Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет Інформатики та Обчислювальної Техніки

Кафедра Автоматизованих Систем Обробки Інформації та Управління

Лабораторна робота № 8

з дисципліни «Теорія алгоритмів»

на тему:

**"** **Бінарні дерева пошуку"**

Виконав:

студент гр. ІС-02

Плостак Ілля

Викладач:

Cт. Вик. Новікова П. А.

Київ – 2021

**1. Завдання**

У даній роботі необхідно виконати два завдання.

1. Перетворити вхідне бінарне дерево у бінарне дерево пошуку
2. Пошук сум послідовних вузлів в дереві

**2. Програмний код:**

[main.py](https://github.com/feedblackg44/kpilabs2/blob/master/TA/Lab8/main.py):

from BinarySearchTree import BinarySearchTree, Node, BinaryTree  
  
temp = open("input.txt", "r").read().split(" ")  
arr = [int(temp[i]) for i in range(len(temp))]  
  
Tree = BinaryTree(arr)  
print("Inorder walk on generated binary tree:")  
Tree.InorderTreeWalk(Tree.root)  
print()  
  
f = open("output.txt", "w")  
SearchTree = BinarySearchTree(Tree, f)  
print("Inorder walk on generated binary search tree:")  
SearchTree.InorderTreeWalk(SearchTree.root)  
print()  
  
print("Write needed sum:")  
S = int(input())  
print("All paths by sum", S, "is in file 'output.txt'.")  
SearchTree.PrintPathsBySum(SearchTree.root, S)  
  
f.close()

[BinarySearchTree.py](https://github.com/feedblackg44/kpilabs2/blob/master/TA/Lab8/BinarySearchTree.py):

from BinaryTree import BinaryTree, Node  
  
class BinarySearchTree:  
 def \_\_init\_\_(self, BinTree, file):  
 self.tempPath = []  
 self.pathList = [[]]  
 self.root = BinTree.root  
 self.iterator = 0  
 self.file = file  
 BinTree.CreateArrForSort(BinTree.root)  
 temp = self.NumbersFrom(BinTree.arrForSort)  
 temp.sort()  
 self.sorted = temp  
 self.SearchTreeCreate(self.root)  
  
 def InorderTreeWalk(self, x):  
 if x != None:  
 self.InorderTreeWalk(x.left)  
 print(x.key, end=" ")  
 self.InorderTreeWalk(x.right)  
  
 def SearchTreeCreate(self, x):  
 if x != None:  
 self.SearchTreeCreate(x.left)  
 if x.key != None and x.key != 0:  
 x.key = self.sorted[self.iterator]  
 self.iterator += 1  
 self.SearchTreeCreate(x.right)  
  
 def NumbersFrom(self, arr):  
 arrOut = []  
 for i in range(len(arr)):  
 if arr[i] != None and arr[i] != 0:  
 arrOut.append(arr[i])  
 return arrOut  
  
 def FindPathsBySum(self, root, path, S):  
 if (not root):  
 return  
 path.append(root.key)  
 self.FindPathsBySum(root.left, path, S)  
 self.FindPathsBySum(root.right, path, S)  
 f = 0  
 for j in range(len(path) - 1, -1, -1):  
 f += path[j]  
  
 if (f == S):  
 self.PrintVector(path, j)  
 path.pop(-1)  
  
 def PrintPathsBySum(self, root, S):  
 path = []  
 self.FindPathsBySum(root, path, S)  
  
 def PrintVector(self, v, i):  
 for j in range(i, len(v)):  
 self.file.write(str(v[j]) + " ")  
 self.file.write("\n")

[BinaryTree.py](https://github.com/feedblackg44/kpilabs2/blob/master/TA/Lab8/BinaryTree.py):

from Node import Node  
  
class BinaryTree:  
 def \_\_init\_\_(self, arr):  
 self.root = None  
 self.count = 0  
 self.arrForSort = []  
 self.GenerateTree(arr)  
 self.Kill0Elems(self.root)  
  
 def InorderTreeWalk(self, x):  
 if x != None:  
 self.InorderTreeWalk(x.left)  
 print(x.key, end=" ")  
 self.InorderTreeWalk(x.right)  
  
 def CreateArrForSort(self, x):  
 if x != None:  
 self.CreateArrForSort(x.left)  
 self.arrForSort.append(x.key)  
 self.CreateArrForSort(x.right)  
  
 def Kill0Elems(self, x):  
 if x != None:  
 self.Kill0Elems(x.left)  
 if x.key == 0:  
 if x.parent.left == x:  
 x.parent.left = None  
 elif x.parent.right == x:  
 x.parent.right = None  
 self.Kill0Elems(x.right)  
  
 def GenerateTree(self, arr):  
 parent = None  
 for i in range(len(arr)):  
 node = Node(0)  
 node.key = arr[i];  
 node.parent = parent;  
 if node.key == 0:  
 parent = node.parent  
 else:  
 parent = node  
  
 if node.parent != None:  
 flag = True;  
 while flag:  
 if node.parent.left == None:  
 node.parent.left = node  
 flag = False  
 elif node.parent.right == None:  
 node.parent.right = node  
 flag = False  
 else:  
 node.parent = node.parent.parent  
 if i == 0:  
 self.root = node

[Node.py](https://github.com/feedblackg44/kpilabs2/blob/master/TA/Lab8/Node.py):

class Node:  
 def \_\_init\_\_(self, key):  
 self.key = key  
 self.parent = None  
 self.left = None  
 self.right = None

**3. Приклад вхідного файлу та приклад вихідного файлу:**

input.txt:



output.txt:



**4. Висновок:**

В цій лабораторній роботі ми вивчили нову структуру даних: бінарні дерева пошуку (BST). Спочатку ми перетворили довільне бінарне дерево у бінарне дерево пошуку. Та за допомогою BST ми вирішили наступну задачу: *Пошук сум послідовних вузлів в дереві.*

Створений алгоритм працює таким чином: він перебирає усі можливі шляхи зверху до низу від кореня до кожного елементу дерева та для кожного з цих шляхів рахує усі можливі суми «хвостів» цих шляхів порівнюючи за потрібною сумою. Таким чином перевіряються усі можливі шляхи та знаходяться лише потрібні для даної суми.

Алгоритм є не найоптимальнішим, оскільки перебирає усі можливі суми не враховуючи основні властивості BST. Однак для оптимізації даного алгоритму потрібен зовсім інший підхід, оскільки пошук тут здійснюється фактично знизу уверх, що робить неможливим використання основних властивостей BST.