Львівський національний університет імені Івана Франка

Факультет електроніки та комп’ютерних технологій

Звіт про виконання лабораторної роботи №1

Теоретичні відомості. Динамічні структури даних. Нелінійні

списки (дерева): загальні положення

Виконав:

Студент групи ФЕП-22

Серафим Д.В.

Львів-2022

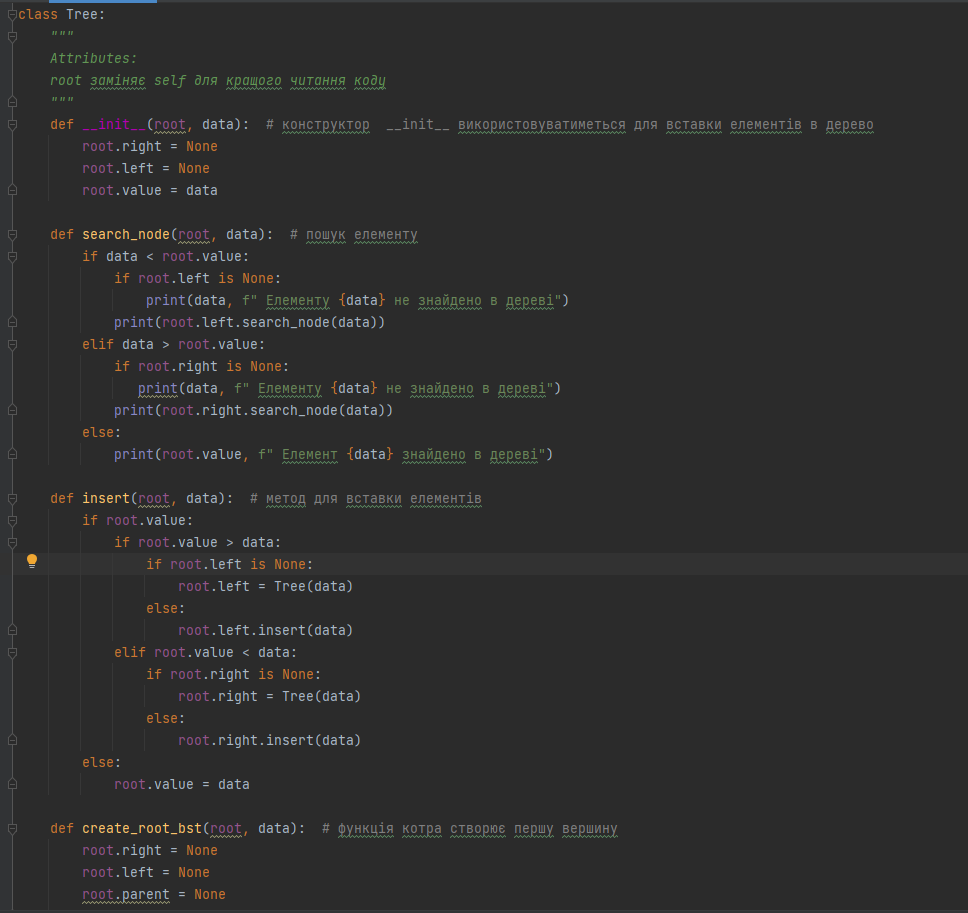
**Хід роботи:**

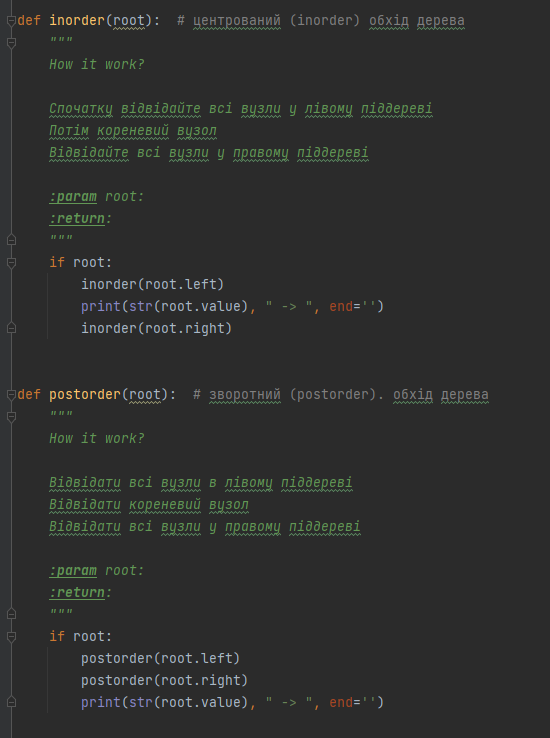
**Частина 1. Побудова та обхід бінарного дерева**

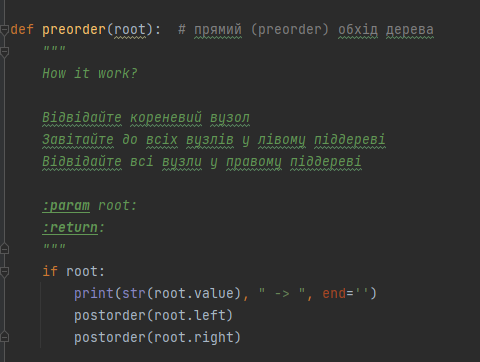
1. Створити нову бібліотеку Tree (файли Tree.h, Tree.cpp).

2. У файлі Tree.h оголосити структуру Node що задає вузол дерева, та за допомогою команди typedef зв’язати тип інформаційного поля вузла дерева datatype з типом даних, заданим викладачем.

3. У бібліотеці Tree, згідно описаних вище алгоритмів, реалізувати функції CreateTree(…) та ShowTree(…) для створення та відображення бінарного дерева. У цьому ж файлі реалізувати функції обходу дерева: PrefixOrder(…), PostfixOrder(…) та InfixOrder(…)



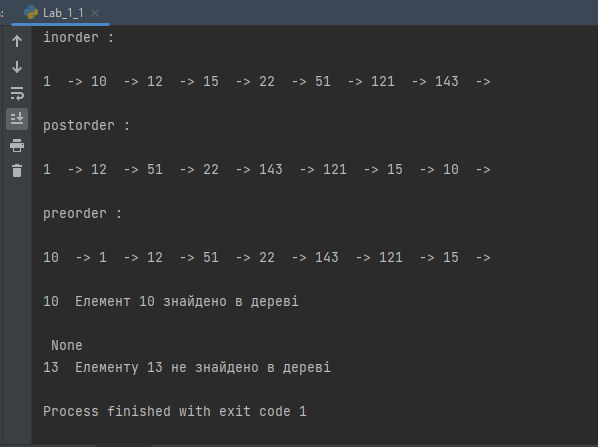




4. Створити новий проект Lab\_1\_1 та підключити до нього бібліотеку Tree. У функції main() проекту реалізувати меню для виконання операцій створення, відображення та обходу дерева.

class Tree:  
 *"""  
 Attributes:  
 root заміняє self для кращого читання коду  
 """* def \_\_init\_\_(root, data): # конструктор \_\_init\_\_ використовуватиметься для вставки елементів в дерево  
 root.right = None  
 root.left = None  
 root.value = data  
   
 def search\_node(root, data): # пошук елементу  
 if data < root.value:  
 if root.left is None:  
 print(data, f" Елементу {data} не знайдено в дереві")  
 print(root.left.search\_node(data))  
 elif data > root.value:  
 if root.right is None:  
 print(data, f" Елементу {data} не знайдено в дереві")  
 print(root.right.search\_node(data))  
 else:  
 print(root.value, f" Елемент {data} знайдено в дереві")  
   
 def insert(root, data): # метод для вставки елементів  
 if root.value:  
 if root.value > data:  
 if root.left is None:  
 root.left = Tree(data)  
 else:  
 root.left.insert(data)  
 elif root.value < data:  
 if root.right is None:  
 root.right = Tree(data)  
 else:  
 root.right.insert(data)  
 else:  
 root.value = data  
   
 def create\_root\_bst(root, data): # функція котра створює першу вершину  
 root.right = None  
 root.left = None  
 root.parent = None  
 root.value = data  
  
  
def inorder(root): # центрований (inorder) обхід дерева  
 *"""  
 How it work?  
  
 Спочатку відвідайте всі вузли у лівому піддереві  
 Потім кореневий вузол  
 Відвідайте всі вузли у правому піддереві* ***:param*** *root:* ***:return****:  
 """* if root:  
 inorder(root.left)  
 print(str(root.value), " -> ", end='')  
 inorder(root.right)  
  
  
def postorder(root): # зворотний (postorder). обхід дерева  
 *"""  
 How it work?  
  
 Відвідати всі вузли в лівому піддереві  
 Відвідати кореневий вузол  
 Відвідати всі вузли у правому піддереві* ***:param*** *root:* ***:return****:  
 """* if root:  
 postorder(root.left)  
 postorder(root.right)  
 print(str(root.value), " -> ", end='')  
  
  
def preorder(root): # прямий (preorder) обхід дерева  
 *"""  
 How it work?  
  
 Відвідайте кореневий вузол  
 Завітайте до всіх вузлів у лівому піддереві  
 Відвідайте всі вузли у правому піддереві* ***:param*** *root:* ***:return****:  
 """* if root:  
 print(str(root.value), " -> ", end='')  
 postorder(root.left)  
 postorder(root.right)  
  
  
root = Tree(10) # створення екземпляру класу  
root.insert(15) # вставка елемента в бінарне дерево  
root.insert(121) # вставка елемента в бінарне дерево  
root.insert(22) # вставка елемента в бінарне дерево  
root.insert(12) # вставка елемента в бінарне дерево  
root.insert(143) # вставка елемента в бінарне дерево  
root.insert(1) # вставка елемента в бінарне дерево  
root.insert(51) # вставка елемента в бінарне дерево  
print("inorder :\n")  
inorder(root) # центрований (inorder) обхід дерева  
print("\n\npostorder :\n")  
postorder(root) # зворотний (postorder). обхід дерева  
print("\n\npreorder :\n")  
preorder(root) # прямий (preorder) обхід дерева  
  
print(f"\n", root.search\_node(10))

5. Відкомпілювати проект та продемонструвати його роботу для данихнелінійного списку (дерева), отриманих від викладача.



inorder :

1 -> 10 -> 12 -> 15 -> 22 -> 51 -> 121 -> 143 ->

postorder :

1 -> 12 -> 51 -> 22 -> 143 -> 121 -> 15 -> 10 ->

preorder :

10 -> 1 -> 12 -> 51 -> 22 -> 143 -> 121 -> 15 ->

10 Елемент 10 знайдено в дереві

None

13 Елементу 13 не знайдено в дереві

**Частина 2. Бінарне дерево пошуку**

1. У бібліотеці Tree, згідно описаних вище алгоритмів, реалізувати функції SearchNodeBST(…), CreateRootBST(…), InsertNodeBST(…), DeleteNodeBST(…), SuccessorNodeBST(…), PredecessorNodeBST(…) для виконання операцій пошуку даних, створення кореневого вузла, додавання, видалення та пошуку термінальних вузлів у бінарному дереві пошуку.

2. Створити новий проект Lab\_1\_2 та підключити до нього бібліотеку Tree. У функції main() проекту реалізувати меню для виконання операцій, описаних у п.1. Після кожної операції вставки/видалення вузла потрібно викликати функцію ShowTree(…), для відображення змін у бінарному дереві пошуку.

def tree\_successor(root): # Алгоритм пошуку термінального вузла Successor  
 # Термінальний вузол завжди є листком або вузлом з одним нащадком і процедура їх видалення описана вище.  
 if root.right != None:  
 return minimum(root.right)  
 y = root.parent  
 while (y != None and x == y.right):  
 x = y  
 y = y.parent  
 return y.keyА  
  
# Програма Python для демонстрації операції видалення бінарному дереві пошуку  
  
# Бінарне дерево Node  
  
  
class Tree:  
  
 # Конструктор для створення нового Node  
 def \_\_init\_\_(self, key):  
 self.key = key  
 self.left = None  
 self.right = None  
  
  
# Допоміжна функція для обходу BST у порядку  
  
  
def inorder(root): # центрований (inorder) обхід дерева  
 *"""  
 How it work?  
  
 Спочатку відвідайте всі вузли у лівому піддереві  
 Потім кореневий вузол  
 Відвідайте всі вузли у правому піддереві* ***:param*** *root:* ***:return****:  
 """* if root is not None:  
 inorder(root.left)  
 print(root.key, " -> ", end=" ")  
 inorder(root.right)  
  
  
def postorder(root): # зворотний (postorder). обхід дерева  
 *"""  
 How it work?  
  
 Відвідати всі вузли в лівому піддереві  
 Відвідати кореневий вузол  
 Відвідати всі вузли у правому піддереві* ***:param*** *root:* ***:return****:  
 """* if root:  
 postorder(root.left)  
 postorder(root.right)  
 print(str(root.key), " -> ", end='')  
  
  
def preorder(root): # прямий (preorder) обхід дерева  
 *"""  
 How it work?  
  
 Відвідайте кореневий вузол  
 Завітайте до всіх вузлів у лівому піддереві  
 Відвідайте всі вузли у правому піддереві* ***:param*** *root:* ***:return****:  
 """* if root:  
 print(str(root.key), " -> ", end='')  
 postorder(root.left)  
 postorder(root.right)  
  
  
# Допоміжна функція для вставки в новий вузол із заданим ключем у BST  
def insert(node, key):  
  
 # Якщо дерево порожнє, поверніть новий вузол  
 if node is None:  
 return Tree(key)  
  
 # Інакше повторюватися вниз по дереву  
 if key < node.key:  
 node.left = insert(node.left, key)  
 else:  
 node.right = insert(node.right, key)  
  
 # повертає (незмінний) покажчик вузла  
 return node  
  
# Дано непорожній двійковий файл  
# дерево пошуку, повертає вузол  
# з мінімальним значенням ключа знайдене в цьому дереві  
# Зверніть увагу, що все дерево не потрібно обшукувати  
  
  
def minValueNode(node):  
 current = node  
  
 # петля вниз, щоб знайти крайній лівий лист  
 while(current.left is not None):  
 current = current.left  
  
 return current  
  
# За наявності бінарного дерева пошуку та ключа ця функція видаляє ключ і повертає новий корінь  
  
  
def deleteNode(root, key):  
 # Базовий випадок  
 if root is None:  
 return root  
  
 # Якщо ключ, який потрібно видалити, менший за root.key, тоді він знаходиться в лівому піддереві  
  
 if key < root.key:  
 root.left = deleteNode(root.left, key)  
  
 # Якщо kеу, який потрібно видалити, більший за кореневий ключ, тоді він лежить у правому піддереві  
  
 elif(key > root.key):  
 root.right = deleteNode(root.right, key)  
  
 # Якщо ключ такий самий, як ключ root, то це вузол, який потрібно видалити  
 else:  
  
 # Вузол лише з одним дочірнім елементом або без нього  
 if root.left is None:  
 temp = root.right  
 root = None  
 return temp  
  
 elif root.right is None:  
 temp = root.left  
 root = None  
 return temp  
  
 # Вузол з двома дочірніми елементами:  
 # Отримати наступника в порядку (найменший у правому піддереві)  
 temp = minValueNode(root.right)  
  
 # Скопіюйте вміст наступника порядку до цього вузла  
 root.key = temp.key  
  
 # Видалити наступника по порядку  
 root.right = deleteNode(root.right, temp.key)  
  
 return root  
  
  
def minimum(root):  
  
 if root == None:  
 return float('+inf')  
 else:  
 res = root.key  
 lres = minimum(root.left)  
 rres = minimum(root.right)  
 if lres < res:  
 res = lres  
 if rres < res:  
 res = rres  
 return res  
  
  
rooot = None  
  
rooot = insert(rooot, 32) # вставка елемента в бінарне дерево  
rooot = insert(rooot, 3) # вставка елемента в бінарне дерево  
rooot = insert(rooot, 43) # вставка елемента в бінарне дерево  
rooot = insert(rooot, 15) # вставка елемента в бінарне дерево  
rooot = insert(rooot, 2) # вставка елемента в бінарне дерево  
rooot = insert(rooot, 54) # вставка елемента в бінарне дерево  
rooot = insert(rooot, 66) # вставка елемента в бінарне дерево  
  
print("\n-------------------------------------------")  
inorder(rooot)  
print("\n")  
postorder(rooot)  
print("\n")  
preorder(rooot)  
print("\n\n-------------------------------------------")  
  
rooot = deleteNode(rooot, 3)  
rooot = deleteNode(rooot, 54)  
  
print("\n-------------------------------------------")  
inorder(rooot)  
print("\n")  
postorder(rooot)  
print("\n")  
preorder(rooot)  
print("\n-------------------------------------------")  
  
print(f"Мінімальний елемент дерева - {minimum(rooot)}")  
  
print("\n-------------------------------------------")  
inorder(rooot)  
print("\n")  
postorder(rooot)  
print("\n")  
preorder(rooot)  
print("\n\n-------------------------------------------")  
  
print(tree\_successor(rooot))  
rooot = deleteNode(rooot, tree\_successor(rooot))  
  
print("\n-------------------------------------------")  
inorder(rooot)  
print("\n")  
postorder(rooot)  
print("\n")  
preorder(rooot)  
print("\n\n-------------------------------------------")

3. Відкомпілювати проект та продемонструвати його роботу для данихбінарного дерева пошуку, отриманих від викладача.

