**Lab 3**

This code has the purpose of introducing the inner workings of a binary search tree, traversing and searching with iterative methods, drawing the tree, creating Trees with sorted lists and then creating sorted lists given a Tree The DrawBST() method uses recursion to traverse the Tree and draws a circle for every node, then it draws the lines connecting child to parent. Isearch() uses iterative loops to search for a given number n in the tree, if it is found it will return the Node containing it, else it will return None. BuildT() receives a sorted python native list and turns it into a balanced tree by recursively adding each number to the tree, starting with the left branch, then adding that to the root center value and finally doing the right side of the tree to add. Extract() receives a Tree object and extracts each item into a sorted list, it does this by getting to the smallest element and then recursively appending them to a list, then recursively going through the elements that are bigger than the root.

**Experimental Results**

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

**Tree T, from list A**

|  |  |
| --- | --- |
| Input | Output |
| DrawBST(T) |  |
| ISearch(T, 7)  ISearch(T, 15) | **7**  **None** |
| BuildT(A) | **With InOrderD(L, ‘ ‘)**  **18**  **15**  **12**  **10**  **9**  **8**  **7**  **5**  **4**  **3**  **2**  **1** |
| Extract(T, B) | **B = [1, 2, 3, 4, 5, 7, 8, 9, 10, 12, 15, 18]** |
| i = [0,1,2,3,4]  PrintAtDepth(T, i) | **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** |

**List A =** **[70, 50, 90, 130, 150, 40, 10, 30, 100, 180, 45, 60, 140, 42]**

**Tree T, from list A**

|  |  |
| --- | --- |
| Input | Output |
| DrawBST(T) |  |
| ISearch(T, 150)  Isearch(T, 110) | **150**  **None** |
| BuildT(A) | **With InOrderD(L, ‘ ‘)**  **180**  **150**  **140**  **130**  **100**  **90**  **70**  **60**  **50**  **45**  **42**  **40**  **30**  **10** |
| Extract(T, B) | **B = [10, 30, 40, 42, 45, 50, 60, 70, 90, 100, 130, 140, 150, 180]** |
| I = [0,1,2,3,4]  PrintAtDepth(T, i) | **Keys at Depth 0 : 70**  **Keys at Depth 1 : 50 90**  **Keys at Depth 2 : 40 60 130**  **Keys at Depth 3 : 10 45 100 150**  **Keys at Depth 4 : 30 42 140 180** |

**Conclusion**

Even though Iterative methods can be made to work, Recursion is the best option for traversing a BST at any time. If information is properly entered, BSTs can quickly organize and find information in them.

**Source Code**

**# Code to implement a binary search tree**

**# Programmed by Hugo Chavez**

**# Last modified March 11, 2019**

**import matplotlib.pyplot as plt**

**import numpy as np**

**import math**

**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**

**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:**

**InOrderD(T.right,space+' ')**

**print(space,T.item)**

**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 height(T):**

**if T is None:**

**return 0**

**else:**

**ldepth= 1 + height(T.left)**

**rdepth = 1 + height(T.right)**

**if ldepth < rdepth:**

**return rdepth**

**else:**

**return ldepth**

**def DrawBST(ax,T,x,y,dx):**

**if T is not None:**

**dx = dx/2**

**circle1 = plt.Circle((x,y), 3, color='k', fill = False)**

**ax.add\_artist(circle1)**

**plt.text(x-1, y-1, T.item, fontsize = 10)**

**if T.left is not None:**

**DrawBST(ax, T.left ,x-dx,y-10,dx)**

**p = np.array([[x,y-3],[x-dx,y-7]])**

**ax.plot(p[:,0],p[:,1],color='k')**

**if T.right is not None:**

**DrawBST(ax, T.right ,x+dx,y-10,dx)**

**q = np.array([[x,y-3],[x+dx,y-7]])**

**ax.plot(q[:,0],q[:,1],color='k')**

**def ISearch(T,k):**

**if k <= T.item:**

**while T is not None:**

**if T == None:**

**return None**

**if k == T.item:**

**return T**

**if k < T.item:**

**T = T.left**

**if k > T.item:**

**T = T. right**

**elif k >= T.item:**

**while T is not None:**

**if T == None:**

**return None**

**if k == T.item:**

**return T**

**if k < T.item:**

**T = T.left**

**if k > T.item:**

**T = T. right**

**def BuildT(A, start, end):**

**if start > end:**

**return None**

**L = None**

**mid = (start + end)//2**

**L = BST(A[mid])**

**L.left = BuildT(A, start, mid-1)**

**L.right = BuildT(A, mid+1, end)**

**return L**

**def Extract(T, A):**

**if T is None:**

**return**

**Extract(T.left, A)**

**A.append(T.item)**

**Extract(T.right, A)**

**return A**

**def PrintAtDepth(T, depth):**

**if T is None:**

**return**

**if depth ==0:**

**print(T.item, end = ' ')**

**else:**

**PrintAtDepth(T.left, depth -1)**

**PrintAtDepth(T.right, depth -1)**

**# Code to test the functions above**

**T = None**

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

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

**for a in A:**

**T = Insert(T,a)**

**A.sort()**

**#-----------------------------------------------------------------------------**

**plt.close("all")**

**fig, ax = plt.subplots()**

**DrawBST(ax,T, 50,97,50)**

**ax.set\_xlim((0, 100))**

**ax.set\_ylim((0, 100))**

**plt.show()**

**fig.savefig('bst.png')**

**print('-----Iterative Search-----')**

**X = ISearch(T, 15)**

**if X is not None:**

**print(X.item)**

**elif X == None:**

**print('None')**

**print('----Build List-----')**

**print(A)**

**L = BuildT(A, 0, len(A)-1)**

**InOrderD(L,' ')**

**#DrawBST(ax,L, 50,97,50)**

**print('----Extract tree-----')**

**print('Original list')**

**print(Y)**

**B = []**

**B = Extract(T, B)**

**print('Extracted List')**

**print(B)**

**print('----Keys at Depth----')**

**depth = height(T)**

**for i in range(depth):**

**print('Keys at Depth', i, ':', end = ' ')**

**PrintAtDepth(T, i)**

**print()**

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.

\_\_\_\_\_\_\_\_\_\_\_\_Hugo Chavez\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_