot(x,y,color='k')  #only plots the current node (a.k.a the parent)
        ax.text(center1[0]-10, center1[1]-5, T.item, fontsize=6) #inserts the number into
    the cirlce
        DrawBranch(ax,T.left ,p, center1 ,center2, angle)
        DrawTree(ax,T.left ,center2, p, w, h, wd*w, angle*w)

        #the folloiwng sets up the location of the right child
        center2 = center1 * 1
        center2[0] +=w * wd
        center2[1] -=h
        DrawBranch(ax,T.right, p ,center1 ,center2, 6-angle)
        DrawTree(ax,T.right,center2, p, w, h, wd*w, angle*w)


#This definition is called from the main program of which initiates the recursive print call
    .
def PrintTree(T):
    fig, ax = plt.subplots()
    ax.axis('on')
    ax.set_aspect(1.0)
    ax.yaxis.set_major_locator(plt.NullLocator())
    ax.xaxis.set_major_locator(plt.NullLocator())
    height = 100
    width = 600
    line = np.array([[0,0],[2000,2000]])

    DrawTree(ax,T,[1000, 1000], line,.50, height, width, 1.5)
    plt.show()
    fig.savefig('Tree.png')
```

```python
"""
Author: Steven J. Robles
Class: CS 2302 Data Structures III
Instructor: Olac Fuentes
TA: Anindita Nath And Maliheh Zargaran
Last Modified: 03/08/2019
Discreption: Lab 3:
        The purpse of this program is to construct the inital binary search tree that is not
    balanced. This tree is
        used to fill a sorted array in Balanced.py which satisfies part 4 of the lab.

"""

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

#retursn an empty BST node
def NewObject(item):
    T = None
    T = BST(item)
    return T

#returns the number of items in a BST
def CountItems (T):
    if T is not None:
        count = 1
        count += CountItems(T.right)
        count += CountItems(T.left)
        return count
    return 0

def BuildTree(A):
    T = None
    for a in A:
        T = Insert(T,a)
    return T

#part 2 of the lab which searches an item in a BST using an irative approach
def Search(T, k):
    while T is not None:
        if T.item == k:
            return T
        if T.item < k:
            T = T.right
        elif T.item > k :
            T = T.left
    return T

#helper function for part 5 which prints the item only if the depth level is correct
def PrintDepthKeys(T, d, i):
    if T is not None:
        if d == i:
            print(T.item, end = ' ')
            if T.left is None and T.right is None: #checks if this is a leaf so it will not
    loop again
                return False
            return True
        if d < i:
            check1 = PrintDepthKeys(T.left, d+1, i)
            check2 = PrintDepthKeys(T.right, d+1, i)

            if check1 or check2:
                return True #returns true if the internal node as at least one childe
    return False

#this is part 5 of the lab which prints all elements located at depth 'd'
def PrintAtDepth(T):
    i = 0
    loop = True
    while loop:
        print("Keys at depth ", end = ' ')
        print(i, end = ' ')
        print(" : ", end = ' ')
        loop = PrintDepthKeys(T, 0, i)
        print()
        i+=1
```

VI. Academic Dishonesty

Scholastic Dishonesty

Any student who commits an act of scholastic dishonesty is subject to discipline. Scholastic dishonesty includes, but not limited to cheating, plagiarism, collusion, the submission for credit of any work or materials that are attributable to another person.

- **Cheating**
  - Copying form the test paper of another student
  - Communicating with another student during a test
  - Giving or seeking aid from another student during a test
  - Possession and/or use of unauthorized materials during tests (i.e. Crib notes, class notes, books, etc)
  - Substituting for another person to take a test
  - Falsifying research data, reports, academic work offered for credit
- **Plagiarism**
  - Using someone's work in your assignments without the proper citations
  - Submitting the same paper or assignment from a different course, without direct permission of instructors
- **Collusion**
  - Unauthorized collaboration with another person in preparing academic assignments

Sign: [illegible] **Date: 03/12/2019**