МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

факультет програмної інженерії та бізнесу

кафедра інженерії програмного забезпечення

**Лабораторна робота № 4**

з дисципліни « Алгоритми і структури даних »

*назва дисципліни*

на тему:«ДЕРЕВА. ФОРМИ ПРЕДСТАВЛЕННЯ ДЕРЕВ. ОБХОДИ ДЕРЕВ»

Виконав: студент 2 курсу групи № 622п

освітньої програми

121 інженерія програмного забезпечення

(шифр і назва ОП)

Зайченко Ярослав Ігорович

(прізвище й ініціали студента)

Прийняв: канд. техн. наук, доцент на кафедрі 603 Волобуєва Ліна Олексіївна

(посада, науковий ступінь, прізвище й ініціали)

Кількість балів:

Харків – 2023

**Мета роботи** − вивчення різних форм представлення структури даних «дерево» та основних операцій роботи з цим типом даних.

Постановка задачі

1. Описати вказаний абстрактний тип даних дерево і основні функції роботи з ним на абстрактному рівні і логічному рівні:

* початкова ініціалізація дерева;
* перевірка дерева на пустоту (дерево порожнє);
* додавання вузла дерева (нового нащадка);
* видалення вузла дерева;
* пошук вузла в дереві.
* виведення дерева на екран ( згідно з варіантом).

1. Створити додаток, який демонструє роботу основних функцій, перелічених вище (див. п.1.).

Зміст звіту

Зміст звіту з лабораторної роботи №4 має включати наступні підрозділи:

1. Опис m-арного дерева згідно з завданням варіанту.
2. Перетворення m-арного дерева з п.1 у бінарне.
3. Абстрактне представлення структури.
4. Представлення даних в графічному вигляді.
5. Проектування необхідних класів і структур.
6. Алгоритм для кожного методу (структури), використовуючи один з наступних видів представлення алгоритмів: псевдокод або блок- схему.
7. Лістинг програми та екранні форми результату роботи програми.
8. Висновки

**Варіант 9** індивідуального завдання до лабораторної роботи 4:

Описати структуру інституту або іншого підприємства (начальник - підлеглий) у вигляді m-арного дерева.

Фізичне зберігання дерева - динамічне

Тип обходу дерева - зворотний

Алгоритм реалізації обходу дерева - ітеративний

Алгоритм реалізації пошуку вузла - рекурсивний

хід роботи

Опис програми:

Мова програмування: С#, операційна система Windows 11 Prо, Версія 23H2, Збірка ОС 22621.1325, процесор: Apple Silicon M1 Pro 3.20 GHz (ядер: 4), компілятор: Microsoft Visual Studio Community 2022 (64-розрядна версія ARM).

Опис m-арного дерева згідно з завданням варіанту.

До завдання варіанта 9 було створену наступну ієрархію в організації

* Керівник
* Підрозділ 1
* Відділ виробництва
* Відділ якості
* Відділ безпеки
* Підрозділ 2
* Відділ маркетингу
* Відділ продажів
* Відділ реклами
* Підрозділ 3
* Відділ фінансів
* Відділ бухгалтерії
* Відділ інвестицій

Дерево зображене на рисунку 1.

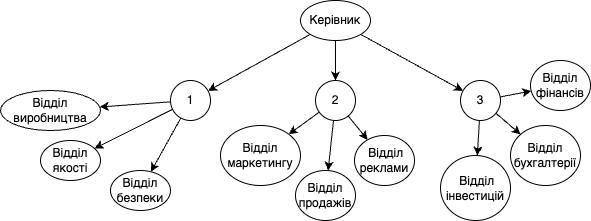


Рисунок 1 – Графічне подання m-арного дерева

Перетворення m-арного дерева з у бінарне.

За наступним алгоритмом виконаємо перетворення дерев:

1. В кожному вузлу залишити тільки гілку до старшого сина;
2. З’єднати горизонтальними ребрами всіх братів одного батька;
3. Таким чином перебудувати дерево за правилом: лівий син – вершина, розташована під даною; правий син – вершина, розташована праворуч від даної (тобто на одному ярусі з нею).
4. Розвернути дерево на 45 градусів так, щоб усі вертикальні гілки відображали лівих синів, а горизонтальні – правих

Результат перетворення зображено на рисунку 2 та 3.

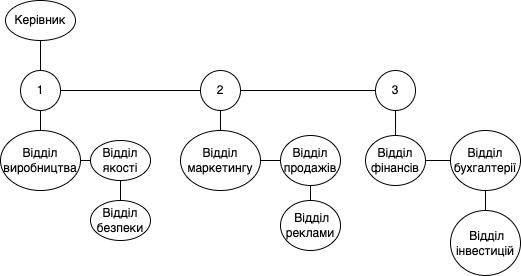


Рисунок 2 – Бінарне подання m-арного дерева за алгоритмом перетворення

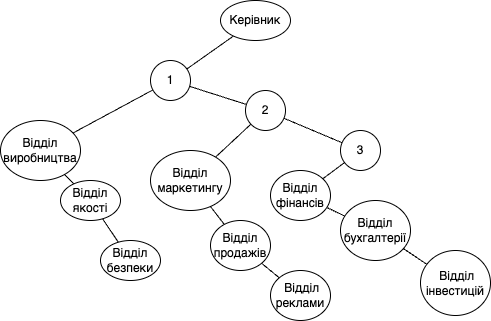


Рисунок 3 – Бінарне подання m-арного дерева за алгоритмом перетворення

Абстрактне представлення структури

Стек (Stack) – АТД з доступом до елементів за правилом «останній прийшов – першим вийшов» (Last In – First Out, LIFO)

**Опис АТД Бінарне дерево:**

BT – binary tree, El\_inf – значення елементу дерева, Side – сторона піддеревабб Father – батьківський елемент

**Операції:**

* 1. Створити (New):  → BT;
  2. Перевірка на порожність (IsEmpty): BT → bool;
  3. Додавання елементу вліво/вправо (Add): El\_inf x Side x Father → ElTree;
  4. Пошук вузла в дереві (Find): BT x El\_inf x Father → BT;
  5. Видалення елементу (Remove): BT x El\_inf → BT;
  6. Обхід дерева (ToString): BT → String;

Проектування необхідних класів і структур.

Клас ElTree представляє собою вузол/елемент дерева. Основні поля класу та їхні призначення:

* ElTree L, R – Покажчики на ліве (L) та праве (R) піддерева;
* string inf – містить інформацію, яка визначає значення вузла;

public class ElTree

{

public ElTree L, R; //покажчик на піддерева

public string inf;

public ElTree(string inf)

}

Клас Tree використовується для створення та управління деревом. Основні поля та методи:

* public ElTree T – Вказівник на кореневий вузол дерева.
* Конструктор public Tree(string inf) – Ініціалізує об'єкт Tree, створюючи кореневий вузол з вказаною інформацією.
* Метод public ElTree Find(ElTree V, string inf) – Пошук вузла з інформацією у піддереві, починаючи з вказаного вузла V.
* Метод public ElTree Find\_Father(ElTree V, string inf) – Пошук предка вузла з інформацією у піддереві, починаючи з вказаного вузла V.
* Метод public bool Add(ElTree V, string inf, char S) – Додавання вузла з інформацією вліво (S = 'L') або вправо (S = 'R') вказаного вузла V.
* Метод public bool Remove(ref ElTree V, string inf) – Видалення вузла з інформацією з дерева, починаючи з кореня V.
* Метод public void TreeInString(ElTree V, ref string s) –Перетворення дерева у рядок (стрічку) у зворотньому порядку.
* Метод public bool TreeExists() – Перевірка, чи існує дерево (чи кореневий вузол не є порожнім).

public class Tree

{

public ElTree T;

public Tree(string inf)

public ElTree Find(ElTree V, string inf) //пошук вузла з інформацією

public ElTree Find\_Father(ElTree V, string inf)//пошук предка вузла з інформацією

public bool Add(ElTree V, string inf, char S)//додавання вузла з інформацією вліво/вправо s=L/R.

public bool Remove(ref ElTree V, string inf)//видалення вузла з інформацією

public void TreeInString(ElTree V, ref string s)

public bool TreeExists()

}

Методи подані у вигляді псевдокоду

Метод перевірки введеного значення з клавіатури за діапазоном:

Функція Ввід\_Числа(текст, нижня\_межа, верхня\_межа) повертає ціле число

Поки істина

Якщо Спроба\_Перетворення\_Рядка\_У\_Число(Console.ReadLine(), input), а також input <= верхня\_межа і input >= нижня\_межа То

Повернути input

Інакше

Вивести "Помилка введення числа. Спробуйте ще раз"

Find: Пошук вузла з інформацією у дереві.

ElTree Find(ElTree V, string inf):

якщо V не є null:

якщо V.inf дорівнює inf, то повернути V

temp = Find(V.L, inf)

якщо temp не є null, то повернути temp

повернути Find(V.R, inf)

повернути null

**Find\_Father**: Пошук предка вузла з інформацією у дереві.

ElTree Find\_Father(ElTree V, string inf):

якщо V не є null:

якщо V.L не є null і V.L.inf дорівнює inf, то повернути V

якщо V.R не є null і V.R.inf дорівнює inf, то повернути V

temp = Find\_Father(V.L, inf)

якщо temp не є null, то повернути temp

повернути Find\_Father(V.R, inf)

повернути null

Add: Додавання вузла з інформацією вліво або вправо.

bool Add(ElTree V, string inf, char S):

temp = new ElTree(inf)

якщо S дорівнює 'L':

якщо V.L є null, то V.L = temp і повернути true

інакше повернути false

інакше (S дорівнює 'R'):

якщо V.R є null, то V.R = temp і повернути true

інакше повернути false

Remove: Видалення вузла з інформацією з дерева.

bool Remove(ref ElTree V, string inf):

father = Find\_Father(V, inf)

temp = Find(V, inf)

якщо temp є null, то повернути false

якщо father є null:

V = null

інакше, якщо father.L не є null і father.L.inf дорівнює inf:

father.L = null

інакше:

father.R = null

повернути true

**TreeInString**: Перетворення дерева у рядок у зворотньому порядку.

void TreeInString(ElTree V, ref string s):

якщо V є null, то повернути

stack = новий Stack<ElTree>()

stack.Push(V)

outputStack = новий Stack<string>()

while stack не є порожнім:

current = stack.Pop()

outputStack.Push(current.inf)

якщо current.L не є null, то stack.Push(current.L)

якщо current.R не є null, то stack.Push(current.R)

while outputStack не є порожнім:

s += outputStack.Pop() + " "

TreeExists: Перевірка наявності дерева.

bool TreeExists():

повернути T не є null

Тестування програми

Перевіримо існування дерева перед його створенням. Результат на рисунку 4.

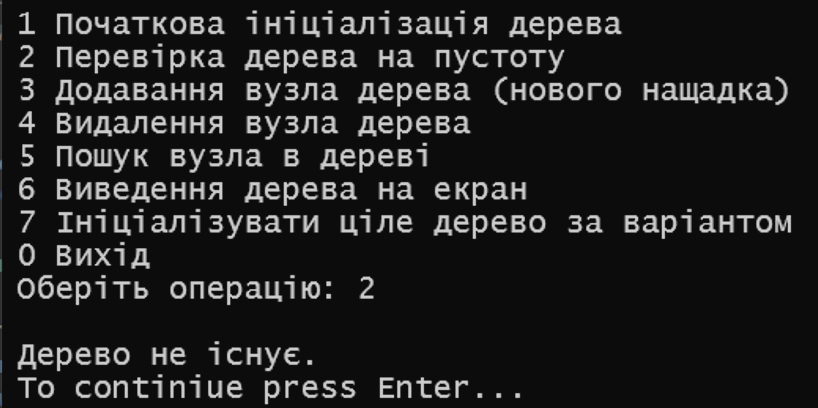


Рисунок 4 – перевірка пустоти дерева

Після перевірки можна ініціалізувати дерево з коренем Керіник. Результат на рисунку 5

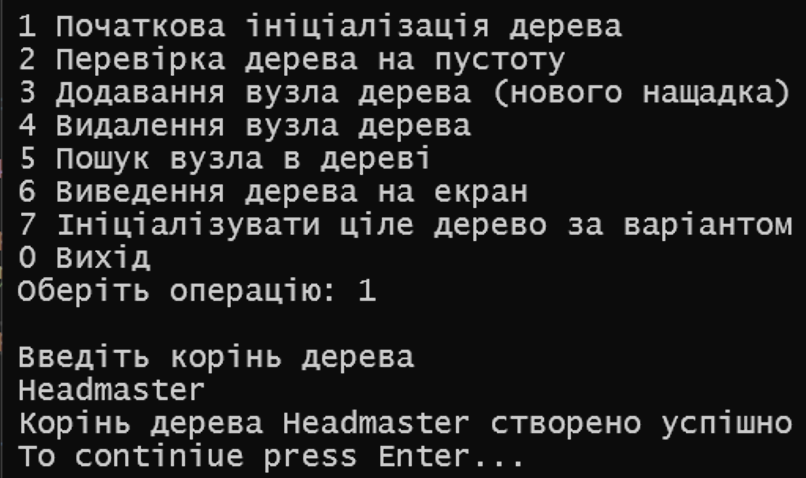


Рисунок 5 – створення кореня дерева

Після створення кореня дерева, можна додати нові вузли дерева.

Приклад процесу додавання нових вузлів в дерево зображено на рисунку 6 та повністю створене дерево на риску 7.

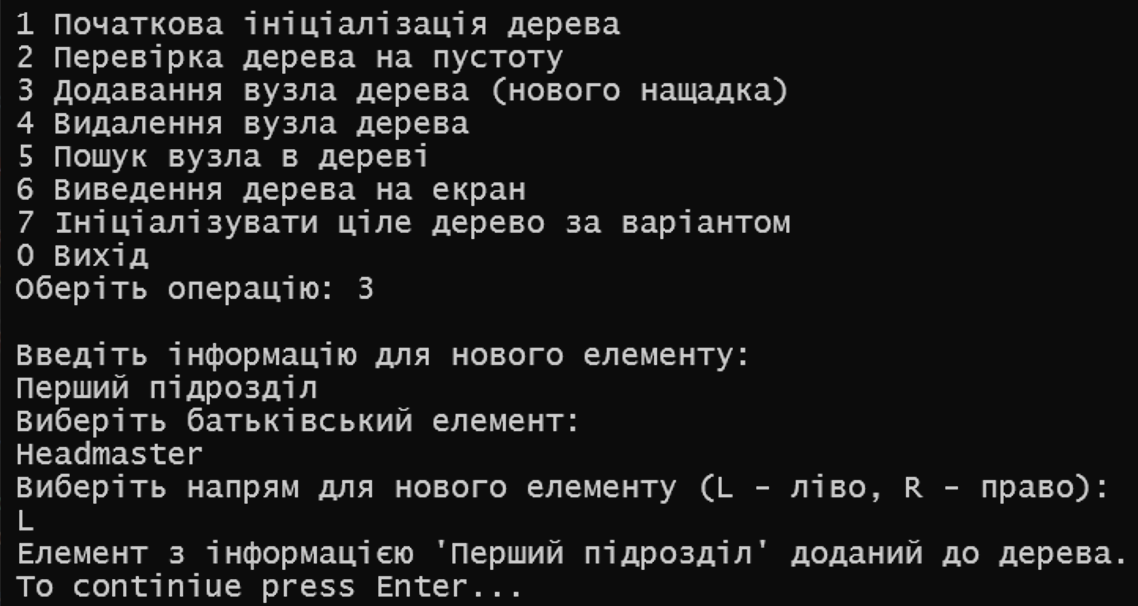


Рисунок 6 – додавання нащадків до дерева

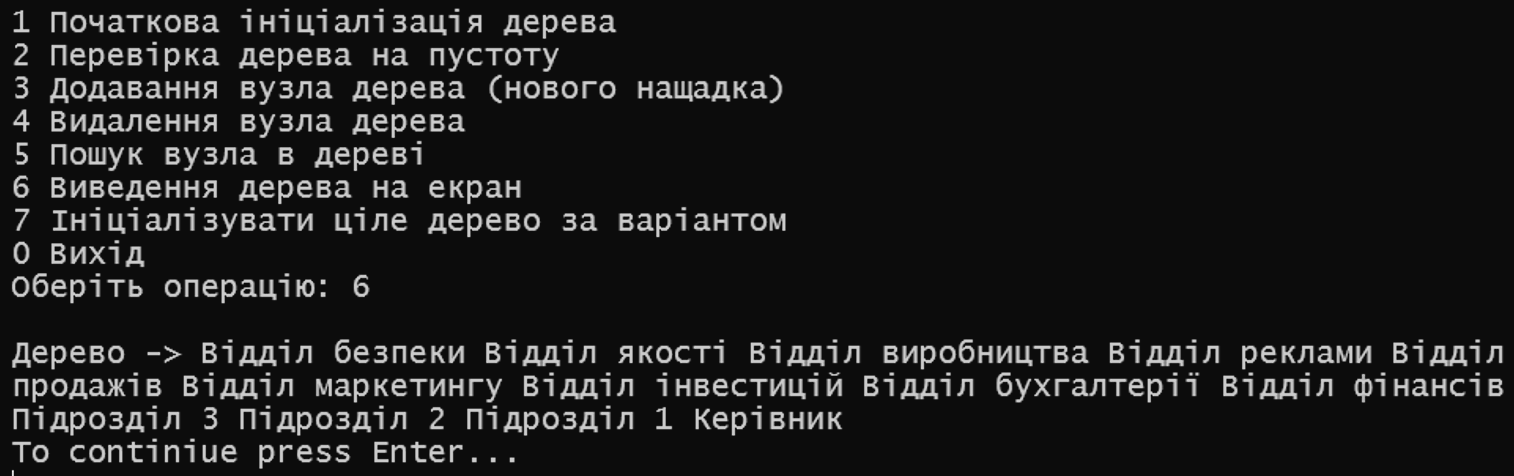


Рисунок 7 – виведення структури дерева на екран

Перевіримо існування дерева після його повного створення. Результат на рисунку 8.

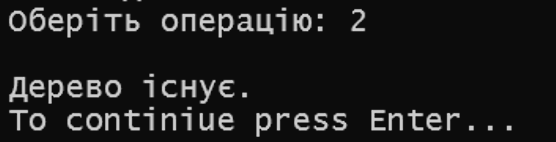


Рисунок 8 – перевірка існування дерева після його створення

Виконаємо пошук елементу з інформацією в комірці «Відділ маркетингу». Результат успішного пошуку зображено на рисунку 9.

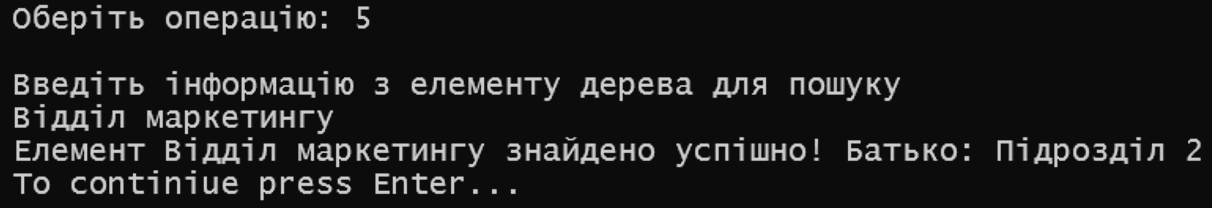


Рисунок 9 – пошук існуючого елементу

Виконаємо пошук елементу якого не існує в дереві інформацією в комірці «Пошук». Результат пошуку зображено на рисунку 10.

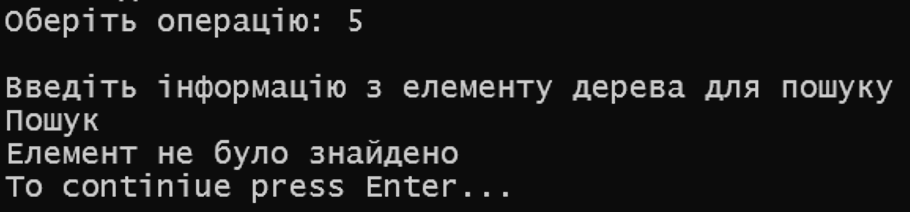
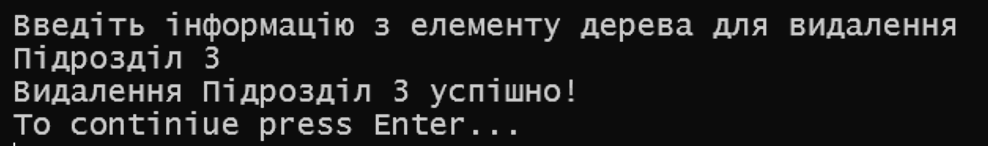


Рисунок 10 – пошук неіснуючого елементу

Видалимо елемент Підрозділ 3 з дерева. Результат на рисунку 11.



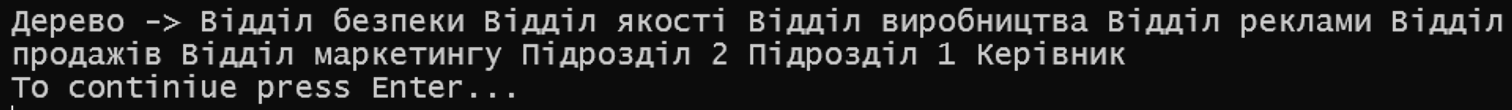


Рисунок 10 – результат видалення вузла

ВИСНОВОК

У ході виконання лабораторної роботи я:

* Детально описав абстрактний тип даних - дерево та визначив основні функції для його роботи на абстрактному та логічному рівнях.

Реалізація цих функцій включає в себе:

* Початкову ініціалізацію дерева, яка передбачає створення кореневого вузла.
* Перевірку на пустоту дерева шляхом перевірки наявності кореневого вузла.
* Додавання нового вузла (нащадка) в дерево, що може включати в себе перевірку унікальності ключа та встановлення відповідних зв'язків між вузлами.
* Видалення вузла з дерева, яке може включати в себе різні випадки - видалення листа, вузла з одним нащадком або вузла з двома нащадками.
* Пошук вузла в дереві, який може виконуватися за його ключем.
* Виведення структури дерева на екран у вигляді, вказаному у варіанті завдання.

Для реалізації цих функцій використовуються базові операції роботи з деревом, такі як додавання, видалення, пошук та перевірка на порожність. Основні функції виконуються на абстрактному рівні, що дозволяє їх використання у різних конкретних реалізаціях дерев.

* Додаток, який я створив, демонструє роботу основних функцій з використанням дерева. Програма включає в себе інтерактивний інтерфейс для введення даних та виклику функцій, таких як додавання, видалення, пошук та виведення дерева на екран. Це дозволяє ефективно перевірити коректність роботи реалізованих функцій.
* Написав звіт

ДОДАТОК А

Машинний лістинг програми

Program.cs:

using System.Linq.Expressions;

using System.Text;

namespace ASD\_Lab\_4

{

internal class Program

{

static void Main(string[] args)

{

// Устанавливаем кодировку консоли на UTF-8

Console.InputEncoding = Encoding.UTF8;

Console.OutputEncoding = Encoding.UTF8;

Tree myTree = null;

string Menu = "1 Початкова ініціалізація дерева\n" +

"2 Перевірка дерева на пустоту\n" +

"3 Додавання вузла дерева (нового нащадка)\n" +

"4 Видалення вузла дерева\n" +

"5 Пошук вузла в дереві\n" +

"6 Виведення дерева на екран\n" +

"7 Ініціалізувати ціле дерево за варіантом\n" +

"0 Вихід";

while (true)

{

try

{

Console.WriteLine(Menu);

switch (Input\_range("Оберіть операцію: ", 7, 0))

{

case 0: Environment.Exit(0); break;

case 1:

Console.WriteLine("Введіть корінь дерева");

string korin = Console.ReadLine();

myTree = new Tree(korin);

Console.WriteLine($"Корінь дерева {myTree.T.inf} створено успішно"); PressEnter();

break;

case 2:

if (myTree == null) { Console.WriteLine("Дерево не існує."); PressEnter(); break; }

if (myTree.TreeExists()) Console.WriteLine("Дерево існує.");

else Console.WriteLine("Дерево не існує."); PressEnter(); break;

case 3:

Console.WriteLine("Введіть інформацію для нового елементу:");

string info = Console.ReadLine();

Console.WriteLine("Виберіть батьківський елемент:");

string parentInfo = Console.ReadLine();

Console.WriteLine("Виберіть напрям для нового елементу (L - ліво, R - право):");

char direction;

if (!char.TryParse(Console.ReadLine().ToUpper(), out direction) || (direction != 'L' && direction != 'R'))

Console.WriteLine("Введіть коректне значення для напряму (L або R).");

ElTree parentNode = myTree.Find(myTree.T, parentInfo);

if (parentNode == null)

Console.WriteLine($"Батьківський елемент з інформацією '{parentInfo}' не знайдений.");

if (myTree.Add(parentNode, info, direction))

Console.WriteLine($"Елемент з інформацією '{info}' доданий до дерева.");

else

Console.WriteLine("Додавання не вдалося. Можливо, елемент вже існує або напрям вже зайнятий.");

PressEnter(); break;

case 4:

Console.WriteLine("Введіть інформацію з елементу дерева для видалення");

string del\_el = Console.ReadLine();

if (myTree.Remove(ref myTree.T, del\_el))

Console.WriteLine($"Видалення {del\_el} успішно!");

else Console.WriteLine("Елемент не був видалений!");

PressEnter(); break;

case 5:

Console.WriteLine("Введіть інформацію з елементу дерева для пошуку");

string find\_el = Console.ReadLine();

ElTree el = myTree.Find(myTree.T, find\_el);

if (el != null)

{

ElTree el\_father = myTree.Find\_Father(myTree.T, el.inf);

Console.WriteLine($"Eлемент {el.inf} знайдено успішно! Батько: {el\_father?.inf ?? "Немає батька"} ");

}

else Console.WriteLine("Елемент не було знайдено");

PressEnter(); break;

case 6:

string res = "";

myTree.TreeInString(myTree.T, ref res);

Console.WriteLine("Дерево -> " + res);

PressEnter(); break;

case 7:

myTree = Initalized\_tree();

Console.WriteLine("Process OK!");

PressEnter(); break;

}

}

catch (Exception ex) { Console.WriteLine(ex.Message); }

}

}

public static Tree Initalized\_tree()

{

Tree myTr = new Tree("Керівник"); //створюємо кореневий елемент

ElTree temp = myTr.T;

myTr.Add(temp, "Підрозділ 1", 'L');

temp = temp.L;

myTr.Add(temp, "Відділ виробництва", 'L');

myTr.Add(temp, "Підрозділ 2", 'R');

temp = temp.L;

myTr.Add(temp, "Відділ якості", 'R');

temp = temp.R;

myTr.Add(temp, "Відділ безпеки", 'R');

temp = myTr.T.L.R;

myTr.Add(temp, "Відділ маркетингу", 'L');

myTr.Add(temp, "Підрозділ 3", 'R');

temp = temp.L;

myTr.Add(temp, "Відділ продажів", 'R');

temp = temp.R;

myTr.Add(temp, "Відділ реклами", 'R');

temp = myTr.T.L.R.R;

myTr.Add(temp, "Відділ фінансів", 'L');

temp = temp.L;

myTr.Add(temp, "Відділ бухгалтерії", 'R');

temp = temp.R;

myTr.Add(temp, "Відділ інвестицій", 'R');

return myTr;

}

public static ushort Input\_range(string text, ushort up\_range, ushort down\_range)

{

while (true)

{

ushort input;

try

{

Console.Write(text);

input = Byte.Parse(Console.ReadLine());

if (input > up\_range || input < down\_range)

throw new Exception($"Value should be in range {down\_range}-{up\_range}.");

}

catch (Exception ex)

{

Console.WriteLine($"{ex.Message}");

continue;

}

Console.WriteLine();

return input;

}

}

public static void PressEnter()

{

while (true)

{

Console.WriteLine("To continiue press Enter...");

var key = Console.ReadKey(intercept: true);

if (key.Key == ConsoleKey.Enter)

{

Console.Clear();

break; // Выход из цикла, если нажата клавиша Enter

}

else Console.WriteLine("Pressed another key");

}

}

#region reference

/\*

- Начальник(Керівник)

- Підрозділ 1

- Відділ виробництва

- Відділ якості

- Відділ безпеки

- Підрозділ 2

- Відділ маркетингу

- Відділ продажів

- Відділ реклами

- Підрозділ 3

- Відділ фінансів

- Відділ бухгалтерії

- Відділ інвестицій

Керівник

/ | \

П1 П2 П3

/|\ /|\ /|\

В Я Б М П Р Ф Б І

\*/

#endregion

}

}

ДОДАТОК Б

Машинний лістинг створеного класу

TreeClass.cs

namespace ASD\_Lab\_4

{

public class ElTree

{

public ElTree L, R; //покажчик на піддерева

public string inf;

public ElTree(string inf)

{

this.inf = inf;

this.L = null; // Початково обидва піддерева пусті

this.R = null;

}

}

public class Tree

{

public ElTree T;

public Tree(string inf)

{

T = new ElTree(inf);

}

public ElTree Find(ElTree V, string inf) //пошук вузла з інформацією

{

if (V != null)

{

if (V.inf == inf) return V;

ElTree temp = Find(V.L, inf);

if (temp != null)

return temp;

return temp = Find(V.R, inf);

}

return null;

}

public ElTree Find\_Father(ElTree V, string inf)//пошук предка вузла з інформацією

{

if (V != null)

{

if ((V.L != null) && (V.L.inf == inf)) return V;

if ((V.R != null) && (V.R.inf == inf)) return V;

//return Find\_Father(V.L, inf);

ElTree temp = Find\_Father(V.L, inf);

if (temp != null)

return temp;

return Find\_Father(V.R, inf);

//return temp = Find(V.R, inf);

}

return null;

}

public bool Add(ElTree V, string inf, char S)//додавання вузла з інформацією вліво/вправо s=L/R.

{

ElTree temp = new ElTree(inf);

switch (S)

{

case 'L':

if (V.L == null)

{

V.L = temp;

return true;

}

else return false;

case 'R':

if (V.R == null)

{

V.R = temp;

return true;

}

else return false;

}

return false;

}

public bool Remove(ref ElTree V, string inf)//видалення вузла з інформацією

{

ElTree father;

ElTree temp = Find(V, inf);//шукаємо вузол

if (temp == null) return false; //вузла не існує

father = Find\_Father(V, inf); //шукаємо предка

if (father == null)

{

//видаляємо корінь

V = null;

return true;

}

if ((father.L != null) && (father.L.inf == inf))

{

//видаляємо лівого потомка

father.L = null;

return true;

}

//видаляємо правого потомка

father.R = null;

return true;

}

public void TreeInString(ElTree V, ref string s)

{

if (V == null)

return;

Stack<ElTree> stack = new Stack<ElTree>();

stack.Push(V);

Stack<string> outputStack = new Stack<string>(); // Для збереження виводу у зворотньому порядку

while (stack.Count > 0)

{

ElTree current = stack.Pop();

outputStack.Push(current.inf); // Додаємо поточний вузол до стеку виводу

if (current.L != null)

stack.Push(current.L);

if (current.R != null)

stack.Push(current.R);

}

// Виводимо вміст стеку у зворотньому порядку до рядка

while (outputStack.Count > 0)

s += outputStack.Pop()+" ";

}

public bool TreeExists()

{

return T != null;

}

}

}