МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра Інформаційної Безпеки

Практикум з Алгоритмів та структур даних

Лабораторна робота №5
Дерева

Мета роботи:

отримати навики застосування двійкових дерев,
реалізувати основні операції над деревами:
обхід дерев, включення, виключення та пошук вузлів.

Виконав:
студент II курсу
групи ФБ-01
Сахній Н.Р.

Київ 2022

Виконання лабораторної роботи

На максимальний бал – реалізувати завдання свого варіанту.

Варіант №2

Побудувати двійкове дерево пошуку, в вершинах якого знаходяться слова з текстового файлу. Вивести його на екран у вигляді дерева. Визначити кількість вершин дерева, що містять слова, які починаються на зазначену букву. Видалити з дерева ці вершини.

/*

Як можна буде помітити у демонстрації виконання програми було реалізовано функціонал завдання за варіантом, а от у прикладених нижче кодах – основні операції над деревами: обхід дерев, включення, виключення та пошук вузлів.

До того ж при виводі бінарного дерева на екран, ми будемо бачити дерево, реалізоване за допомогою псевдографіки, а також додатково буде вивід центрованого (симетричного) обходу.

Як відомо із теорії, бінарне дерево пошуку має наступну обов'язкову властивість, яка також буде зберігатися і у моєму дереві:

Нехай `x` – довільна вершина двійкового дерева пошуку.

- Якщо вершина y знаходиться в лівому піддереві вершини `x`, то `key[y] <= key[x]`.
- Якщо `y` знаходиться в правому піддереві `x`, то `key[y] >= key[x]`.

Можливі помилки такі, як: неправильний тип або формат текстового файлу, файл або шлях до файлу не існує, символ не є буквою – були враховані в реалізації програмного коду для коректного виконання даного завдання

*/

Виконання програми за варіантом:

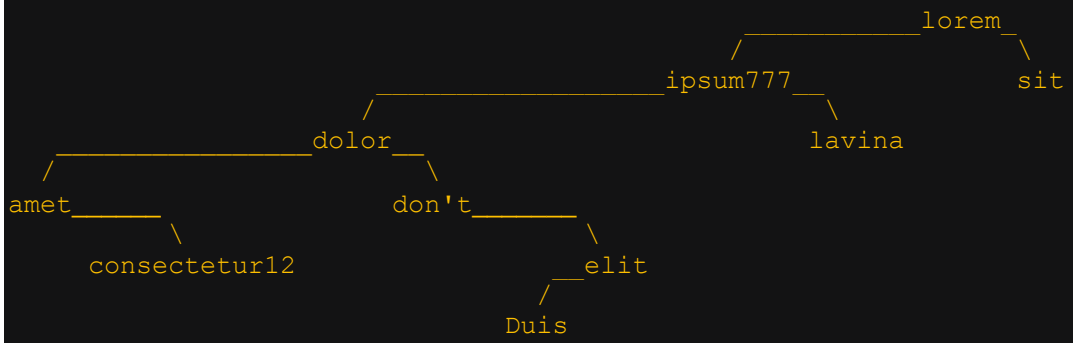
```
"D:\KPI\АСД\ASD_Sakhnii Nazar FB-01\venv\Scripts\python.exe" "D:/KPI/АСД/ASD_Sakhnii Nazar FB-01/using_BST.py"


◘ Напишіть абсолютний або відносний шлях до файлу наступного типу та формату:
    ▼ Файл повинен бути текстовим, тобто мати розширення '.txt'
    >>> D:\KPI\АСД\ASD_Sakhnii Nazar FB-01\Звіти про виконану роботу\ASD_Lab5.docx
Йой, файлове Розширення не відповідає заданому формату!
```

Якщо файл буде не правильного формату розширення, то отримаємо відповідне попередження.

◘ Напишіть абсолютний або відносний шлях до файлу наступного типу та формату:
    ▼ Файл повинен бути текстовим, тобто мати розширення '.txt'
    >>> ./word_text.txt
Ойва, схоже, що було не правильно введено Назву файлу або Шлях до нього!

Якщо шлях до файлу або назва файлу будуть помилковими, то отримаємо відповідне буде не правильного формату розширення, то отримаємо відповідне попередження.

◘ Напишіть абсолютний або відносний шлях до файлу наступного типу та формату:
    ▼ Файл повинен бути текстовим, тобто мати розширення '.txt'
    >>> ./Symbolic Text.txt
Список слів із файлу: ['Lorem', 'ipsum777', 'dolor', 'sit', 'amet', 'consectetur12', "don't", 'elit', 'duis', 'lavina']



☼ Центрований (симетричний) обхід бінарного дерева пошуку:
{ amet consectetur12 dolor don't duis elit ipsum777 lavina lorem sit }, де [lorem] - це корінь даного дерева.

◘ Щоб визначити Кількість вершин дерева, що містять слова, які починаються на зазначену букву, а опісля Видалити з дерева ці вершини;
    ▼ Потрібно в не залежності від регістру ввести символ, який є буквою
    >>> d
Кількість таких слів, які починаються із букви 'd' -> 3

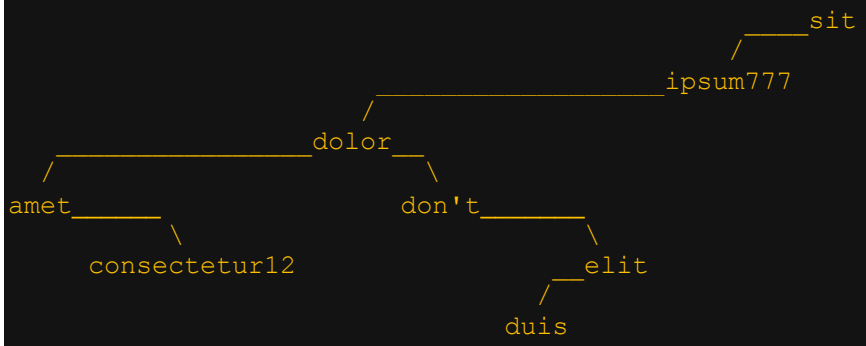
Як можна помітити із дерева були видалені усі (три) слова, які починалися на літеру ‘d’.
При цьому основна властивість бінарного дерева пошуку зберігається.

```
☼ Центрований (симетричний) обхід бінарного дерева пошуку:
{  amet  consectetur12  don't  elit  ipsum777  lavina  lorem  sit  }, де [lorem] - це корінь даного дерева.


Process finished with exit code 0
```

```
◘ Щоб визначити Кількість вершин дерева, що містять слова, які починаються на зазначену букву, а опісля Видалити з дерева ці вершини;
        ▼ Потрібно в не залежності від регістру ввести символ, який є буквою
        >>> $
! Символ, який було введено, не є Буквою !
```

Якщо символ, із якого мають починатися слова, які повинні бути видалені із дерева, не буде літерою (буквою), то отримаємо відповідне попередження

```
◘ Щоб визначити Кількість вершин дерева, що містять слова, які починаються на зазначену букву, а опісля Видалити з дерева ці вершини;
        ▼ Потрібно в не залежності від регістру ввести символ, який є буквою
        >>> l
Кількість таких слів, які починаються із букви 'l' -> 2
```



Як можна помітити із дерева були видалені усі (два) слова, у тому числі корінь дерева, які починалися на літеру ‘l’. При цьому основна властивість бінарного дерева пошуку зберігається і коренем стає нова вершина, а саме ‘sit’.

```
☼ Центрований (симетричний) обхід бінарного дерева пошуку:
{  amet  consectetur12  dolor  don't  duis  elit  ipsum777  sit  }, де [sit] - це корінь даного дерева.


Process finished with exit code 0
```

```
------------------------------------------------------------------------------------------------------------------------

◘ Щоб визначити Кількість вершин дерева, що містять слова, які починаються на зазначену букву, а опісля Видалити з дерева ці
вершини;
        ▼ Потрібно в не залежності від регістру ввести символ, який є буквою
        >>> x
Кількість таких слів, які починаються із букви 'x' -> 0

                                                    ___________lorem_
                                                   /                 \
                                               ipsum777__             sit
                       __________________/          \
                      /                              lavina
     _____________dolor___
    /                     \
amet______             don't______
          \                       \
     consectetur12              __elit
                               /
                            duis

☼ Центрований (симетричний) обхід бінарного дерева пошуку:
{  amet  consectetur12  dolor  don't  duis  elit  ipsum777  lavina  lorem  sit  }, де [lorem] - це корінь даного дерева.


Process finished with exit code 0
```

> Як можна помітити із дерева не було видалено жодного слова, так як не має такої вершини, яка починалася б на літеру ‘x’. При цьому основна властивість бінарного дерева пошуку зберігається.

/* ↓ Текстовий файл, який використовується у даній програмі ↓ */

```
*Символьний текст.txt: Блокнот
Файл  Редагування  Формат  Вигляд  Довідка
{Lorem}. ipsum777 dolor! sit: amet, consectetur12 don't "elit" duis lavina!
Рд 1, ствп 76    100%    Windows (CRLF)    UTF-8
```

↑ (Як можна поміти у ньому також присутні різноманітні символи пунктуації, а також цифри та пропуски) ↑

Програмні коди:

- Файл **`BinarySearchTree.py`** містить реалізацію бінарного дерева пошуку + вивід його за допомогою інфографіки

(Фрагменти коду для реалізації методів класу BinarySearchTree: Центрованого (симетричного) обходу бнарного дерева, Пошуку вершини у двійковому дереві, Максимум, Мінімум, Наступний та Попередній елемент до заданого, Додавання та Видалення вершини із бінарного дерева пошуку – були взяті із презентації лекції з теми "Двійкові дерева пошуку"):

```
from binarytree import Node


class DrawBinaryTree:
    # Клас, який буде додавати вершини до дерева, яке буде виведено за допомогою псевдографіки
    root = None

    def Add_For_Drawing(self, vertex):
        if self.root is not None:
            temp = self.root
            while True:
                if vertex > temp.value:
                    if temp.right is not None:
                        temp = temp.right
                    else:
                        temp.right = Node(vertex)
                        break
                else:
                    if temp.left is not None:
                        temp = temp.left
                    else:
                        temp.left = Node(vertex)
                        break
        else:
            self.root = Node(vertex)


class Vertex:
    # Допоміжний клас - вершина бінарного дерева пошуку """

    def __init__(self, item):
        """ Конструктор вершини

        :param item: Елемент бінарного дерева
```

```
        """
        self.key = item
        self.left = None
        self.right = None
        self.parent = None


class BinarySearchTree:
    # Бінарне дерево пошуку

    def __init__(self):
        """ Конструктор бінарного дерева пошуку - створює порожнє дерево """
        self.root = None
        self.vertex_set = []

    def Inorder_Tree_Walk(self, vertex) -> None:
        if vertex is not None:
            self.Inorder_Tree_Walk(vertex.left)
            print(vertex.key, end="  ")
            self.Inorder_Tree_Walk(vertex.right)
        return None

    def Add_For_Drawing(self, vertex):
        if self.root is not None:
            temp = self.root
            while True:
                if vertex > temp.value:
                    temp = temp.right
                    if temp.right is not None:
                        temp = temp.right
                    else:
                        temp.right = Node(vertex)
                        break
                else:
                    if temp.left is not None:
                        temp = temp.left
                    else:
                        temp.left = Node(vertex)
                        break
        else:
            self.root = Node(vertex)

    def Tree_Search(self, vertex, key) -> "Vertex object":
        while (vertex is not None) and (key != vertex.key):
            if key < vertex.key:
                vertex = vertex.left
```

```
            else:
                vertex = vertex.right
        return vertex

    def Tree_Minimum(self, vertex) -> "Vertex object":
        while vertex.left is not None:
            vertex = vertex.left
        return vertex

    def Tree_Maximum(self, vertex) -> "Vertex object":
        while vertex.right is not None:
            vertex = vertex.right
        return vertex

    def Tree_Successor(self, x_vertex) -> "Vertex object":
        if x_vertex.right is not None:
            return self.Tree_Minimum(x_vertex.right)
        y_vertex = x_vertex.parent
        while (y_vertex is not None) and (x_vertex == y_vertex.right):
            x_vertex = y_vertex
            y_vertex = y_vertex.parent
        return y_vertex

    def Tree_Predecessor(self, x_vertex) -> "Vertex object":
        if x_vertex.left is not None:
            return self.Tree_Maximum(x_vertex.left)
        y_vertex = x_vertex.parent
        while (y_vertex is not None) and (x_vertex == y_vertex.left):
            x_vertex = y_vertex
            y_vertex = y_vertex.parent
        return y_vertex

    def Tree_Insert(self, new_item) -> None:
        y_vertex = None
        x_vertex = self.root
        new_vertex = Vertex(new_item)
        while x_vertex is not None:
            y_vertex = x_vertex
            if new_vertex.key < x_vertex.key:
                x_vertex = x_vertex.left
            else:
                x_vertex = x_vertex.right
        new_vertex.parent = y_vertex
        if y_vertex is None:
            self.root = new_vertex
```

```
        elif new_vertex.key < y_vertex.key:
            y_vertex.left = new_vertex
        else:
            y_vertex.right = new_vertex
        return None

    def Tree_Delete(self, del_vertex) -> "Vertex object":
        if (del_vertex.left is None) or (del_vertex.right is None):
            y_vertex = del_vertex
        else:
            y_vertex = self.Tree_Successor(del_vertex)

        if y_vertex.left is not None:
            x_vertex = y_vertex.left
        else:
            x_vertex = y_vertex.right

        if x_vertex is not None:
            x_vertex.parent = y_vertex.parent

        if y_vertex.parent is None:
            self.root = x_vertex
        elif y_vertex == y_vertex.parent.left:
            y_vertex.parent.left = x_vertex
        else:
            y_vertex.parent.right = x_vertex

        if y_vertex != del_vertex:
            del_vertex.key = y_vertex.key
        return y_vertex
```

- Файл **`using_BST.py`** містить реалізацію власного завдання згідно варіанту із використанням бінарного дерева пошуку:

```
from BinarySearchTree import *
import re
import string


def count_and_delete(our_tree, vertex, symbol) -> tuple:
    global counter, draw_again
    if vertex is not None:
        if vertex.key.startswith(symbol.lower()):
```

```
            counter += 1
            del_vertex = our_tree.Tree_Delete(vertex)
            if del_vertex.key == our_tree.root.key:
                draw_again.Add_For_Drawing(del_vertex.key)
        else:
            draw_again.Add_For_Drawing(vertex.key)
        count_and_delete(our_tree, vertex.left, symbol)
        count_and_delete(our_tree, vertex.right, symbol)

    return counter, draw_again


while True:
    filepath = input("\n◘ Напишіть абсолютний або відносний шлях до файлу наступного типу та формату: \n\
    ▼ Файл повинен бути текстовим, тобто мати розширення '.txt'\n\
    >>> ")
    try:
        with open(f"{filepath}", "r") as file:
            if filepath.endswith(".txt"):
                text = file.read()
                word_tree = BinarySearchTree()
                draw_tree = DrawBinaryTree()
                print("Список слів із файлу:", re.findall(fr"[\w']+|[{string.ascii_letters}]", text))
                for word in re.findall(fr"[\w']+|[{string.ascii_letters}]", text):
                    word_tree.Tree_Insert(str(word).lower())
                    draw_tree.Add_For_Drawing(str(word).lower())
                print(draw_tree.root)
                print("☼ Центрований (симетричний) обхід бінарного дерева пошуку: \n{", end=" ")
                word_tree.Inorder_Tree_Walk(word_tree.root)
                print("}, де [", word_tree.root.key, "] - це корінь даного дерева.\n", sep="")

                counter = 0
                draw_again = DrawBinaryTree()
                while True:
                    letter = input("\n◘ Щоб визначити Кількість вершин дерева, що містять слова, які починаються "
                                   "на зазначену букву, а опісля Видалити з дерева ці вершини; \n"
                                   "\t▼ Потрібно в не залежності від регістру ввести символ, який є буквою \n"
                                   "\t>>> ")
                    if letter.isalpha():
                        counter, new_tree = count_and_delete(word_tree, word_tree.root, letter)
                        print(f"Кількість таких слів, які починаються із букви '{letter}' -> {counter}")
                        print(new_tree.root)
                        print("☼ Центрований (симетричний) обхід бінарного дерева пошуку: \n{ ", end=" ")
                        word_tree.Inorder_Tree_Walk(word_tree.root)
                        print("}, де [", word_tree.root.key, "] - це корінь даного дерева. \n ", sep="")
                        break
```

```
                    else:
                        print("! Символ, який було введено, не є Буквою ! \n")
                break
            else:
                print("Йой, файлове Розширення не відповідає заданому формату! \n")
    except FileNotFoundError:
        print("Ойва, схоже, що було не правильно введено Назву файлу або Шлях до нього! \n")
```