НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Лабораторна робота № 5

з дисципліни «Алгоритми та структури даних»
На тему: «Дерева»

Виконав:
студент групи ФІ-12
Завалій Олександр

Київ-2023

Реалізація завдання

Варіант №5

Реалізувати завдання свого варіанту.

Побудувати двійкове дерево пошуку з букв рядка, що вводиться. Вивести його на екран у вигляді дерева. Знайти букви, що зустрічаються більше одного разу. Видалити з дерева ці літери. Вивести елементи дерева, що залишилися, при його постфіксному обході.

```
from random import choice
from string import ascii_lowercase


class Node:
    def __init__(self, key):
        self.left = self.right = None
        self.key = key


class Tree:
    def __init__(self):
        self.root = None
    # Add new item
    def insert(self, key):
        for i in key:
            if self.root is None:
                self.root = Node(i)
            else:
                self.__insert(self.root, i)
    def __insert(self, node, key):
        if key < node.key:
            if node.left is None:
                node.left = Node(key)
            else:
                self.__insert(node.left, key)
        else:
            if node.right is None:
                node.right = Node(key)
            else:
                self.__insert(node.right, key)
    # Show root
    def root_item(self):
        print(self.root.key)
    # Show tree
    def __show_tree(self, node):
        if node is None:
            print("Binary tree is empty")
            return
        v = [('r:', node)]
        while v:
            vn = []
            for x in v:
                print(x[0], x[1].key, end='   ')
                if x[1].left:
                    vn += [(f"L,{x[1].key}:", x[1].left)]
                if x[1].right:
                    vn += [(f"R,{x[1].key}:", x[1].right)]
            v = vn
            print()
```

```
    def __postorder_dfs(self, node):
        if node is None:
            print("Binary tree is empty")
            return
        v = [node]
        result = []
        while v:
            vn = []
            for x in v:
                result += x.key
                if x.left:
                    vn += [x.left]
                if x.right:
                    vn += [x.right]
            result += '\n'
            v = vn
        return result[::-1]
    def __nlr(self, node):
        if node is None:
            return
        self.__nlr(node.left)
        self.__nlr(node.right)
        print(node.key, end=' -> ')
    def display(self, method):
        match method:
            case 1:
                self.__show_tree(self.root)
                print()
            case 2:
                data = self.__postorder_dfs(self.root)
                if data is None:
                    print("Binary tree is empty")
                    return
                for i in data[:-1]:
                    print(i, end=' -> ')
                print(data[-1], end='\n\n')
            case 3:
                self.__nlr(self.root)
                print()
    # Del
    def delete(self, key):
        if self.root is None:
            return None
        else:
            self.root = self.__delete(self.root, key)
```

```
    def __delete(self, node, key):
        if node is None:
            return node

        if key < node.key:
            node.left = self.__delete(node.left, key)
        elif key > node.key:
            node.right = self.__delete(node.right, key)
        else:
            if node.left is None and node.right is None:
                node = None
            elif node.left is None:
                node = node.right
            elif node.right is None:
                node = node.left
            else:
                min_right = self.__find_min(node.right)
                node.key = min_right.key
                node.right = self.__delete(node.right, min_right.key)
        return node
    def __find_min(self, node):
        if node.left is None:
            return node
        else:
            return self.__find_min(node.left)
    # Find the same letters
    def same_letters(self, node):
        if node is None:
            print("Binary tree is empty")
            return
        letters = {}
        for i in set(node):
            if node.count(i) > 1:
                letters[i] = node.count(i) - 1
        print(letters, end='\n\n')
        for key, value in letters.items():
            for _ in range(value):
                self.delete(key)


def ascii_low():
    print()
    for i in ascii_lowercase:
        print(i, end=' ')
    print()
```

```
if __name__ == "__main__":
    node1 = Tree()

    ascii_low()
    #data = [choice(ascii_lowercase) for _ in range(10)]
    data = ['y', 't', 'l', 'v', 'q', 'h', 'w', 'u', 'i', 'm', 'q', 'v', 'v', 'r
    ', 'o']
    #data = input().split()
    print(*data, end='\n\n')

    node1.insert(data)

    print("Binary tree:")
    node1.display(1)
    print("Postfix")
    node1.display(2)

    node1.delete('l')

    node1.same_letters(data)

    print("Binary tree after deleting:")
    node1.display(1)
    print("Postfix")
    node1.display(2)
```